SAP®

ABAP® – Die Programmiersprache des SAP®-Systems R/3®

PROFESSIONELL

PROFESSIONELL

PEARSON Education

ADDISON-WESLEY

Bernd Matzke

ABAP®

Die Programmiersprache des SAP®-Systems R/3®

4., erweiterte Auflage

ADDISON-WESLEY

An imprint of Pearson Education Deutschland

München • Boston • Harlow, England • Don Mills, Ontario
Sydney • Mexico City • Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei
Der Deutschen Bibliothek erhältlich.

Umwelthinweis:
Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1

04 03 02

ISBN 3-8273-1960-9

Einbandgestaltung:	Barbara Thoben, Köln
Lektorat:	Rolf Pakendorf, rpakendorf@pearson.de
Korrektorat:	mediaService, Siegen
Herstellung:	Philipp Burkart, pburkart@pearson.de
CD Mastering:	Gregor Kopietz, gkopietz@pearson.de
Satz:	mediaService, Siegen
Druck und Verarbeitung:	Berker, Kevelaer

Printed in Germany

Inhaltsverzeichnis

1 Einführung

Die Programmiersprache ABAP ist untrennbarer Bestandteil der SAP-R/3-Systems. Abgesehen von einem in C programmierten Systemkern besteht das gesamte System mit seinen Entwicklungswerkzeugen und Anwendungen aus ABAP-Programmen. Dies sichert eine betriebssystemunabhängige Entwicklung. Allerdings muss sich der Programmierer, der sich mit der Sprache ABAP beschäftigt, auf einige Eigenschaften dieses Werkzeuges einstellen, die eine Einarbeitung nicht unwesentlich erschweren.

Die Sprache ABAP hat ihren Ursprung bereits im System R/2. Dort ermöglichte ein so genannter Allgemeiner Berichts-Aufbereitungs-Prozessor die Erstellung von Drucklisten mittels einer einfachen Programmiersprache. Diese zunächst nur zum Reporting eingesetzte Sprache wurde nach und nach zu einem vollwertigen Entwicklungswerkzeug erweitert. Daraus resultiert auch die heute gebräuchliche Bezeichnung Advanced Business Application Language. Mit der Aufnahme objektorientierter Sprachelemente wird statt der Bezeichnung ABAP/4 der allgemeinere Begriff ABAP oder ABAP-OO benutzt.

Die Programmiersprache ABAP ist keine Sprache, die von einer Einzelperson oder einem kleinen Expertenkreis am grünen Tisch entworfen wurde. Ausgehend von den ersten Versionen im System R/2 wurde und wird diese Sprache beständig weiterentwickelt. Diese Entwicklung wird von einem größeren Team durchgeführt, wobei einzelne Teilbereiche (z.B. Typkonzept, Datenbankanbindung, Gestaltung der Bedienoberfläche) relativ unabhängig voneinander bearbeitet werden. Von jeder Version zur nächsten halten somit neue Merkmale Einzug in die Sprache, die Entfernung veralteter Bestandteile ist aus Gründen der Kompatibilität allerdings nicht oder nur in sehr seltenen Fällen möglich. Zwangsläufig wird der Umfang der Sprache immer größer, das Werkzeug selbst immer mächtiger, aber auch unübersichtlicher.

Die Sprache ABAP gehört nicht wie C oder Pascal zu den universell einsetzbaren Programmiersprachen. Das R/2- und R/3-System benutzen bestimmte Konzepte zur Gestaltung von Programmen, z.B. das Dynpro-Konzept. Dieses legt den Aufbau einer Anwendung in relativ engen Grenzen fest. Die Bestandteile der Programmiersprache ordnen sich diesem Konzept unter. ABAP besteht somit aus Anweisungen, die speziell für einen genau definierten Programmtyp mit vorgegebener Funktionalität und Struktur entworfen wurden. In Laufe der Zeit, vor allem beim Übergang auf grafische Benutzeroberflächen und die Realisierung einer Client-Server-Architektur, musste die Programmiersprache flexibler werden. Um die bestehenden Kommandos herum wurden neue Kommandos geschaffen oder vorhandene mit neuen Optionen erweitert, wodurch einige von ihnen eine völlig neue Funktionalität erhalten. Oft sagt eine Option zu einem Kommando mehr über dessen Aufgabe aus als der eigentliche Name der Anweisung. Einige der so geschaffenen Kommandovarianten erfüllen eine sehr scharf abgegrenzte, zum Teil systemnahe Aufgabe. Sie sind daher nicht unbedingt von allgemeinem Interesse, sondern nur von Spezialisten verwendbar.

Es existiert kein Standard, in dem der Sprachumfang von ABAP festgeschrieben wurde. Die Sprache ist ihr eigener Standard, der sich von Version zu Version verändert. Es ist daher weder möglich noch sinnvoll, eine allumfassende Einführung in die Syntax dieser Sprache zu geben. Da sich die Sprache einem Programmierkonzept unterordnet, ohne dessen Kenntnis die Bedeutung vieler Kommandos nicht ersichtlich ist, muss eine Einführung in die ABAP-Programmierung von den Programmierkonzepten bzw. Modellen ausgehen. Obwohl mit dem aktuellen Release 6 völlig neue Prinzipien Einzug in das SAP-System halten, dürfen die herkömmlichen Bestandteile des Systems nicht vernachlässigt werden. Im SAP-System werden alle Programmierkonzepte parallel benutzt, zumindest für einen längeren Zeitraum existieren alte und neue Programmiermodelle nebeneinander. Es ist daher notwendig, sich mit allen Aspekten der ABAP-Programmierung auseinander zu setzen, um das System beherrschen zu können.

Dieses Buch soll nicht als Alternative zu den vielfältigen Dokumentationen des Systems oder zu den Lehrgängen des Hauses SAP verstanden werden. Während meiner eigenen Arbeit habe ich erfahren, dass es einem Einsteiger relativ schwer fällt, die Fülle von Fakten zum R/3-System oder zur Programmiersprache zu kanalisieren und die für die eigene Arbeit wesentlichen Punkte herauszufiltern. Ich halte es daher für problematisch, ohne Programmiererfahrung in die R/3-Programmierung einsteigen zu wollen. Ich habe zwar versucht, die Lücke zwischen allgemeinen Einführungen wie Buck-Emden[1] und den sehr speziellen Systemdokumentationen zu schließen. Dieses Buch soll aber vor allem Anwendern mit DV-Erfahrung den schnellen Einstieg in die Programmierung mit

1. Buck-Emden, Rüdiger: Die Technologie des SAP-Systems: Basis für betriebswirtschaftliche Anwendungen, 4. Aktualisierte und erweiterte Auflage, München, Addison-Wesley 1998

ABAP ermöglichen. Es wendet sich insbesondere an Betreuer von SAP-Systemen und potenzielle Anwendungsentwickler. Es ist nicht als Lehrbuch für Programmier-Einsteiger gedacht, denn der Umfang des Stoffs macht es unmöglich, außer der Einführung in ABAP auch noch eine allgemeine Einführung in die Programmierung zu geben. Grundkenntnisse zur Programmierung und zu relationalen Datenbanken werden daher vorausgesetzt.

Aufbauend auf die Programmiersprache ABAP existieren im R/3-System eine Vielzahl von Bibliotheken (hier Funktionsgruppen genannt) oder vorgefertigte Objekte, mit denen Spezialaufgaben erfüllt werden. Genannt seien als Beispiel die Batch-Input-Verarbeitung, die Textverarbeitung, GUI-Controls und andere. Kenntnisse zu diesen Elementen sind ebenfalls oft erforderlich, wenn anspruchsvollere Anwendungen erstellt werden sollen. Aus Umfangsgründen ist es allerdings nicht möglich, in diesem Buch intensiv auf dieses Themengebiet einzugehen. In diesem Zusammenhang sei auf Riekert[2] verwiesen.

1.1 Zum Inhalt des Buches

Die vierte Auflage dieses Buches baut hinsichtlich des Grundkonzeptes natürlich auf den vorangegangen Auflagen auf. Allerdings hielten mit der Version 6 eine Reihe von neuen Konzepten Einzug in das System, die sich in erheblichem Maße auf die Entwicklungsumgebung und die Programmentwicklung auswirken. Ältere Versionen werden nur noch in Ausnahmefällen erwähnt.

Die Entwicklungsumgebung verfügt über eine Reihe von Bestandteilen und definiert einige Begriffe. Deren Kenntnis ist für das Verständnis des Gesamtsystems von entscheidender Bedeutung. In Kapitel 2 werden die wichtigsten Elemente des Entwicklungssystems und die häufig benutzten Begriffe vorgestellt. In diesem Kapitel wird nicht auf die technischen Details eingegangen. Es enthält lediglich Erläuterungen zu den grundlegenden Aufgaben der beschriebenen Elemente.

Kapitel 3 geht auf die unterschiedlichen Programmarten und deren Eigenschaften ein. In diesem Zusammenhang werden auch die für diese Programmarten typischen ABAP-Anweisungen vorgestellt. Der Schwerpunkt in diesem Kapitel liegt nicht auf der vollständigen Beschreibung der diversen Kommandos, sondern auf der Demonstration der Aufgaben der ABAP-Anweisungen innerhalb des jeweiligen Programms. Trotz dieser Einschränkungen reichen die vermittelten Informationen zu den ABAP-Kommandos aus, um bestehende Standardanwendungen zu analysieren und selbst neue Programme schreiben zu können. Das Kapitel 3 beginnt mit einem kurzen Tutorial, das Ihnen den Umgang mit

2. Riekert, Rainer: ABAP-Programmierung: Fortgeschrittene Programmiertechniken für ABAP, München, Addison Wesley 2001

den wichtigsten Entwicklungswerkzeugen vorstellt. Es schließt sich ein Abschnitt an, der übergreifende Konzepte und allgemein verwendbare Kommandos beschreibt. Es folgt die Beschreibung der so genannten Oberfläche, die z.B. die Menüs einer Anwendung enthält. In den darauf folgenden Abschnitten wird einer der beiden grundlegenden Programmtypen, der Report, detailliert beschrieben. Daran schließt sich ein Abschnitt zur Dialogprogrammierung an.

Beide Programmiermodelle können auch miteinander verknüpft werden. Im Anschluss an die Erläuterungen zur Dialogprogrammierung finden Sie dementsprechende Hinweise.

Es folgt ein Abschnitt, in dem die Funktionsbausteine beschrieben werden. Diese Bausteine erlauben die Verwendung von Programmcodes von mehreren anderen Anwendungen aus. Das SAP-System stellt eine Vielzahl von Bausteinen bereit, die Sie in eigenen Anwendungen benutzen können. Die beiden abschließenden Abschnitte demonstrieren die Anwendung SAP-eigener Funktionsbausteine. Im Mittelpunkt steht dabei die Formularverarbeitung mit SAPScript.

Bereits seit Release 4 gehören objektorientierte Sprachelemente zur Sprache ABAP. Inzwischen ist dieser Zweig der Programmiersprache in den Vordergrund gerückt. Leistungsfähige Klassenbibliotheken ermöglichen es, sowohl Funktionalität als auch Oberfläche einer Anwendung auf völlig neue Art und Weise zu realisieren. Das Kapitel 4 beschreibt zunächst die prinzipiellen Kommandos von ABAP OO. Anschließend werden die Werkzeuge zur Erstellung und Verwaltung von Klassen vorgestellt. Es schließen sich einige Beispiele zur Verwendung existierender Klassenbibliotheken an, wobei die dynamischen Dokumente im Mittelpunkt stehen.

Neben der reinen Programmierung ist es zur Erstellung von Anwendungen unter anderem auch erforderlich, Datenbanktabellen anzulegen. Diese Tabellen gehören zu den so genannten Data-Dictionary-Elementen, die im Mittelpunkt des fünften Kapitels stehen.

In Kapitel 6 wird näher auf einige Hilfsmittel zur Programmierung eingegangen. Dabei handelt es sich insbesondere um das Korrektur- und Transportwesen, das zur Weiterleitung von Anwendungen und zur Dokumentation von Änderungen benutzt wird, und um den Online-Debugger.

Während in Kapitel 3 vor allem die Programmierkonzepte im Mittelpunkt stehen, beschreibt Kapitel 7 die Erstellung einer einfachen Anwendung. Hier wird demonstriert, wie aus den Einzelkomponenten funktionsfähige Anwendungen entstehen können.

Im Mittelpunkt des Kapitels 8 stehen einige Tipps, Tricks und Fallen. Die Analyse dieser Tipps, die allerdings Vorkenntnisse erfordert, vertieft das Verständnis für die Arbeitsweise einiger ABAP-Kommandos.

Den Abschluss des Buches bildet eine Kurzreferenz der aktuellen ABAP-Kommandos. In dieser Referenz fehlen veraltete Kommandos, die zwischenzeitlich durch andere ersetzt wurden. Ebenso fehlen einige Kommandos, die nur für den internen Gebrauch bei SAP gedacht sind.

Noch ein Wort zu den Programmbeispielen. Das SAP-System benutzt zur Speicherung von Daten eine relationale Datenbank. Nahezu alle Beispiele benutzen Datenbanktabellen. Um dem Leser die Mühe zu ersparen, vor der eigentlichen Programmierung Tabellen anzulegen und mit Daten zu füllen, benutze ich überwiegend bereits existierende Tabellen des SAP-Standards. Auf ein durchgängiges Beispiel in den Programmierkapiteln habe ich bewusst verzichtet. Die Programmiersprache ABAP ist so komplex und vielfältig, dass ein schrittweise ausgebautes Beispiel, das alle Facetten der ABAP-Programmierung demonstrieren soll, sehr schnell ebenso komplex und damit schwer verständlich wird. Spezielle, auf das konkrete Thema zugeschnittene Beispiele erleichtern das Verständnis und erlauben es, gezielt auf bestimmte Eigenheiten der Programmiersprache einzugehen.

1.2 Übungen

Eine Programmiersprache erlernt man am besten durch Analyse vorhandener Programme und durch eigene Übungen. Im Buch werden Sie daher an vielen Stellen Übungsaufgaben finden. Solange der Umfang der vermittelten Kenntnisse noch gering ist, befinden sich die Übungen am Schluss eines Abschnitts. Sie dienen vorrangig der Festigung der vermittelten Informationen. Später werden die Übungsaufgaben in den Text eingestreut. Beispiele zur Verwendung neuer Anweisungen werden teilweise recht komplex, so dass sie nicht immer vollständig abgedruckt werden können. Ihre Aufgabe wird es in solchen Fällen sein, aufbauend auf vorhandene Kenntnisse und vorhandene Programme eine Testumgebung für die neuen Anweisungen oder Objekte zu schreiben. Dabei festigen Sie Kenntnisse aus vorangegangenen Abschnitten und gewinnen gleichzeitig mehr Sicherheit und Erfahrung im Umgang mit der Entwicklungsumgebung. Auch wenn im Text nicht ausdrücklich darauf hingewiesen wird, sollten Sie für alle abgedruckten Programmfragmente eigene Programme schreiben und testen.

1.3 Namenskonventionen

Das SAP-System kann auf unterschiedliche Weise konfiguriert werden. Unter anderem wird zwischen SAP-Entwicklungssystemen (nur direkt bei SAP anzutreffen), zwischen Entwicklungssystemen bei Partnerfirmen der SAP und Kundensystemen unterschieden. Diese Systeme unterscheiden sich beispielsweise bezüglich der zulässigen Namensräume für Entwicklungsobjekte (Programme, Tabellen etc.). Die Beispiele in diesem Buch wurden komplett in einem Kundensystemen erstellt. Falls Sie diese Beispiele in einem anders eingerichteten System

nachvollziehen, werden Sie an einigen Stellen Hinweise bezüglich der Verletzung von Namenskonventionen erhalten. Die gewünschten Objekte können üblicherweise nach Bestätigen der Warnung trotzdem angelegt werden. Bei größeren Schwierigkeiten wenden Sie sich bitte an Ihren Systemverantwortlichen, der Ihnen Auskunft über die Art Ihres Systems und die damit verbundenen Auswirkungen bezüglich der Namenskonventionen geben kann. In vielen Fällen hilft bereits die Online-Hilfe weiter, die in den diversen Warnhinweisen mit der Drucktaste Hilfe erreichbar ist.

2 Die Elemente der Entwicklungsumgebung

Eine mit ABAP geschriebene Anwendung besteht aus mehreren Elementen unterschiedlichen Typs, so genannten Entwicklungsobjekten. Dazu gehört natürlich der eigentliche Quelltext, aber auch andere Elemente wie Menüs oder Eingabemasken. Diese Objekte erfüllen jeweils spezielle Aufgaben. Zwischen ihnen bestehen daher vielfältige Beziehungen. Außerdem sind, von Ausnahmefällen abgesehen, für jede Anwendung mehrere Arten von Entwicklungsobjekten erforderlich. Selbst einfache Anwendungen erfordern somit eine relativ umfangreiche Vorarbeit und das Erzeugen mehrerer verschiedener, aber aufeinander abgestimmter Entwicklungsobjekte. Die ABAP-Programmierung unterscheidet sich aus diesem Grund erheblich von der Programmierung mit herkömmlichen Programmiersprachen der 3. Generation wie z. B. C oder Pascal. Vor der detaillierten Besprechung der Elemente einer ABAP-Anwendung ist es daher zweckmäßig, die Aufgaben der verschiedenen Objekte, ihre Eigenschaften und Beziehungen untereinander zunächst verbal zu beschreiben, ohne detailliert auf die Programmiersprache ABAP einzugehen.

2.1 Das Data Dictionary

Ein Computer bzw. ein Programm erzeugt und verarbeitet Daten. Diese können zwar auch temporärer Natur sein, wesentlich häufiger handelt es sich aber um Datenbestände, die dauerhaft gespeichert werden müssen, also auch über Betriebspausen der Hardware hinweg. Eine der wichtigsten Aufgaben eines Betriebssystems besteht daher in der Verwaltung des Massenspeichers, der üblicherweise aus einer oder mehreren Festplatten besteht. Dadurch ist zwangsläufig die Arbeit mit dem Massenspeicher eines Systems stark vom Betriebssystem abhängig. Der Anwender ist gezwungen, mit Datei- und Verzeichnisnamen zu arbeiten, für die unter verschiedenen Systemen voneinander abweichende Konventionen zu beachten sind. Weitere Unterschiede betreffen z. B.

den Aufbau des Dateisystems, spezielle Eigenschaften von Dateien sowie Zugriffsverfahren. Unabhängig davon organisieren verschiedene relationale Datenbanksysteme speziell reservierte Bereiche auf dem Massenspeicher in eigener Regie, wobei jedes System wiederum eine eigene Strategie verfolgt. Der Zugriff auf diese Daten kann nur mit Hilfe des Datenbanksystems erfolgen. All diese Umstände erschweren es, betriebssystemunabhängige Anwendungen zu erstellen oder gar Werkzeuge zu entwickeln, mit denen ebenfalls unabhängig vom konkreten Betriebssystem derartige Anwendungen erstellt werden können.

Das R/3-System greift nicht direkt auf die Dienste des Betriebssystems zurück, sondern benutzt für die eigentliche Speicherung von Datenbeständen ein relationales Datenbanksystem. Alle Informationen, nicht nur die echten betriebswirtschaftlichen Daten, sondern auch Programme, Dynpros, Menüs und andere Elemente, werden durch das Datenbanksystem in Tabellenform abgespeichert. Um die kurz angerissenen Probleme der Systemabhängigkeit zu umgehen, muss das R/3-System eine allgemeine und systemunabhängige Schnittstelle zum Datenbestand bereitstellen. Diese Schnittstelle ist das Data Dictionary.

Über das Data Dictionary werden alle Informationen völlig plattformunabhängig gespeichert. Die Anwendungen und damit die Programmierer müssen sich nicht mehr um den konkreten Aufbewahrungsort, Laufwerks- und Verzeichnisnamen oder Ähnliches kümmern. Diese eher physischen Ordnungskriterien werden abgelöst durch eine Einordnung nach logischen Gesichtspunkten wie Zugehörigkeit zu einer Anwendung, einem in der Hierarchie höher stehenden Objekt oder einer Tabelle.

Das Data Dictionary ist damit eine virtuelle Datenbank, deren Funktionalität über die eines der üblichen relationalen Datenbanksysteme hinausgeht. Insbesondere erfüllt es folgende Aufgaben:

- Schaffen einer allgemeinen Schnittstelle zum Datenbanksystem.
- Bereitstellen von Metadaten über die eigentlichen Datenbanktabellen.
- Beschreibung von allgemeinen Datentypen.
- Bereitstellen von allgemeinen Werkzeugen zur Datenbearbeitung.

Das Data Dictionary umfasst verschiedene Elemente, die an unterschiedlichen Stellen einer Anwendung genutzt werden. Diese Elemente werden relativ unabhängig voneinander gepflegt, wobei aber gewisse Abhängigkeiten zu beachten sind. Die bedeutendsten dieser Elemente werden im Folgenden etwas näher beschrieben.

2.1.1 Tabellen

Das zentrale Element einer datenbankorientierten Anwendung sind die in Tabellen gespeicherten Datensätze. Ein oder mehrere Felder bilden einen Datensatz (auch Tupel genannt), der die Eigenschaften eines realen Objekts (einer Entität) beschreibt. Beliebig viele gleichartige Datensätze bilden eine Tabelle.

Beim Einsatz relationaler Datenbanksysteme werden Informationen so aufbereitet, dass möglichst keine redundanten Daten gespeichert werden müssen. Dies bedingt eine entsprechende Datenmodellierung, in deren Folge üblicherweise mehrere logisch zusammengehörige Tabellen entstehen. Die Menge aller zusammengehörigen Tabellen wird als Datenbank bezeichnet, die auf dem Massenspeicher des Systems abgelegt wird. Zur Verwaltung dieser Tabellen dient ein relationales Datenbanksystem, das üblicherweise mit der Abfragesprache SQL arbeitet.

In R/3 ist dieses Datenbanksystem mit den Tabellen der Datenbank, von Ausnahmen abgesehen, nur über die Schnittstelle des Data Dictionary zugänglich. Das Data-Dictionary kann die zu bearbeitenden Daten auf der Datenbank in einer anderen Form ablegen, als sie über die Schnittstelle für den Anwender verfügbar sind. Der eigentliche Grund für diese Transformierung ist die Tatsache, dass die verwendeten Datenbanksysteme gewissen Beschränkungen hinsichtlich der Anzahl der Tabellen und der Felder in diesen Tabellen unterliegen. Das Data Dictionary sorgt durch spezielle Speichermethoden für die Entschärfung dieses Problems. Der Preis dafür besteht vor allem im Verzicht auf die Ausnutzung spezieller Eigenschaften der verschiedenen Datenbanksysteme.

Für den überwiegenden Teil der Tabellen stellt das Data Dictionary eine völlig transparente Schnittstelle zur Datenbank bereit. Dies bedeutet, dass alle Felder einer Tabelle des Dictionary einem Feld in der realen Datenbanktabelle entsprechen. Die im Dictionary sichtbare Datenstruktur entspricht damit völlig der Struktur der vom Datenbanksystem erzeugten Tabelle. Die jeweiligen Tabellen heißen daher transparente Tabellen.

Neben den einfachen transparenten Tabellen, die durch das Dictionary unterschiedlich behandelt werden, existiert auch noch eine Sonderform. Es handelt sich dabei um Tabellen, die mit speziellen ABAP-Anweisungen zusammenarbeiten können, um so genannte Datencluster zu speichern. Unter einem Datencluster versteht man eine Gruppe von beliebigen Datenobjekten. Unterschiedlich aufgebaute Datencluster können in ein und derselben Tabelle gespeichert werden. Die entsprechenden Tabellen werden mit einem vorgegebenen Aufbau als transparente Tabellen im Dictionary angelegt.

Neben diesen transparenten Tabellen kannten frühere Versionen des R/3-Systems noch einige weitere Tabellenarten, die inzwischen allerdings ihre Bedeutung verloren haben. Sie sollen daher auch nicht mehr beschrieben werden.

2.1.2 Datentypen

Neben den Datenbanktabellen speichert das Data Dictionary auch Beschreibungen zu Datenstrukturen, die nicht permanent in einer Datenbank abgelegt werden, sondern lediglich zur Laufzeit zur Beschreibung von programminternen Daten gedacht sind. In früheren Versionen des R/3-Systems waren das lediglich die so genannten Strukturen, auch interne Tabellen genannt. Seit der Überarbeitung des Typ-Konzepts und der Einführung objektorientierter Sprachelemente existieren mehrere Varianten von Dictionary-Typdefinitionen. Sie sind inzwischen unter der allgemeineren Bezeichnung *Datentyp* verfügbar. Die Spanne reicht dabei von einfachen Feldern (Datenelementen) bis hin zu komplexen Tabellenbeschreibungen.

2.1.3 Felder, Datenelemente und Domänen

Ein Datenbanksystem speichert Eigenschaften (Attribute) zu einem real vorhandenen Objekt (Entität) in Feldern von Datensätzen. Diese Eigenschaften können einerseits bezüglich ihrer logischen Bedeutung für den Anwender, andererseits aber auch nach ihren technischen Eigenschaften (Datentyp, Länge u. Ä.) näher beschrieben werden.

Die angestrebte Systemunabhängigkeit erfordert zunächst eine allgemeine, vom jeweiligen Datenbanksystem unabhängige Datenbeschreibung. Die unterschiedlichen Datenbanksysteme können durchaus über einen Satz unterschiedlicher Datentypen verfügen und diese intern auf völlig unterschiedliche Weise darstellen und bearbeiten. Diese Unterschiede können beispielsweise den darstellbaren Zahlenbereich betreffen oder aber die Form der Speicherung komplexerer Daten (z. B. Bilder).

Das Dictionary speichert eine Reihe von Verwaltungsinformationen, so genannte Metadaten, zu den einzelnen Tabellenfeldern. Diese sind nicht nur technischer Natur, sondern liefern auch anwendungsbezogene Informationen. Möglicherweise verfügen unterschiedliche Attribute einer Entität über gleichartige technische Eigenschaften. Andererseits können unterschiedliche Entitäten über identische Attribute verfügen. Beispielsweise können sowohl der Stammdatensatz eines Mitarbeiters als auch der Arbeitsauftrag die Personalnummer des jeweiligen Mitarbeiters enthalten sein. Beide Felder haben einen identischen Aufbau und trotz der Verwendung in verschiedenen Tabellen die gleiche Bedeutung. Es wäre nun denkbar, dass ein Feld mit dem gleichen Format existiert, welches aber eine Artikelnummer enthält. Angesichts der Komplexität des R/3-Systems ist es wünschenswert, derartige Informationen nicht getrennt von den eigentlichen Daten in einer Dokumentation o. Ä. abzulegen, sondern diese untrennbar mit den Daten zu verbinden.

Innerhalb des SAP-Systems werden daher wesentlich mehr Informationen zu den Eigenschaften von Feldern gepflegt, als das relationale Datenbanksystem

zur Bearbeitung der Tabellen benötigt. Dazu gehören neben verschiedenen technischen und anwendungsbezogenen Angaben auch zulässige Wertebereiche für dieses Feld oder aber ein Verweis auf eine Prüftabelle. Wegen der möglichen mehrfachen Verwendung derartiger Feldbeschreibungen werden sie unabhängig von Datenbanktabellen gepflegt. Dies bietet den Vorteil, Felder mit ihren Eigenschaften unabhängig von ihrer späteren Verwendung zunächst detailliert beschreiben zu können. Diese Beschreibung steht dann an verschiedenen Stellen zur Verfügung, z. B. bei der Tabellenpflege oder beim Schreiben von Eingabeprogrammen.

Für die Beschreibung eines Felds werden stets zwei verschiedene Elemente benötigt. Elementarste Grundlage für Feldbeschreibungen sind Domänen. Eine Domäne beschreibt den technischen Aufbau eines Felds, also seine Größe und seinen Datentyp. Domänen erhalten eindeutige Namen. Die Datenelemente bauen auf den Domänen auf. Sie enthalten den Verweis auf eine Domäne, um die technischen Eigenschaften zu definieren, und ergänzen diese mit Angaben zur logischen Bedeutung des Datenelements aus Sicht des Anwenders. Eine Domäne kann in mehreren Datenelementen und diese wiederum in verschiedenen Datenbankfeldern verwendet werden.

Da Datenelemente und Domänen unabhängig von der eigentlichen Tabelle existieren, können sie in mehreren Tabellen benutzt werden. Bevor im Data Dictionary eine Tabelle erzeugt werden kann, müssen Sie die für deren Datenfelder notwendigen Datenelemente zusammentragen oder erzeugen.

Zusätzlich zu den Eigenschaften, die einem Tabellenfeld über das Datenelement zugewiesen werden, existieren einige weitere Attribute, die für jedes Tabellenfeld einzeln gepflegt werden müssen. Es handelt sich dabei um die so genannten Fremdschlüssel sowie um Referenzen für Felder mit Bezug auf Währungen oder Maßeinheiten.

In praktischen Anwendungen existieren aber auch Daten, für die eine aufwändige Pflege von Domänen und Datenelementen eigentlich nicht gerechtfertigt ist. Ab Release 3.0 gibt es daher die Möglichkeit, Tabellenfelder mit direktem Bezug auf einen Datentyp, also ohne Datenelement und Domäne, anzulegen. Derartige Felder weisen aber nicht die volle Funktionalität der Felder mit Bezug auf ein Datenelement auf und sind daher nur in speziellen Fällen von Bedeutung.

Den Zusammenhang der beschriebenen Elemente soll das folgende Beispiel verdeutlichen. Gegeben sei eine Domäne mit dem Namen `NUM4`, die vier Stellen lang ist und den Datentyp `NUMC` besitzt. Sie kann also einen vier Zeichen langen Text aufnehmen, der nur aus Ziffern bestehen darf.

Diese Domäne kann zur technischen Beschreibung verschiedener Datenelemente dienen. Vor der Umstellung der Postleitzahlen in Deutschland wäre es denkbar gewesen, aufbauend auf der Domäne ein Datenelement `PLZ` zur Beschreibung einer Postleitzahl zu definieren. Diese Bedeutung wird innerhalb des Datenelements durch einen beschreibenden Kurztext definiert. Unabhängig da-

von könnte die Domäne NUM4 auch in einem anderen Datenelement, z.B. PNR, verwendet werden, das eine Personalnummer beschreibt.

Beim Anlegen einer Tabelle wird einem Feld KPLZ, das die Postleitzahl eines Kunden aufnehmen soll, nun das Datenelement PLZ als Beschreibung zugeordnet. Damit sind alle Eigenschaften dieses Felds auf einen Schlag definiert.

Das Datenelement PLZ kann auch in anderen Tabellen zur Beschreibung eines Felds dienen, das ebenfalls eine Postleitzahl aufnehmen soll. Diese Felder können (müssen aber nicht) andere Namen als KPLZ besitzen.

Der Vorteil liegt auf der Hand. Die damals recht aufwändige Umstellung der Postleitzahlen von vier auf fünf Stellen erfordert nun nicht mehr das Ändern vieler Feldbeschreibungen in unterschiedlichen Tabellen, sondern nur noch die Erzeugung einer neuen Domäne NUM5 (fünf Stellen, Typ NUMC) und das Eintragen der neuen Domäne in die Beschreibung des Datenelements PLZ. Das Data Dictionary unterstützt einen Verwendungsnachweis der verschiedenen Objekte (Domänen, Felder, Tabellen etc.). Einer durch das Data Dictionary erzeugten Liste kann nun entnommen werden, welche Tabellen Felder besitzen, die das Datenelement PLZ benutzen. Diese Tabellen wären dann lediglich inhaltlich zu ändern (Eintragen der neuen Postleitzahl mit Konvertierungsprogramm).

Die Domäne entspricht damit in etwa den Typdeklarationen der verschiedenen Programmiersprachen (typedef in C/C++ oder TYPE in Pascal). Für das Datenelement verfügen herkömmliche Programmiersprachen nicht über eine adäquate Entsprechung.

Neben der reinen Beschreibung von Feldeigenschaften ermöglichen Domänen auch die Automatisierung einiger Standardprüfungen in Anwendungen. In der Beschreibung einer Domäne kann beispielsweise der Name einer Prüftabelle oder eine Liste mit Festwerten eingetragen werden. Beim Bearbeiten eines Datensatzes, der ein auf der Domäne beruhendes Feld enthält, wird durch verschiedene Mechanismen des Systems automatisch geprüft, ob der Wert des betroffenen Tabellenfelds mit einem Wert aus der Prüftabelle oder der Festwertliste übereinstimmt. Ist dies nicht der Fall, so wird der Datensatz zurückgewiesen. Diese Prüfungen finden automatisch statt, der Programmierer muss dafür keinen zusätzlichen Code erzeugen.

2.1.4 Views

Bereits mehrfach wurde erwähnt, dass beim Einsatz relationaler Datenbanksysteme die zu speichernden Daten auf mehrere Tabellen verteilt werden, um Redundanzen zu vermeiden. Die Kopplung der zusammengehörigen Datensätze erfolgt über eindeutige Schlüsselfelder. Diese Verbindung erfolgt erst durch entsprechende Anweisungen, z.B. mehrstufige Abfragen in logisch zusammengehörenden Tabellen.

Zur Arbeitserleichterung kann eine virtuelle Tabelle angelegt werden, die Felder aus einer oder mehreren Tabellen miteinander verknüpft und dem Benutzer als eine einzige Tabelle darbietet. Da dies keine neue, real existierende Tabelle ist, sondern nur eine spezielle Darstellungsform, wird sie Sicht oder View genannt. Innerhalb des SAP-Systems existieren verschiedene Viewtypen mit unterschiedlichen Einsatzgebieten. Für eigene Anwendungen sind insbesondere die Datenbank- und die Customizing-Views von Interesse. Views werden nicht nur manuell durch den Programmierer angelegt, sondern auch vom System automatisch generiert. Das geschieht z. B. beim Erzeugen einer Pflegeoberfläche für Tabellen.

Datenbank-Views

Datenbank-Views werden als echte Views des Datenbanksystems realisiert, können also nur transparente Tabellen aufnehmen. Sie bieten eine Zusammenfassung von Tabellen. Sie können nicht editiert werden, wenn sie Felder aus mehr als einer Tabelle enthalten. Derartige Views eignen sich besonders, um Tabellen auf einfache Weise mit Daten zu füllen, ohne dass dazu eigene Programme geschrieben werden müssen.

Customizing-Views

Customizing-Views ermöglichen das Editieren von Datensätzen, auch wenn mehrere Tabellen in die View aufgenommen werden. Beim Erzeugen eines derartigen Views generiert das System Pflegeprogramme, die über eine spezielle Transaktion aufgerufen und ausgeführt werden können. Wie der Name schon sagt, werden Customizing-Views insbesondere beim Customizing eingesetzt. Das Customizing ist ein Vorgang, bei dem die Anwendung durch Eintragen von Vorgabewerten in ausgewählte Tabellen an die konkreten Erfordernisse des Anwenders angepasst werden kann.

Streich-Views

Streich-Views bieten eine Sicht auf eine einzige Tabelle, wobei nicht interessierende Felder ausgeblendet (gestrichen) werden können.

Help-Views

Help-Views wurden geschaffen, um spezielle Hilfsinformationen anzeigen zu können. Sie haben nur innerhalb des Hilfesystems eine Bedeutung.

Entitäts-Views

Entitäts-Views sind nur für das Unternehmensdatenmodell von Bedeutung.

2.1.5 Sperrobjekte

Während der Arbeit mit einer Datenbank soll diese üblicherweise gegen Zugriffe fremder Anwender gesperrt werden, da gleichzeitiger schreibender Zugriff mehrerer Anwender auf ein und dieselbe Tabelle unter Umständen einen undefinierten Zustand der Daten zur Folge haben kann.

Während der Bearbeitung einer Tabelle können Datensätze einzeln oder in Gruppen durch Aufruf einiger Funktionen gegen Fremdzugriffe gesperrt und auch wieder freigegeben werden. Um die Anwendung dieser Sperrfunktionen zu erleichtern und die einfache Änderbarkeit bei Änderungen der zu Grunde liegenden Datenbanktabelle zu ermöglichen, verwendet das R/3-System so genannte Sperrobjekte. Diese werden unabhängig von einer konkreten Anwendung gepflegt. Zum Erzeugen von Sperrobjekten ist keinerlei eigene Programmierarbeit erforderlich. Alle erforderlichen Informationen werden durch den Programmierer in einige Bildschirmmasken eingetragen. Nach der vollständigen Definition des Sperrobjekts generiert das System zwei Funktionsbausteine, die alle erforderlichen Aktivitäten zum Sperren oder Entsperren (Freigeben) der jeweiligen Tabellen bzw. ausgewählter Datensätze ausführen. Diese Funktionsbausteine können von allen Anwendungen aufgerufen werden, welche die Tabelle bearbeiten wollen.

Ändern sich Rahmenbedingungen bezüglich des Sperrwunsches, so wird lediglich das Sperrobjekt bearbeitet und anschließend die Neugenerierung der Funktionsbausteine durchgeführt. Änderungen im Quelltext der Anwendungen, die dieses Sperrobjekt bzw. die Funktionsbausteine benutzen, können meist entfallen. Falls sie doch erforderlich werden, unterstützt auch hier ein per Menüfunktion erzeugbarer Verwendungsnachweis die Änderungen.

2.1.6 Typgruppen

Beginnend mit Version 3 der R/3-Software wurde das Typkonzept modernisiert. Die Eigenschaften von Datenfeldern können nun auch durch frei definierbare Datentypen beschrieben werden, die unabhängig von den bereits beschriebenen Datenelementen existieren und nicht mit diesen identisch sind. Derartige Definitionen können in global verfügbaren Typgruppen abgelegt und von jedem Programm benutzt werden. Da auch mit diesen Typdefinitionen Informationen über Daten bereitgestellt werden, werden die Typgruppen ebenfalls vom Data Dictionary verwaltet.

2.2 Berechtigungen

Mit einem SAP-System arbeiten gewöhnlich mehrere Anwender unterschiedlicher Fachgebiete mit unterschiedlichen Aufgaben. Dies erfordert es, wirksame Mechanismen gegen unautorisierten Zugriff zu installieren. Das SAP-System stellt dafür ein sehr flexibles Verfahren zur Verfügung, das vor allem auf der expliziten Berechtigungsprüfung innerhalb einer Anwendung beruht. Vor Ausführen einer Aktion, z. B. vor dem Start eines Programmzweiges, werden die Berechtigungen des jeweiligen Anwenders geprüft. Dies erfolgt, von Ausnahmen abgesehen, nicht automatisch, sondern erfordert spezielle Anweisungen im Programm. Dieses Verfahren unterscheidet sich damit grundsätzlich von impliziten Berechtigungsprüfungen, wie sie aus Netzwerk- oder Multiuser-Betriebssystemen bekannt sind. Dort wird ja meist nur der Zugriff auf Dateien mittels einiger Zugriffsrechte, beispielsweise für Lesen, Schreiben, Ausführen oder Löschen, überwacht. Mit SAP-Berechtigungen hingegen können nicht nur Tabellen oder einzelne Felder, sondern auch andere Objekte oder Aktionen wie Programme, Reports, Programmzweige o. Ä. geschützt werden. Dabei kann beispielsweise die Bearbeitung oder das Anzeigen von Daten ebenso von vorhandenen Berechtigungen abhängig gemacht werden wie die Abarbeitung eines Programms. Anwendungen können sogar, abhängig von den vorhandenen Berechtigungen des Anwenders, von diesem unbemerkt verzweigen und jeweils unterschiedliche Aktivitäten ausführen.

Diese Flexibilität wird möglich, da Berechtigungen zunächst unabhängig vom zu schützenden Objekt erstellt und verwaltet werden. Der Bezug zum Objekt muss innerhalb der Anwendung explizit durch Test einer Berechtigung hergestellt werden. Das System selbst stellt lediglich Hilfsmittel zur Verwaltung einiger Elemente zur Verfügung, mit denen die Berechtigungen eines Anwenders definiert werden können.

Der Verzicht auf implizite Berechtigungsprüfungen besitzt allerdings auch einen Nachteil. Es ist jedem Programmierer möglich, Programme zu schreiben, die ohne Einschränkung auf nahezu alle Tabellen des Systems zugreifen und diese ungehindert lesen oder sogar manipulieren können. Auch das Recht, Programme im Debug-Modus auszuführen, kann missbraucht werden, um den Rückgabewert der Berechtigungsprüfung zu überschreiben und diese Prüfung damit unwirksam zu machen. In produktiven Systemen stellt somit jeder Benutzer, der über Berechtigungen zur Systemadministration oder zur Programmentwicklung verfügt, ein potenzielles Sicherheitsrisiko dar. Mit den entsprechenden Berechtigungen sollte daher sehr restriktiv umgegangen werden. Allerdings existieren inzwischen einige weiterführende Konzepte, bei denen der Kernel des R/3-Systems selbst einige Berechtigungen prüft.

Die Berechtigungsprüfung beruht auf einem einfachen Prinzip. Ein Benutzer erhält so genannte Berechtigungen. Diese stellen ein komplexes Objekt dar, das mehrere Berechtigungsfelder enthält, denen wiederum ein Wert zugewiesen

wird. Die Berechtigungen werden im Stammsatz des Benutzers hinterlegt. Im Programm wird dann mit einer speziellen ABAP-Anweisung geprüft, ob ein bestimmtes Feld einer Berechtigung des aktuellen Anwenders einen bestimmten Wert aufweist oder nicht. Das Ergebnis dieser Prüfung wird innerhalb des Programms explizit ausgewertet; die Prüfung selbst hat noch keine Auswirkungen auf den Programmablauf. Was konkret durch diese Berechtigung geschützt werden soll, ergibt sich also erst durch die Auswertung des Prüfergebnisses im Programm. Erst dort wird der Zusammenhang zwischen Berechtigung und zu schützendem Objekt hergestellt. Daher ist es auch möglich, mit einer Berechtigung mehrere Aktionen oder Datenobjekte gleichzeitig zu schützen. Ebenso können unterschiedliche Berechtigungen den Zugriff auf ein und dasselbe Objekt erlauben.

Diese Art der Berechtigungsprüfung erfordert im praktischen Einsatz die Verwendung vieler einzelner Felder, wenn wirklich flexible Prüfungen erforderlich sind, da möglicherweise für jedes zu schützende Objekt auch ein eigenes Berechtigungsfeld erforderlich ist. Die Felder und Berechtigungen werden daher dem Benutzer nicht einzeln zugeordnet. Vielmehr existiert eine mehrstufige Hierarchie von Elementen, mit denen Berechtigungen verwaltet und dem Benutzer zugeordnet werden.

Die erforderlichen Elemente werden zunächst unabhängig vom Benutzer definiert. Dazu wird zunächst ein Berechtigungsobjekt erzeugt, das ein oder mehrere Felder (maximal 10) besitzt. Die Berechtigungsobjekte wiederum gehen in so genannte Berechtigungsklassen ein, in denen die zu einer Anwendung oder einem Aufgabenbereich gehörenden Objekte zusammengefasst werden. Die so erzeugten Objekte stellen zunächst nur allgemeine Beschreibungen dar, entfernt vergleichbar mit den Typdefinitionen prozeduraler Programmiersprachen wie C oder Pascal. Den Feldern der Berechtigungsobjekte werden also noch keine Werte zugewiesen.

Um einem Anwender konkrete Rechte zuzuordnen, werden zunächst, wieder unabhängig vom Benutzer, von einem Berechtigungsobjekt die eigentlichen Berechtigungen abgeleitet. Innerhalb einer Berechtigung enthalten die Felder dann konkrete Werte. Eine Berechtigung erhält einen eindeutigen Namen. Von einem Berechtigungsobjekt können beliebig viele Berechtigungen abgeleitet werden, die sich durch unterschiedliche Feldwerte voneinander unterscheiden können. Verglichen mit herkömmlichen Programmiersprachen ähneln Berechtigungen den Instanzen eines Typs (dem Berechtigungsobjekt).

Mehrere Berechtigungen werden nun zu so genannten Profilen zusammengefasst, wobei mehrere derartige Profile in ein Sammelprofil eingehen können. Profile und Sammelprofile fassen logisch zusammengehörende Berechtigungen zusammen und kennzeichnen somit Berechtigungen für ein bestimmtes Aufgabengebiet, z. B. die Berechtigungen für einen Anlagenbuchhalter oder einen Personalsachbearbeiter.

Im Benutzerstammsatz werden alle Berechtigungsprofile, über die ein Benutzer verfügen soll, eingetragen. Er verfügt damit (über die Zwischenstufen Sammelprofil, Profil und Berechtigung) über Berechtigungsfelder mit Werten. Diese Werte können einfache Ja-Nein-Informationen aufnehmen. Es ist aber auch möglich, anhand des Feldwertes detaillierte Informationen weiterzugeben. Innerhalb der im SAP-System bereits vorhandenen Berechtigungen wird anhand des Wertes eines Felds beispielsweise unterschieden, ob ein Benutzer eine Tabelle einsehen, ändern oder löschen darf. Für jede dieser Aktivitäten steht eine Schlüsselzahl zur Verfügung, deren konkrete Bedeutung sich wiederum nur aus der Berechtigungsprüfung im Programm ergibt. Zur Erleichterung der Dokumentation können für die möglichen Feldwerte Erläuterungen abgespeichert werden.

Da die explizite Berechtigungsprüfung bei unsachgemäßer Anwendung eine Sicherheitslücke darstellen kann, wurden inzwischen zusätzliche Prüfungen in das System eingebaut. Sie beruhen zwar auf dem existierenden Berechtigungskonzept, allerdings erfolgt der Aufruf der Prüfung durch das R/3-System selbst, muss also nicht separat programmiert werden. Diese automatischen Prüfungen finden beim Start jeder Transaktion und beim Zugriff auf bestimmte Objekte wie z.B. Programme statt. Konkret bedeutet dies, dass einem Anwender für jede aufzurufende Transaktion eine spezielle Berechtigung zugewiesen werden muss, damit er sie starten kann.

Im Zusammenhang mit den erweiterten Berechtigungsprüfungen stehen die Berechtigungsgruppen. Berechtigungsgruppen sind Bezeichner, die in einer speziellen Tabelle angelegt werden. Diversen Objekten wie z.B. Programmen oder Transaktionen zur Tabellenpflege können Sie eine derartige Berechtigungsgruppe zuordnen. Ein Anwender kann dann nur auf dieses Objekt zugreifen, wenn in seinem Benutzerprofil in einer speziellen Berechtigung der Name der Berechtigungsgruppe enthalten ist.

2.3 Nummernkreise

In vielen Anwendungen wird der Schlüssel einer Tabelle durch eine eindeutige laufende Nummer gebildet, z.B. eine Beleg- oder Personalnummer. Die Eindeutigkeit dieser Nummern muss in jedem Fall gewährleistet sein, auch wenn mehrere Anwendungen gleichzeitig eine solche Nummer vergeben wollen. Es ist daher nicht Aufgabe der Anwendung, diese Nummer selbst zu ermitteln. Das System stellt so genannte Nummernkreise zur Verfügung. Nach Aufforderung seitens einer Anwendung (durch Aufruf spezieller Funktionsbausteine) übergibt ein Nummernkreis die nächste freie Nummer an die Anwendung. Im System können mehrere Nummernkreise existieren. Sie erhalten daher einen eindeutigen Namen, über den sie in Programmen angesprochen werden können.

2.4 Transaktionen

Transaktionen bilden den Kern der SAP-Programmabarbeitung. Sie sind die kleinste unmittelbar durch den Anwender ausführbare Einheit. Sie bilden eine Schale um das eigentliche Programm. Ausführbare Anwendungen werden für den Anwender erst zugänglich, wenn für sie eine Transaktion erzeugt wurde. Jede Transaktion wird durch einen vierstelligen Transaktionscode identifiziert. Der Name kann aus Buchstaben und Ziffern bestehen. Diese Transaktionscodes können entweder direkt im Eingabefeld der Bedienoberfläche eingegeben oder aber in den Menüs hinterlegt werden.

Durch die interne Steuerlogik, die weder dem Anwender noch dem Programmierer zugänglich ist, wird die Transaktion ausgeführt. Dabei überwacht die Steuerlogik die Anwendung vollständig, fängt z. B. auch Laufzeitfehler ab, hebt unter bestimmten Bedingungen Datenbanksperren auf oder schreibt automatisch Datenbankänderungen fest.

Für den Anwender erscheint eine Transaktion in SAP das zu sein, was man gemeinhin unter einem Programm versteht. Im SAP-System existieren nur wenige Programmtypen. Zu jedem Programmtyp gehört auch eine spezielle Transaktionsart.

2.4.1 Dialogtransaktion

Eine Dialogtransaktion ist die am häufigsten benutzte Transaktion. Sie ermöglicht die Ausführung einer dialogorientierten Anwendung. Eine derartige Anwendung beruht auf einer oder mehreren Bildschirmmasken, die in SAP-Anwendungen den Namen *Dynpro* tragen. An Stelle des Begriffs Dynpro setzt sich auch im deutschen Sprachraum immer mehr die englische Entsprechung *Screen* durch. Dynpros oder Screens sind die einzige Möglichkeit, mit dem Anwender eines Programms in einen echten Dialog zu treten.

2.4.2 Reporttransaktionen

Neben den dialogorientierten Anwendungen gibt es eine zweite bedeutende Anwendungsart, die Reports. Diese laufen, zumindest in ihrer Grundform, ohne Beeinflussung durch den Anwender ab. Sie produzieren eine Auswertungsliste. Damit derartige Reports problemlos aus dem Menü heraus gestartet werden können, muss für sie ebenfalls eine spezielle Transaktion angelegt werden. Sie verbindet den Transaktionscode mit dem jeweiligen Report.

2.4.3 Parametertransaktionen

Die bereits erwähnten Dialogtransaktionen starten eine dialogorientierte Anwendung. Alle Daten sind durch den Benutzer einzutragen. Sehr oft werden Dialogprogramme so aufgebaut, dass der Anwender im ersten Dynpro einige Werte eingibt, mit denen ein zu bearbeitendes Objekt ausgewählt wird. Alle weiteren Dynpros dienen dann zur Bearbeitung dieses Objekts.

Beispielsweise gibt es die Transaktion SM30, mit der die bereits erwähnten Views bearbeitet werden. Im Startdynpro dieser Transaktion muss der Name der View eingetragen und eine Bearbeitungsart gewählt werden. Im darauf folgenden Dynpro werden die Felder der View eingeblendet, der Anwender kann die Daten bearbeiten.

Mitunter ist es aber wünschenswert, dem Benutzer der Anwendung die Arbeit zu erleichtern oder ihm nur den Zugriff auf bestimmte Objekte zu ermöglichen. Dazu werden die Werte, die eigentlich im Startdynpro manuell einzutragen sind, automatisch übergeben. Die Anwendung startet dann gleich mit dem nachfolgenden Dynpro, in der eine View bearbeitet werden kann. Diese Funktionalität erfordert eine weitere Transaktionsart, die Parametertransaktion. Eine Parametertransaktion startet eine dialogorientierte Anwendung und übergibt ihr Vorgabewerte.

Für jede Parametertransaktion werden der Transaktionscode der zu startenden Anwendung und die zu übergebenden Parameter erfasst. Soll ein und dieselbe Anwendung mit unterschiedlichen Parametern gestartet werden, so sind dafür unterschiedliche Parametertransaktionen erforderlich.

2.4.4 Variantentransaktionen

In dialogorientierten Anwendungen findet die Kommunikation mit dem Anwender über die Dynpros (Eingabemasken) statt. Die Dynpros besitzen zwar einen statischen Aufbau, allerdings kann das Aussehen der Dynpros, z. B. durch Ausblenden von Feldern, zur Laufzeit durch Programmanweisungen modifiziert werden. Eine weitere Möglichkeit zur Anpassung eines Dynpros sind die Variantentransaktionen. Sie ermöglichen es, Varianten zu bereits existierenden Dialogtransaktionen anzulegen. In diesen Varianten können Dynprofelder ausgeblendet oder mit Standardwerten belegt werden.

2.4.5 OO-Transaktion

Über eine OO-Transaktion (Objektorientierte Transaktion) kann die Methode einer Klasse aufgerufen werden.

2.4.6 Bereichsmenü

Um die verschiedenen Transaktionen aufzurufen, existieren zwei Möglichkeiten. Zum einen kann der vierstellige Transaktionscode direkt in das OK-Feld der Oberfläche eingetragen werden. Diese Variante gestattet den schnellstmöglichen Zugriff auf eine Anwendung, falls deren Transaktionscode bekannt ist.

Eleganter und vor allem den Möglichkeiten der grafischen Bedienoberfläche besser entsprechend ist der Aufruf von Transaktionen über Menüs. Innerhalb einer Anwendung (einer Transaktion) werden Menüs in der so genannten Oberfläche hinterlegt. Über diese Menüs erfolgt der Aufruf von Teilfunktionen einer Anwendung. Für anwendungsübergreifende Menüs, also solche, aus denen heraus komplette Transaktionen aufgerufen werden, ist diese Variante nicht möglich. Dazu existieren so genannte Bereichsmenüs. Ein Bereichsmenü kann als spezielle Transaktionsart aufgefasst werden, deren einzige Aufgabe darin besteht, dem Anwender ein Menü zur Verfügung zu stellen. Ein Bereichsmenü besitzt daher einen Transaktionscode, über den es aufgerufen werden kann. Den einzelnen Menüelementen im Bereichsmenü werden Transaktionscodes zugeordnet. Wählt der Anwender ein Menüelement aus, so wird die entsprechende Transaktion ausgeführt. Diese kann eine der bereits erwähnten Transaktionen zur Ausführung einer Anwendung oder aber ein weiteres Bereichsmenü sein.

2.5 Reports

Neben den dialogorientierten Anwendungen sind Reports die zweite große Programmgruppe. In ihrer Grundform greifen Reports auf eine oder mehrere Datenbanktabellen zu und stellen deren Inhalt in Listenform aufbereitet dar. Diese Liste kann am Bildschirm eingesehen oder ausgedruckt werden. Im Gegensatz zu dialogorientierten Transaktionen können Reports direkt aus der Entwicklungsumgebung heraus abgearbeitet werden, ohne dass zusätzliche Elemente wie Dynpros oder Oberflächen erzeugt werden müssen. Man nennt sie daher auch Online-Anwendungen. Reports werden daher auch als Hilfsmittel zur Programmentwicklung und Systemwartung eingesetzt. Soll ein Endanwender einen Report ausführen können, so ist dieser wie bereits erwähnt in eine Anwendung oder eine Reporttransaktion einzubinden.

Im Prinzip sind in einem Report nahezu alle ABAP-Anweisungen sinnvoll verwendbar, sogar eine Interaktion mit dem Anwender ist möglich. Dies erfordert dann aber eine relativ aufwändige Programmierung, die den Einsatz spezieller Anweisungen und die Einbindung von dialogorientierten Elementen erzwingt. Die Programmierweise unterscheidet sich dann erheblich von der einfacher Reports ohne interaktive Elemente. Reports werden daher in Standardreports und interaktive Reports aufgeteilt. Reports entsprechen am ehesten den Programmen anderer Programmiersprachen. Da sie einfacher zu erzeugen sind als dia-

logorientierte Anwendungen, werden sie später benutzt, um die Programmiersprache ABAP vorzustellen.

2.5.1 Selektionen und Parameter

SAP-Tabellen und damit Auswertungslisten können sehr umfangreich sein. Für den Anwender ist es daher wünschenswert, vor Ausführung eines Reports die zu bearbeitende Datenmenge einschränken zu können. Vor dem eigentlichen Start kann ein Report einen Selektionsbildschirm anzeigen, in dem der Anwender Werte eintragen kann.

Ein Selektionsbildschirm ist ein automatisch generiertes Dynpro. Jeder Report verfügt maximal über einen Selektionsbildschirm. Die über den Selektionsbildschirm an den Report übergebenen Werte müssen im Report ausgewertet werden. In den Selektionsbildschirm können durch spezielle Anweisungen im Report Elemente zweier Grundtypen aufgenommen werden. Über so genannte *Selektionen* kann der Anwender komplexe Teilbereiche festlegen, z.B. Von-bis-Spannen oder eine Gruppe von Einzelwerten. Ein *Parameter* hingegen gestattet nur die Übergabe eines einzelnen Wertes.

Zwecks Arbeitserleichterung ist es möglich, verschiedene Wertzuweisungen zum Selektionsbildschirm als Varianten zum Report abzuspeichern. Dies kann auch der spätere Anwender des Reports in eigener Verantwortung vornehmen. Programmierarbeiten sind dazu nicht erforderlich. Die Varianten erhalten einen eindeutigen Namen. Sie können später jederzeit aufgerufen und ausgeführt werden.

2.5.2 Logische Datenbanken

Zwischen Tabellen in relationalen Datenbanken werden Zusammenhänge über den Primärschlüssel gebildet. Es entsteht eine mehrstufige Hierarchie von Tabellen. Beim Auswerten von Daten in einem Report muss in der Regel der Inhalt mehrerer logisch voneinander abhängiger Tabellen gelesen werden. Dazu muss der Report die Verbindung der Tabellen untereinander durch entsprechend formulierte Abfragen explizit herstellen. Sollen mehrere Reports auf ein und denselben Datenbestand zugreifen, so muss dies in jedem Report erfolgen. Das ist arbeitsaufwändig und zieht einen hohen Wartungsaufwand nach Änderungen der Datenstruktur nach sich. Abhilfe ermöglichen so genannte *Logische Datenbanken.* Das sind spezielle Programme, die mit mehreren Reports zusammenarbeiten können. Sie lesen Daten aus mehreren Tabellen, wobei sie die Hierarchie und die Verknüpfungen berücksichtigen. Mittels spezieller Anweisungen werden die gelesenen Datensätze dem eigentlichen Report zur Verfügung gestellt, der sie nur noch auswerten und gegebenenfalls anzeigen muss. Ein Report kann nur auf eine logische Datenbank zugreifen, allerdings kann eine solche Daten-

bank von mehreren Reports benutzt werden. Die realen Datenbanktabellen können in beliebig viele logische Datenbanken aufgenommen werden.

Eine logische Datenbank wird mittels eines pseudografischen Bildschirms angelegt, in dem die hierarchische Verknüpfung der einzelnen Tabellen ersichtlich ist. Aus der so vorgegebenen logischen Hierarchie wird dann ein ABAP-Programm generiert, das durch den Entwickler mit dem Programmeditor weiterbearbeitet werden muss.

2.6 Dialogorientierte Anwendungen

Neben den Reports stehen im SAP-System natürlich auch dialogorientierte Anwendungen zur Verfügung. Bis zum Release 4.6 konnten diese im Prinzip nur über den proprietären Client des R/3-Systems, den SAPGUI, ausgeführt werden. Inzwischen existieren aber auch einige Varianten, mit denen Web-basierte Dialoganwendungen erstellt werden können.

2.6.1 Dynpro-Anwendungen

Die ersten Versionen der R/3-Software benutzten zur Ausführung von dialogorientierten Anwendungen das so genannte *Dynpro*-Konzept. Der Begriff Dynpro ist historisch gewachsen. Er steht als Abkürzung für „dynamisches Programm“ und hat seinen Ursprung im System R/2. Da es sich bei einem Dynpro letztendlich um eine Bildschirmmaske handelt, werden auch die Begriffe Screen, Bildschirm oder Maske benutzt. Allerdings unterscheidet sich ein Dynpro erheblich von den Bildschirmmasken, die z.B. Betriebssysteme mit grafischer Oberfläche verwenden. Ein Dynpro ist eine Kombination einer Eingabemaske mit einemProgrammcode, der zu dieser Eingabemaske gehört. Die Maske ermöglicht die Eingabe von Daten durch den Anwender, der Programmcode verarbeitet diese Daten. Eine Dialoganwendung besteht unter anderem aus einem oder mehreren Dynpros, die nacheinander aufgerufen und abgearbeitet werden. Dynpros und zugehöriger Programmcode werden zu einem speziellen Programm zusammengefasst. Dynpros werden innerhalb eines Programms durch eine vierstellige Ziffer eindeutig bezeichnet.

Die Abarbeitung eines Dynpros verläuft in mehreren Phasen. Zunächst werden Anweisungen zur Initialisierung des Dynpros ausgeführt. Daran schließt sich die Darstellung des Dynpros auf dem Bildschirm und die Eingabe von Daten in das Dynpro an. Nach Beendigung der Eingabe wird eine zweite Anweisungsgruppe abgearbeitet, die vor allem für die Be- und Verarbeitung der eingegebenen Daten sowie die Steuerung des weiteren Ablaufs der Anwendung zuständig ist. Die Initialisierungsphase wird `PROCESS BEFORE OUTPUT` (PBO) genannt, die Phase nach der Beendigung der Dateneingabe wird mit `PROCESS AFTER INPUT` (PAI) bezeichnet. Die während dieser Phasen auszuführenden Aktionen können Sie durch Programmieren so genannter Module festlegen. Da

das Programm einer Dialoganwendung vor allem aus solchen Modulen besteht, wird es auch als Modul-Pool bezeichnet. Er ähnelt einer Programmbibliothek.

Die einzelnen Module des Modul-Pools müssen den Dynpros zugeordnet werden. Dabei ist sowohl das Dynpro als auch die korrekte Verarbeitungsphase (`PBO` oder `PAI`) festzulegen. Für diese Aufgabe ist die so genannte Ablauflogik zuständig, über die jedes Dynpro verfügt. Die Ablauflogik ist ein kurzer Quelltext, in dem Module des Modul-Pools aufgerufen werden. Die Ablauflogik ist kein ABAP-Programm im engeren Sinne. Sie benutzt andere Anweisungen als normale ABAP-Programme.

Der Aufruf einer Dialoganwendung ist nur über eine Dialogtransaktion möglich. Dieser Dialogtransaktion muss sowohl der Name des auszuführenden Programms als auch der Startpunkt bekannt sein. Da für den Ablauf einer Dialoganwendung immer der Aufruf von Dynpros verantwortlich ist, muss der Dialogtransaktion ein Dynpro als Startdynpro bekannt gemacht werden.

Neben den eben erwähnten Dynpros verfügt eine Dialoganwendung über weitere Elemente. Die Dynpros stellen lediglich Elemente zur Ein- und Ausgabe von Daten bereit. Die zur Interaktion mit dem Anwender erforderlichen Menüs und Schaltflächen (Drucktasten) sind nicht in den Dynpros enthalten, sondern in der so genannten *Oberfläche*.

2.6.2 Internet Service

Ein Internet Service stellt die einfachste Variante dar, um Dialog-Anwendungen über das Web in einem Internet Browser ausführen zu können. Dabei werden die Daten einer herkömmlichen Dialogtransaktion in ein HTML-Template eingemischt und über einen Web-Server nach außen zur Verfügung gestellt. Spezielle Meta-Anweisungen in den Templates unterstützen dabei die Übertragung von Daten sowie den Aufbau der HTML-Seite. Kennzeichnend für einen Internet Service ist im Wesentlichen, dass die komplette Anwendungslogik sowie die Ablaufsteuerung weiterhin im R/3-System liegt und mit ABAP-Programmen realisiert wird.

2.6.3 BSP-Anwendungen

BSP-Anwendungen gehen einen Schritt weiter als ein Internet Service. Hier werden ebenfalls Templates verwendet, bei denen in den HTML-Text aber ABAP-Anweisungen gemischt werden. Diese Anweisungen dienen vor allem dazu, die Methoden von Objekten (Business Objects) aufzurufen, die innerhalb des R/3-Systems programmiert werden. Diese Objekte liefern die betriebswirtschaftliche Funktionalität, während in den BSP-Templates vor allem die Ablaufsteuerung realisiert wird. Zur Laufzeit der Anwendung werden zunächst die in den BSP-Seiten enthaltenen ABAP-Anweisungen und damit auch die Methodenaufrufe der Business Objects ausgeführt. Das Ergebnis der diversen Anweisungen wird

in das umgebende HTML-Template eingemischt, wodurch schließlich reiner HTML-Text entsteht, der an den Browser gesendet wird.

Das Verfahren ähnelt stark den aus der Java-Welt bekannten JSP-Seiten, nur dass an Stelle von Java die Programmiersprache ABAP benutzt wird.

2.7 Objektorientierte Erweiterung

Zur Modellierung und Programmierung komplexer Anwendungen haben sich in vielen Fällen objektorientierte Methoden durchgesetzt. Dabei werden die Daten und die zu ihrer Bearbeitung erforderlichen Funktionen zu einem gemeinsamen Element, dem Objekt, zusammengefasst. Ein solches Objekt verbirgt seine innere Funktionalität hinter einer Schnittstelle. Der Verwender eines Objekts kann nur über die Bestandteile dieser Schnittstelle auf die Daten und Funktionen des Objekts zugreifen. Zur Laufzeit einer Anwendung können mehrere Objekte gleichen Typs existieren, die jeweils unterschiedliche Daten enthalten.

Einer der Vorteil der objektorientierten Programmierung besteht darin, dass die innere Funktionalität eines Objekts relativ unabhängig von der Schnittstelle weiterentwickelt werden kann. Die Schnittstelle wird während der Entwurfsphase vor allem nach logischen Gesichtspunkten festgelegt, während die Implementierung und die Berücksichtigung technischer Details später erfolgen kann.

Die logische und entwurfsorientierte Sicht auf Objekte erleichtert die Modularisierung großer Anwendungen und verringert so deren Komplexität. Weitere Vorteile ergeben sich bei der Programmierung ereignisgesteuerter Anwendungen wie z.B. grafischer Oberflächen.

Beginnend mit Version 4.0 bietet das R/3-System eine erste objektorientierte Erweiterung der Sprache ABAP an, die bis zur aktuellen kontinuierlich ausgebaut wurde.

Innerhalb des R/3-Systems kann die objektorientierte Spracherweiterung auf zwei unterschiedlichen Ebenen genutzt werden. Zunächst ist es möglich, lokale Objekte zu programmieren, die nur innerhalb der eigenen Anwendung bekannt und gültig sind. Die Programmierung erfolgt dabei ausschließlich durch Niederschreiben von Quelltext. Die wirklichen Vorteile der Objektorientierung ergeben sich aber erst dann, wenn Objekte bzw. deren Beschreibung global verfügbar sind. Aus diesem Grund existiert ein so genannter Class Builder, der die dialoggestützte Definition von globalen Objekten ermöglicht. Diese Klassen sind dann systemweit verfügbar.

2.7.1 Klassen

Eine Klasse ist die Beschreibung eines Objekts. Sie ist vergleichbar mit einer Typbeschreibung eines Datentyps, weist aber eine größere Komplexität auf, da sowohl die Daten als auch die Schnittstelle deklariert werden. Von einer Klasse können in einem Programm Objekte (Instanzen) erzeugt werden, ebenso wie durch Bezug auf einen Datentyp Datenfelder erzeugt werden können.

Eine Klasse besteht immer aus zwei Teilen, einem Deklarations- und einem Implementierungsteil.

2.7.2 Interfaces

Ein Interface beschreibt die Schnittstelle für eine Klasse. Es besitzt dazu ähnlich wie eine Klasse einen Deklarations-, aber keinen Implementierungsabschnitt. Innerhalb einer Klassendeklaration können beliebig viele Interfaces eingebunden werden. Die Implementierung der Methoden erfolgt dann im Implementierungsteil der Klasse. Auf diese Weise ist es möglich, dass mehrere Klassen ganz oder teilweise identische Schnittstellen aufweisen.

2.7.3 Objekte

Ein Objekt ist die reale Instanz einer Klasse innerhalb einer Anwendung. Das Verhältnis entspricht etwa dem eines Datenfelds zu seinem Datentyp. Normalerweise werden Aktionen immer nur mit den Instanzen einer Klasse, also den Objekten, ausgeführt. Allerdings besteht die Möglichkeit, Klassen so zu gestalten, dass auch ohne Erzeugung einer Instanz auf Elemente der Klasse zugegriffen werden kann.

2.7.4 Vererbung

Innerhalb einer Klassendeklaration können Sie den Bezug auf eine bereits existierende Klasse herstellen. Dadurch werden in Ihrer neuen Klasse automatisch alle Daten und Funktionen der übergeordneten Klasse verfügbar. Sie können nun eigene Daten und Funktionen hinzufügen oder existierende Funktionen ersetzen. Auf diese Weise nutzen Sie existierende Funktionalität und müssen sie nicht neu implementieren. Die von Ihnen neu geschaffene Klasse ist typkompatibel zur alten Klasse.

Dieser Vorgang wird als Vererbung bezeichnet. Die Klasse, von der Sie erben, ist die Superklasse der von Ihnen erzeugten neuen Klasse.

2.7.5 Events

Innerhalb der OO-Erweiterung von ABAP stehen so genannte Events zur Verfügung. Ein Event ist eine Nachricht, die von Objekten ausgesendet bzw. empfangen werden kann. Dabei ist sowohl die gezielte Kommunikation zwischen einander bekannten Sendern und Empfängern als auch eine Rundruf-Funktionalität möglich.

Events besitzen einen eindeutigen Namen und lösen bereits durch die Tatsache ihrer Existenz Reaktionen aus. Sie können optional mit zusätzlichen Daten versehen werden, um die auszulösenden Reaktionen näher zu spezifizieren.

2.7.6 Dynamische Dokumente

Im Zusammenhang mit der Einführung der Objektorientierten Programmierung und der Öffnung des SAP-Systems zum Web erwies sich das Dynpro-Konzept als nicht flexibel genug. Dynamische Dokumente ermöglichen es, eine Bildschirmmaske zur Laufzeit zu erstellen. Dabei werden die Eingabeelemente durch Anweisungen im Programmcode definiert. Damit wird es möglich, eine Anwendung zu erstellen, die ohne die herkömmlichen Dynpros arbeitet. Außerdem kann die Oberfläche deutlich attraktiver und funktionaler gestaltet werden, da eine Vielzahl von GUI-Objekten zur Verfügung steht. Die Liste reicht dabei von einfachen Eingabefeldern bis hin zu sehr komplexen Gebilden wie Editoren, HTML-Browsern und Baumdarstellungen.

2.8 Bedienoberfläche

Die Bedienoberfläche des SAP-Systems erfuhr in den vergangenen Jahren erhebliche Veränderungen. Beim Übergang zur Version 4.0 wurde das Aussehen der vorhandenen Elemente überarbeitet, ohne die Funktionalität der Oberfläche und deren Bedienweise zu verändern. Beginnend mit der Version 4.5 standen in der Programmiersprache ABAP neue GUI-Elemente zur Verfügung, die nach und nach in den verschiedensten Anwendungen zum Einsatz kamen. Dies führte zu einer erheblichen Veränderung der Funktionalität der Oberfläche und der Bedienung einer Anwendung. Als Nebeneffekt dieser Überarbeitung ist der eigentliche SAPGUI nur noch unter Microsoft-Windows-Plattformen lauffähig. Für alle anderen Betriebssysteme existiert ein Java-basierter Client.

Das laufende R/3-System besteht aus Sicht des Anwendungsprogrammierers prinzipiell aus zwei Teilen. Ein Bestandteil ist das System im engeren Sinn, das unter anderem die interne Steuerlogik und den ABAP-Interpreter enthält. Die Funktionalität dieses Teils des Systems ist fest vorgegeben und kann durch den Anwendungsprogrammierer nicht verändert werden. Der zweite Bestandteil ist die momentan ausgeführte, in ABAP geschriebene Anwendung. Eine ABAP-

Anwendung läuft stets unter vollständiger Kontrolle des Systemkerns ab. Fehlfunktionen dieser Anwendung gefährden daher nicht die Funktionsfähigkeit des Gesamtsystems.

Diese Zweiteilung des Systems und die Unterordnung unter das Frontend-Betriebssystem spiegeln sich in der Gestaltung der visuellen Oberfläche des Systems wider. Zunächst stellt das Betriebssystem einige Ressourcen bereit. Diese bestehen im eigentlichen systemeigenen Bildschirmfenster aus einigen Bedienelementen, z. B. dem Systemmenü oder Schaltflächen zum Vergrößern oder Verkleinern des Fensters. Diese Elemente entziehen sich vollständig der Beeinflussung durch den Programmierer, ihre Funktionalität ist durch das System fest vorgegeben. Einige weitere Eingabeelemente werden direkt durch den Kern des R/3-Systems bereitgestellt und ausgewertet. Diese Elemente können durch den Programmierer ebenfalls nicht oder nur sehr eingeschränkt bearbeitet oder ausgewertet werden. Beispiele sind Teile der Statuszeile oder die Funktionalität des Befehlsfelds. Andere Elemente stehen hingegen zur freien Verfügung einer ABAP-Anwendung. Abbildung 2.1 zeigt das Bildschirmfenster einer solchen Anwendung.

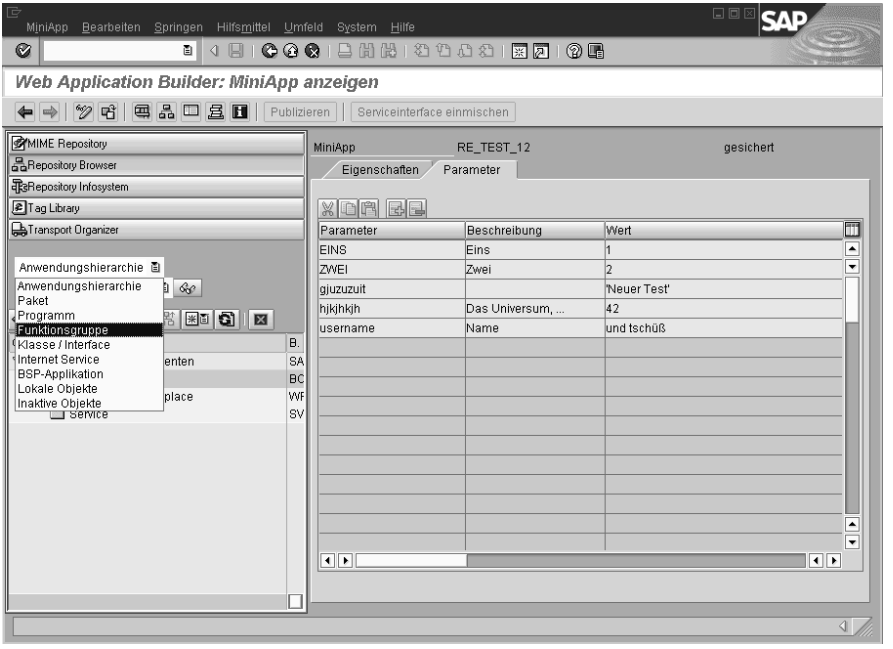

Abbildung 2.1 **© SAP AG**
Oberfläche einer R/3-Anwendung

Die durch das Betriebssystem bereitgestellten Bedienelemente erlauben insbesondere das Verschieben des Bildschirmfensters sowie die Veränderung seiner Größe. Die Symbolleiste direkt unterhalb des Menüs erlaubt die Kommunikation

mit dem R/3-Systemkern. In der Symbolleiste erscheinen einige Symbole mit systemweit einheitlicher Bedeutung sowie das so genannte Befehlsfeld, mitunter auch OK-Code-Feld genannt. Die Symbole können durch die ABAP-Anwendung nur aktiviert oder deaktiviert werden. Das Hinzufügen von neuen oder das Löschen vorhandener Symbole ist nicht möglich. Die mit diesen Symbolen ausgelösten Funktionscodes dienen direkt oder indirekt zur Steuerung der laufenden ABAP-Anwendung. Je nach Anwendung werden die entsprechenden Funktionscodes entweder durch das System verarbeitet (Beispiele: Rollen einer Liste) oder direkt an die ABAP-Anwendung weitergeleitet und dort ausgewertet.

Das Befehlsfeld ist immer eingabebereit. Es kann allerdings durch das kleine Dreieck rechts neben dem Eingabefeld ein- und ausgeblendet werden. Eingaben in das Befehlsfeld werden direkt durch den Systemkern ausgewertet. Mit speziellen Kommandos ist es beispielsweise möglich, laufende ABAP-Anwendungen abzubrechen, Transaktionen aufzurufen, neue Modi (Bildschirmfenster) zu erzeugen oder einige systemnahe Funktionen auszulösen. Ein Symbol neben dem Eingabefeld zeigt eine Listbox mit den letzten Eingaben in das Befehlsfeld an. Tabelle 2.1 zeigt die wichtigsten Eingabemöglichkeiten.

Eingabe	Wirkung
Xxxx	Aufruf der Transaktion xxxx, falls momentan keine Transaktion aktiv ist. Falls doch, wird xxxx als Funktionscode innerhalb der Transaktion gewertet.
/nxxxx	Abbruch der aktuellen und Start der Transaktion xxxx
/oxxxx	Erzeugen eines neuen Modus (Bildschirmfensters) und Start der Transaktion xxxx in diesem Fenster
/n	Abbruch der aktuellen Transaktion
/i	Aktuellen Modus löschen
/o	Modusliste erzeugen
/h	Einschalten des Debugger-Modus
/$sync	Zurückschreiben aller Puffer
/nend	Abmelden

Tabelle 2.1
Wichtige Eingabemöglichkeiten für das Befehlsfeld

Am unteren Rand des Fensters befindet sich die Statusleiste. Hier zeigt das System in einigen Feldern systembezogene Informationen an. Von besonderer Bedeutung ist hierbei das erste, in der Abbildung leere Feld, in dem die Kurztexte

der Nachrichten eingeblendet werden. Ein Doppelklick auf eine Nachricht in diesem Feld blendet ein Popup mit dem Langtext der Nachricht ein.

Alle anderen Elemente außer Symbol- und Statusleiste und den Elementen des Betriebssystems können durch den Programmierer freizügig bearbeitet werden. Diese Elemente stellen somit die eigentliche Oberfläche einer Transaktion dar. Diese besteht aus zwei Teilen: zum einen aus der Arbeitsfläche, die durch das Dynpro oder die Ausgabesteuerung einer Liste verwaltet wird, und zum anderen aus den eigentlichen Bedienelementen, also dem Menü, der Drucktastenzeile und der Belegung der Funktionstasten der Tastatur. Diese Elemente können unabhängig voneinander bearbeitet werden.

Die Bedienelemente einer Anwendung werden als Oberfläche bezeichnet. Innerhalb eines herkömmlichen Dialog-Programms können verschiedene als Status bezeichnete Varianten der Oberflächen existieren. Zu bestimmten Zeitpunkten kann der jeweils aktuelle Status mit einem ABAP-Kommando gesetzt werden. Jeder Status erhält innerhalb eines Programms einen eindeutigen Namen. Die aktuelle Ausprägung der Bedienoberfläche wird daher durch den Programmnamen und den Statusnamen eindeutig identifiziert. Ein Status umfasst die folgenden Elemente:

- Menü
- Drucktastenzuordnung
- Funktionstastenzuordnung
- Titelleiste

Da die Elemente in einem Status dem Anwender die Steuerung des Programmablaufes ermöglichen, muss vor der Bearbeitung eines Status Klarheit darüber bestehen, was im Programm eigentlich geschehen soll, also welche Funktionen über die Bedienelemente ausgelöst werden sollen. Beispiele sind der Aufruf verschiedener Folgedynpros in Abhängigkeit vom gewählten Menüeintrag oder vom Zustand einiger Auswahlknöpfe.

Hilfsmittel zur Kommunikation zwischen Bedienoberfläche und Programm sind die so genannten Funktionscodes. Ein Funktionscode ist eine vierstellige Zeichenkette. Von Ausnahmen wie dem Titel oder dem Befehlsfeld abgesehen wird jedem Bedienelement ein solcher Funktionscode zugeordnet. Bei Betätigung des Bedienelementes wird der zugehörige Funktionscode in eine spezielle Variable gestellt und kann innerhalb des Programms ausgewertet werden.

Damit die Anwendung bei Bedarf komplett über die Tastatur gesteuert werden kann, sollten alle in einem Status verfügbaren Funktionscodes auf jeden Fall über einen Menüeintrag verfügbar sein. Die wichtigsten Funktionscodes können auf die Funktionstasten gelegt werden. Sie stehen damit unmittelbar zur Verfügung. Ein Klick mit der rechten Maustaste blendet die wichtigsten Tastenbelegungen in einem Popup ein. Diese Tasten bzw. die zugehörigen Funktionscodes können in diesem Popup mit einem weiteren Mausklick ausgelöst

werden. Ab Release 3.0C stehen weitere Tastenkombinationen (theoretisch insgesamt 99) zur Verfügung, die mit Funktionscodes belegt werden können. Für einige Funktionstasten existieren Standardvorgaben durch das System. Derartige Tasten sind meist mit den Symbolen der Symbolleiste verbunden. Einige der den Funktionstasten zugeordneten Funktionscodes können auf zusätzliche Drucktasten gelegt werden. Diese erscheinen unterhalb der Symbolleiste und ermöglichen den schnellstmöglichen Zugriff mittels der Maus. Ab Release 3.0 können Drucktasten auch als Icon dargestellt werden.

2.9 Funktions- und Dialogbausteine

Die derzeitigen R/3-Anwendungen weisen eine erhebliche Komplexität auf. Dies erfordert leistungsfähige Methoden zur Modularisierung einer Anwendung. Des Weiteren werden natürlich auch anwendungsübergreifende Hilfsmittel notwendig. Zu nennen wären beispielsweise Funktionen zur Kalenderrechnung oder Dynpros zum Anzeigen oder Erfassen von Standarddaten wie Adressen etc.

Neben der Möglichkeit, eine Anwendung durch Module und Unterprogramme zu strukturieren, stehen zwei Bausteine zum anwendungsübergreifenden Einsatz zur Verfügung. Es handelt sich dabei um Funktions- und Dialogbausteine. Sie können unabhängig von einer konkreten Anwendung entworfen und getestet werden. Sie können allerdings nicht wie ein Programm eigenständig ablaufen, sondern müssen von anderen Anwendungen aus aufgerufen werden. Sie verfügen über eine genau beschriebene Schnittstelle zum Datenaustausch; andere Möglichkeiten zum Datentransfer existieren nicht. Damit werden unerwünschte Nebeneffekte ausgeschlossen.

2.9.1 Dialogbausteine

Dialogbausteine umfassen ähnlich wie Dialogtransaktionen einen Modul-Pool, Dynpros und eine Oberfläche. Die Bausteine besitzen ein Startdynpro und können auf einen Status zugreifen. Dialogbausteine werden daher auch ähnlich wie Dialogtransaktionen programmiert. Im Gegensatz zu Funktionsbausteinen werden sie recht selten eingesetzt.

2.9.2 Funktionsbausteine

Funktionsbausteine stellen dem Programmierer allgemeine oder anwendungsübergreifende Funktionen zur Verfügung. Funktionsbausteine unterliegen hinsichtlich der inneren Funktionalität keinen Einschränkungen. Obwohl für die Ausführung von Dialogen entsprechende Dialogbausteine existieren, können auch in Funktionsbausteinen Dynpros aufgerufen werden, wodurch sie auch die Funktionalität von Dialogbausteinen bieten können.

Funktionsbausteine existieren nicht losgelöst von ihrem Umfeld. Ebenso wie andere Elemente der SAP-Anwendungen sind auch Funktionsbausteine in eine Hierarchie eingebunden. Mehrere logisch zusammengehörige Funktionsbausteine werden in einen so genannten *Function-Pool* aufgenommen. Dieser ähnelt entfernt einem Modul-Pool. In gewisser Weise können die Funktionsbausteine mit den einzelnen Modulen verglichen werden. Der Function-Pool enthält globale Daten für alle Funktionsbausteine des jeweiligen Pools. Diese Daten sind von außen nicht sichtbar; auf sie kann lediglich über die Funktionsbausteine zugegriffen werden. Die aktuellen Werte bleiben nach Verlassen eines Bausteins erhalten!

Da die Funktionsbausteine nur über die definierte Schnittstelle mit dem rufenden Programm kommunizieren können, werden eventuell auftretende Fehlerzustände mittels eines speziellen Ausnahmemechanismus behandelt, der ebenfalls Teil der Schnittstelle ist.

Funktionsbausteine können unabhängig von einem rufenden Programm getestet werden, wobei Testwerte aufbewahrt und später zu Vergleichszwecken benutzt werden können.

2.10 Fehlerbehandlung und Nachrichtenkonzept

Bei der Abarbeitung eines Programms können unvorhergesehene Ereignisse auftreten, die eine Reaktion seitens des Anwenders erfordern. Eine ABAP-Anwendung kann daher individuelle Fehlermeldungen ausgeben, die hier Nachrichten genannt werden. Über spezielle Optionen der Nachrichten auslösenden Anweisung kann auf die Darstellung der Nachricht innerhalb der Anwendung und die notwendige Reaktion des Anwenders in gewissen Grenzen Einfluss genommen werden.

Die Anzeige von Nachrichten hat Einfluss auf den Programmablauf. Es ist daher nicht unbedingt sinnvoll, Nachrichten sofort nach Auftreten eines speziellen Ereignisses auszulösen. Das kann z.B. der Fall sein, wenn innerhalb eines Funktionsbausteines ein Fehler erkannt wird. Üblicherweise lösen die Funktionsbausteine keine Fehlermeldung, sondern eine so genannte Ausnahme (Exception) aus, die den Funktionsbaustein beendet und dem rufenden Programm einen Fehlercode übermittelt, welcher dort ausgewertet werden muss.

2.10.1 Nachrichten

Insbesondere bei der Überprüfung der Eingaben in den PAI-Modulen wird bei erkannten Fehlern eine Fehlersignalisierung an den Anwender erforderlich, um ihn auf eine fehlerhafte Eingabe hinzuweisen und zur Korrektur aufzufordern. Auch in verschiedenen anderen Fällen kann eine Benachrichtigung des Anwenders notwendig werden. Innerhalb von R/3 erfolgt dies über so genannte Nach-

richten. Das Auslösen von Nachrichten kann die Ablauflogik des Dynpros beeinflussen, worauf der Programmierer nur beschränkt Einfluss nehmen kann. Beim Auslösen einer Nachricht sind insgesamt drei Parameter erforderlich.

R/3-Nachrichten sind vordefinierte Textelemente. Sie bestehen aus einem einzeiligen Kurztext und einem optionalen, mehrzeiligen Langtext. Als Identifikator dient eine dreistellige Zahl. Die Nachrichtentexte können übersetzt werden. Das System verwendet stets die Nachrichten der Anmeldesprache.

Da die Anzahl von 999 Nachrichten für das R/3-System keinesfalls ausreicht, werden die einzelnen Nachrichten so genannten Nachrichtenklassen zugeordnet. Eine Nachrichtenklasse wird durch einen zweistelligen Namen identifiziert. Sie umfasst üblicherweise die für eine oder mehrere verwandte Anwendungen erforderlichen Nachrichten. Funktionale Unterschiede zwischen den Nachrichtenklassen bestehen nicht.

Zusätzlich zur Nachrichtennummer und -klasse ist bei der Auslösung einer Nachricht in einem Programm noch eine dritte Angabe erforderlich, die der Nachrichtenart. Dieser Wert hat keinen Einfluss auf den Text der Nachricht, sondern bestimmt lediglich die Art und Weise ihrer Darstellung und der Beeinflussung der Ablaufsteuerung des Dynpros.

Die Nachrichten können auf vier verschiedene Arten angezeigt werden. Die Unterschiede bestehen sowohl in der Form der Anzeige als auch in der weiteren Reaktion der Anwendung. Eine Variante besteht beispielsweise in der Darstellung eines Kurztextes in der Statuszeile. Eine andere Möglichkeit besteht darin, die Nachricht in einem eigenen Fenster anzuzeigen, das nur durch das Betätigen einer Schaltfläche geschlossen werden kann. Mittels einer weiteren Schaltfläche in diesem Fenster kann das ausführlichere Hilfsfenster mit dem Langtext der Nachricht angezeigt werden. Nach dem Schließen des Fensters wird das Programm fortgesetzt.

2.10.2 Ausnahmen

Der Begriff Ausnahme kann in zwei unterschiedlichen Zusammenhängen benutzt werden. Zunächst treten Ausnahmen auf, wenn bei der Ausführung eines ABAP-Kommandos ein Fehler auftritt, z. B. ein Zahlenüberlauf. Einige dieser Fehler können durch spezielle ABAP-Anweisungen abgefangen werden, andere nicht. Sofern diese Ausnahmen nicht abgefangen werden, entsteht ein Laufzeitfehler. Diese Gruppe von Ausnahmen sind im Sprachumfang von ABAP fest definiert und können durch den Programmierer nicht verändert werden.

Eine zweite Form von Ausnahmen wird von Funktionsbausteinen ausgelöst. Für diese Ausnahmen wird oft die englische Bezeichnung *Exception* benutzt. Über Exceptions signalisieren Funktionsbausteine dem rufenden Programm Fehlerzustände. Damit sind nicht unbedingt echte Laufzeitfehler gemeint. Exceptions können auch in anderen Fällen ausgelöst werden, z. B. wenn ein Funk-

tionsbaustein einen Datensatz nicht finden kann. Exceptions können vom Programmier eines Funktionsbausteins selbst definiert und ausgelöst werden. Es existiert nur eine Ausnahmeart ohne Varianten, wie sie etwa bei Nachrichten möglich sind. Allerdings kann es beliebig viele Ausnahmen geben. Die genaue Reaktion einer Anwendung auf eine Ausnahme hängt sowohl vom Kommando ab, mit dem die Ausnahme erzeugt wird, als auch vom Ort ihrer Bearbeitung.

Beim Erzeugen eines Funktionsbausteins wird dessen Schnittstelle genau definiert. Zu dieser Schnittstellendefinition gehören auch die Bezeichner für die Ausnahmen, die innerhalb des Funktionsbausteins erzeugt werden sollen. Beim Erzeugen einer Ausnahme wird dem dazu benutzten Kommando der Bezeichner der Ausnahme als Parameter übergeben. Wird hingegen die Ausnahme durch den Funktionsbaustein selbst behandelt, so erscheint deren Name im Fehlertext.

Sollen Ausnahmen außerhalb eines Funktionsbausteins behandelt werden, muss das rufende Programm dies dem Funktionsbaustein beim Aufruf explizit mitteilen. Dazu werden den Bezeichnern der möglichen Ausnahmen nummerische Werte zugeordnet. Beim Erzeugen einer Ausnahme stellt die Steuerlogik des Systems den zur Ausnahme gehörenden nummerischen Wert in eine Systemvariable. Das rufende Programm kann nun anhand des Inhaltes dieser Variablen unterscheiden, ob und wenn ja, welche Ausnahme ausgelöst wurde.

2.11 Hilfestellungen und Eingabehilfen

Bei der komplexen Struktur des SAP-Systems und der Vielzahl der Eingabemöglichkeiten sind Hilfsmittel zur Eingabe in ein Feld unerlässlich. Innerhalb ausgewählter Dynprofelder kann mittels der Funktionstaste F4, dem Lupensymbol aus der Symbolleiste oder dem Pfeilsymbol neben einem Eingabefeld eine Eingabehilfe angefordert werden. Zwei Varianten werden bevorzugt eingesetzt: die Auswahl aus einem vorgegebenen Vorrat an Werten (z.B. Maßeinheiten) und die Suche nach Werten, die nicht exakt oder zu denen nur Zusatzinformationen bekannt sind, z.B. die Suche nach der Personalnummer anhand eines Namens. Beide Varianten werden unterstützt, wobei drei prinzipielle Mechanismen existieren:

- Der erste ist die Verwendung einer Suchhilfe. Sie wird üblicherweise eingesetzt, um den Schlüssel eines Datensatzes zu ermitteln, zu dem nur untergeordnete oder Nicht-Schlüsselwerte bekannt sind. Suchhilfen erfordern keine ABAP-Programmierung, da sie mit speziellen Werkzeugen des Data Dictionary im Dialog erzeugt werden können.
- Die zweite Möglichkeit besteht in der Nutzung der `PROCESS ON VALUE-REQUEST` (POV). Diese ähnelt der PAI-Verarbeitung und erfordert die Erzeugung eines ABAP-Programms. Sie ermöglicht die Implementierung einer individuellen Eingabehilfe.

- Als letzte Variante bleibt der Zugriff auf eine Wertetabelle oder die Festwerte einer Domäne. Für Felder einer Tabelle kann ein Fremdschlüssel definiert werden. Er verweist auf ein Feld einer anderen Tabelle. Das Feld, für das der Fremdschlüssel definiert wurde, darf nur Werte enthalten, die in dem per Fremdschlüssel zugewiesenen anderen Tabellenfeld enthalten sind. Für solche Fremdschlüssel kann das System automatisch eine Eingabehilfe bereitstellen. Dieselbe Eingabehilfe wird auch aktiv, wenn an Stelle einer Wertetabelle in der Domäne Festwerte definiert wurden. Diese können auch ohne Fremdschlüsselbeziehungen angezeigt werden.

Neben der Werthilfe ist mittels der Funktionstaste F1 ein allgemeiner Hilfetext abrufbar. Dieser beruht auf einem Langtext, der neben der Kurzbeschreibung für viele Elemente der Entwicklungsumgebung erfasst werden kann. Alternativ dazu kann eine `PROCESS ON HELP-REQUEST`-Verarbeitung (POH) definiert werden. Sie zeigt einen individuellen Hilfetext für ein Element an.

2.11.1 Matchcodes und Suchhilfen

Im relationalen Datenbankmodell kommt dem Begriff „Schlüssel" eine zentrale Bedeutung zu. Einzelne Tabellen werden über die Inhalte von Schlüsselfeldern miteinander verknüpft. Ein Schlüssel muss im Normalfall einen Datensatz eindeutig identifizieren. In vielen Eingabemasken ist vom Anwender ein Schlüsselwert einzugeben, um einen zu bearbeitenden Datensatz zunächst zu selektieren. Mitunter ist dieser Wert (oft eine nummerische Zeichenkette) nicht bekannt, wohl aber andere Merkmale des zu bearbeitenden Datensatzes, die allerdings mehrdeutig sein können. Ein Personalsachbearbeiter wird nicht alle Personalnummern auswendig kennen, sondern den Datensatz eines Beschäftigten über dessen Namen suchen wollen. Für derartige Suchvorgänge stehen Matchcodes (bis Version 3.1x) bzw. Suchhilfen (ab Version 4.0) zur Verfügung. Suchhilfen stellen eine Verallgemeinerung der Matchcodes dar. Sie bieten einen etwas erweiterten Funktionsumfang. Im Gegensatz zu Matchcodes, wo verschiedenen Typen existierten, gibt es für Suchhilfen nur noch eine Variante.

Aus Anwendersicht ist eine Suchhilfe ein Abfrageprogramm, das vordefinierte Suchvorgänge innerhalb eines vorgegebenen Datenbestandes ausführt und einen oder mehrere Werte aus dem gefundenen Datensatz zurückliefert. Innerhalb der Suchhilfe können mehrere Unterabfragen definiert werden, die den gewünschten Suchbegriff auf jeweils unterschiedliche Weise ermitteln. Bezogen auf das obige Beispiel, „Suche nach einer Personalnummer", könnte diese beispielsweise über den Namen eines Beschäftigten, seine Tätigkeit, den Arbeitsort oder die Gehaltsklasse ermittelt werden. Aus Sicht des Programmierers sind Suchhilfen spezielle Sichten auf den Datenbestand. Sie werden unabhängig von der späteren Anwendung erzeugt und können in mehreren Anwendungen benutzt werden.

Bei Aktivierung einer Suchhilfe in einer Anwendung (meist mit der Funktionstaste F4 oder einem Symbol neben dem Eingabefeld) wird zunächst eine Liste mit den zur Verfügung stehenden Suchmöglichkeiten angeboten, aus denen eine gewählt werden muss. Es erscheint eine Eingabemaske, in der die Suchbegriffe eingetragen werden. Dabei ist die Verwendung von Musterzeichen möglich. Anschließend wird die Abfrage gestartet. Falls Datensätze gefunden werden, die den Suchkriterien entsprechen, werden diese wiederum in einer Liste angeboten, aus der nun der gewünschte Datensatz und damit der zu übergebende Wert ausgewählt wird.

Von Dynpros aus gestaltet sich der Zugriff auf Suchhilfen sehr einfach, da er als Attribut zu einem Eingabefeld eingetragen werden kann. In einem fertigen Dynpro werden Eingabefelder, denen eine Suchhilfe zugeordnet wurde, mit einem kleinen Dreieck in der rechten oberen Ecke des Eingabefelds gekennzeichnet, solange dieses Feld nicht den Eingabefokus besitzt.

2.11.2 Der Zeitpunkt POV

Alternativ zu den Matchcodes kann für Eingabefelder innerhalb der Ablauflogik eine Process-on-Value-Request-Verarbeitung programmiert werden. Diese Verarbeitungsphase ist ebenso wie die PBO- und die PAI-Verarbeitung (optionaler) Teil der Ablauflogik. Innerhalb des POV-Abschnittes werden einzelnen Feldern des Dynpros mit einer speziellen Anweisung Module zugeordnet. Im jeweiligen Modul können beliebige Anweisungen ausgeführt werden, die letztlich einen Wert in das korrespondierende Dynprofeld stellen. Oft wird dazu ein interaktiver Report aufgerufen.

2.11.3 Der Zeitpunkt POH

Ähnlich wie bei der selbst programmierten Eingabehilfe kann einem Feld im Verarbeitungsabschnitt Process-on-Help-Request (POH) ein Modul bzw. ein Feldzusatz zugeordnet werden, der eine Hilfsinformation zu diesem Feld darstellt. Diese Verarbeitungsart wird relativ selten benutzt, da kontextsensitive Hilfsinformationen auch auf andere Weise bereitgestellt werden können.

2.12 Systemtabellen und Systemfelder

ABAP-Programme werden durch die interne Ablauflogik ausgeführt. Diese überwacht das Programm und steuert einige Aktionen selbsttätig. Die dazu notwendigen Informationen bewahrt die Ablauflogik in einigen Systemvariablen und Tabellen auf, die dem Programmierer zugänglich sind. Sie können z. B. zur Steuerung des Programmablaufs benutzt werden. Eine dieser Variablen (`SY-SUBRC`) liefert beispielsweise eine Information über den Erfolg des letzten ABAP-Kommandos oder Funktionsaufrufes. Dies ist z. B. wichtig, um den Er-

folg oder Misserfolg von Datenbankselektionen auszuwerten. Andere Felder enthalten die Nummer des aktuellen Datensatzes einer internen Tabelle oder die Zahl der bei einem Suchvorgang selektierten Datensätze. Systemvariablen sind vordefiniert. Sie sind in allen Anwendungen automatisch verfügbar, ohne dass sie definiert werden müssten.

Bei der Arbeit mit Dynpros erzeugt die interne Steuerlogik interne Tabellen, die Angaben zum jeweiligen Dynpro oder zu einigen seiner Elemente enthalten. Beispielsweise existiert eine vordefinierte interne Tabelle `SCREEN`, die einige Attribute zu jedem Dynprofeld beinhaltet. Wird diese Tabelle zum Zeitpunkt PBO, also vor Anzeige des Dynpros bearbeitet, so wirken sich die Modifikationen auf die Eingabefelder des Dynpros aus. Auf diese Weise kann der Programmierer das Dynpro dynamisch beeinflussen, also Felder ausblenden oder sie in reine Anzeigefelder verwandeln. Von dieser Möglichkeit wird sehr häufig Gebrauch gemacht.

2.13 Korrektur- und Transportwesen

Das SAP-System ist sehr komplex. Installationen sowohl bei der SAP selbst als auch beim Kunden bestehen fast immer aus mehreren Systemen. Es ist daher notwendig, für die Weitergabe von Anwendungen oder Korrekturen (Patches) sowie die Verwaltung der diversen Entwicklungsobjekte geeignete Werkzeuge zu benutzen. Diese können unter dem Oberbegriff Korrektur- und Transportwesen zusammengefasst werden. Grundlegende Kenntnisse zu diesem Themengebiet erleichtern die Planung und Realisierung einer portablen Anwendung sowie die Zusammenarbeit in einem Team.

2.13.1 Objektkatalog

Alle Entwicklungsobjekte im SAP-System erhalten einen eindeutigen Schlüssel. Zur Verwaltung aller Objekte dient ein Eintrag im so genannten Objektkatalog, der diesen Schlüssel enthält. Es ist sogar möglich, für mehrere SAP-Systeme einen gemeinsamen Objektkatalog einzurichten, der somit die Eindeutigkeit von Entwicklungsobjekten über Systemgrenzen hinweg ermöglicht.

Der Objektkatalog ist zentrales Element für die Verwaltung der Entwicklungsobjekte. Die Informationen im Objektkatalog sind Voraussetzung für die diversen Werkzeuge des Korrektur- und Transportwesens.

2.13.2 Namensraum

Eine wesentliche Voraussetzung für die problemlose Weitergabe von Anwendungen ist eine eindeutige Identifizierung ihrer Elemente. Der Objektkatalog erlaubt es, diese Eindeutigkeit innerhalb eines Systems oder, unter bestimmten

Voraussetzungen, innerhalb mehrerer miteinander verbundener Systeme zu gewährleisten. Um auch Kunden und Partnern von SAP die Möglichkeit zur Anwendungsentwicklung zu geben, können bei der SAP so genannte Namensräume reserviert werden. Ein Namensraum besteht aus einem Bezeichner, der dem Namen des Entwicklungsobjekts vorangestellt wird. Auch auf diese Weise kann Eindeutigkeit der Namen erreicht werden.

2.13.3 Transport

Um eine Anwendung von einem System in das andere übertragen zu können, ist ein so genannter Transport erforderlich. Alle Objekte einer Anwendung werden durch geeignete Werkzeuge identifiziert und aus dem System exportiert. Dabei entstehen ein oder mehrere Transportdateien, deren Format unabhängig vom konkreten Betriebssystem ist. Je nach Aufgabe eines Transportes können verschiedene logische Formen unterschieden werden wie z.B. eine Installation, eine Korrektur (Beheben eines Fehlers), ein Hot Fix (Zusammenfassung mehrerer Korrekturen) usw. Die technische Grundlage ist aber immer dieselbe.

2.13.4 Entwicklungsklasse

Der Begriff der Entwicklungsklasse ist inzwischen überholt. Er wird in neueren Versionen durch den Begriff *Paket* ersetzt. In einer Entwicklungsklasse wurden alle Elemente einer komplexen Anwendung zusammengefasst. Die Entwicklungsklasse bestimmte dabei die so genannten Transporteigenschaften. Diese Eigenschaften legten fest, ob, wie und wohin ein Objekt vom Entwicklungssystem aus transportiert werden kann. Eine Entwicklungsklasse kann im Sprachgebrauch anderer Programmiersprachen auch als Projekt angesehen werden.

2.13.5 Paket

Aktuelle R/3-Versionen benutzen den Begriff des Pakets, um Gruppen von logisch zusammengehörenden Entwicklungsobjekten zu kennzeichnen. Das Paket löst somit die in älteren Versionen vorhandene Entwicklungsklasse ab. Pakete zeichnen sich gegenüber den Entwicklungsklassen dadurch aus, dass sie verschachtelt werden können. Außerdem kann festgelegt werden, welche Elemente eines Paketes von außen, also von Anwendungen, die nicht zum Paket gehören, benutzt werden dürfen.

3 Der Weg zum Programm

Eine sehr wichtige, aber nicht die einzige Voraussetzung zur erfolgreichen Programmierung eigener Anwendungen innerhalb des R/3-Systems ist die Kenntnis der Programmiersprache ABAP. Diese Sprache ist trotz aller Modifikationen der letzten Jahre keine universell einsetzbare Sprache wie z.B. C oder Pascal, auch wenn sie sich in der Praxis als sehr flexibel erwiesen hat. Vielmehr wurde sie speziell für das SAP-System entworfen. Die Sprache wird ständig weiterentwickelt und aktuellen Erfordernissen angepasst. Von einem Releasestand zum anderen ergeben sich zum Teil beträchtliche Ergänzungen. Einige Sprachelemente sind völlig auf ausgewählte Elemente der Entwicklungsumgebung bzw. auf die SAP-Philosophie zugeschnitten und können nur im Zusammenhang mit diesen Elementen beschrieben werden. Es ist deshalb wenig sinnvoll, die einzelnen Kommandos der Programmiersprache losgelöst vom übrigen SAP-Umfeld zu betrachten. Viel wichtiger als die detaillierte Beschreibung aller Kommandos mit zum Teil Dutzenden Parametern und Aufrufoptionen ist zunächst die Vermittlung von Grundwissen zu den verschiedenen Bestandteilen einer R/3-Anwendung. Dieses Kapitel soll daher keine vollständige Beschreibung der ABAP-Kommandos sein. Vielmehr erläutert es eine Auswahl der wichtigsten Anweisungen und demonstriert an sehr einfachen Beispielen deren praktischen Einsatz, das Zusammenwirken mit anderen Elementen der Entwicklungsumgebung und damit auch die Programmstruktur einer Anwendung. Dieser Abschnitt ist daher nicht als vollständige Referenz der Programmiersprache, sondern als schrittweise praktische Einführung in die R/3-Programmierung zu verstehen. Dieses Kapitel eignet sich daher weniger zum Nachschlagen bestimmter Fakten, sondern sollte sequenziell durchgearbeitet werden.

Bereits in den zurückliegenden Versionen, vor allem seit der Version 4.6, versucht die SAP die Sprache ABAP und das Konzept des Gesamtsystems intensiv auf die Erfordernisse des Internets auszurichten. Dafür wurden einige neue Programmierkonzepte eingeführt, die mit bisherigen Traditionen brechen. Trotz aller neuen Elemente ist die Kenntnis der seit Jahren existierenden Program-

miermodelle aber keineswegs überflüssig. Zum einen können nicht alle Anwendungen von einer Version zur anderen komplett umgestellt werden. Pflege und Anpassung existierender Anwendungen stellt aber einen großen Anteil der praktischen ABAP-Programmierung dar. Zum anderen erleichtert die Kenntnis der Wurzeln des R/3-Systems das Verständnis für die neuen Elemente. Darüber hinaus bringt ein Internet-basierter Zugriff auf das R/3-System nicht nur Vorteile mit sich. Die auf dem HTML-Standard beruhenden Objekte der Benutzerschnittstelle verfügen nicht über die Komplexität und Interaktivität der Elemente, die in einer reinen Desktop-Anwendung zur Verfügung stehen.

Am genannten Anliegen orientiert sich auch der Aufbau dieses Kapitels. Voraussetzung zur Erstellung eigener Programme ist die Kenntnis der Entwicklungsumgebung und der diversen Werkzeuge. Im ersten Unterabschnitt lernen Sie die grundlegenden Eigenschaften der Entwicklungsumgebung kennen, in dem Sie ein denkbar einfaches Programm erzeugen. Der darauf folgende Abschnitt beschreibt grundlegende Kommandos, die in allen Programmtypen eingesetzt werden können. Nahezu alle dieser Anweisungen sind von globaler Bedeutung. Die Kenntnis dieser Anweisungen ist auf jeden Fall Voraussetzung zum Verständnis aller ABAP-Anwendungen. Dabei werden zwar Programme vom Typ *Online-Report* benutzt, diese Programme sind aber keine vollwertigen Reports im engeren Sinn des Wortes. Vielmehr dienen sie nur als Hilfsmittel zur praktischen Demonstration der ersten Anweisungen.

Eine R/3-Anwendung besteht nicht nur aus Quelltext, sondern umfasst auch weitere Elemente. Zu diesen Elementen gehört die so genannte *Oberfläche*. Hinter diesem Begriff verbergen sich vor allem die Menüs, über die Sie eine Anwendung steuern. In nahezu allen Dialoganwendungen und vielen Reports müssen Sie eigene Menüs verwenden. Der dritte Abschnitt beschreibt die Eigenschaften dieser Menüs sowie das entsprechende Pflegewerkzeug.

Echte Reports mit ihren speziellen Eigenschaften und Anweisungen stehen im Mittelpunkt des vierten Unterabschnitts. Einige ABAP-Anweisungen können nur in Reports benutzt werden. Außerdem besitzen Reports eine spezielle Programmstruktur. Abgesehen von der Beschreibung dieser speziellen Anweisungen festigt das Studium dieses Abschnittes Kenntnisse zu den bereits vorgestellten Anweisungen allgemeiner Natur.

Das starre Ablaufschema der klassischen Reports kann durch interaktive Sprachelemente aufgebrochen werden. Der fünfte Unterabschnitt beschäftigt sich insbesondere mit diesen Erweiterungen, wie z.B. Verzweigungen in Teillisten.

Logische Datenbanken erleichtern in Reports die Datenselektion und die Wiederverwendbarkeit von Quelltext. Reports können auf eine logische Datenbank zugreifen, müssen dies aber nicht. Der sechste Abschnitt beschreibt die Bearbeitung und Verwendung von logischen Datenbanken.

Im siebenten Unterabschnitt wird das Dynpro-Konzept genauer erläutert. Dazu ist es zunächst erforderlich, die Struktur einer dialogorientierten Anwendung

zu beschreiben. Erst dann werden die Anweisungen, mit denen diese Programmstruktur realisiert wird, vorgestellt.

Der daran anschließende Abschnitt beschreibt die Verknüpfung zwischen List- und Dialogverarbeitung.

Ein wichtiges Hilfsmittel zur Bereitstellung anwendungsübergreifender Funktionalität und zur Modularisierung von Anwendungen sind Funktions- und Dialogbausteine. Auch für diese Bausteine ist eine eigenständige Programmierweise erforderlich. Einige vorgefertigte, allgemein verwendbare Funktionsbausteine werden durch das System bereitgestellt. Der vorletzte Abschnitt dieses Kapitels führt in die Bearbeitung dieser Bausteine ein, der letzte bietet einige Beispiele zur Nutzung der vom System bereitgestellten Funktionsbausteine an.

Der objektorientierten Erweiterung von ABAP und den damit verbundenen Prinzipien ist ein eigenes Kapitel (Kapitel 4, ABAP Objects) gewidmet. Dieses Kapitel beschäftigt sich ausdrücklich nur mit Kommandos, die unabhängig von der objektorientierten Erweiterung benutzt werden können.

Noch eine Vorbemerkung zur eigentlichen Oberfläche. Ab Version 4.6 steht eine völlig neu gestaltete, auf dem Einsatz von Dynpro-Controls beruhende Oberfläche zur Verfügung. Diese Oberfläche ähnelt den Werkzeugen, die z.B. aus der Windows-Welt bekannt sind. Abbildung 3.1 zeigt Ihnen beispielhaft eine Anwendung in der neuen Darstellungsform.

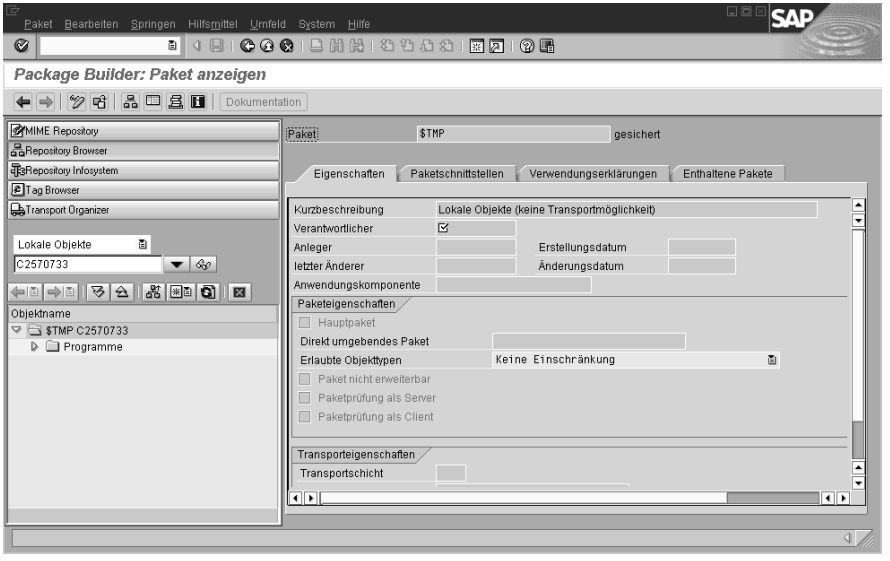

Abbildung 3.1 **© SAP AG**
Beispiel für neue Oberflächen-Elemente

Die neue Oberfläche ermöglicht eine einfachere und schnellere Navigation zwischen den einzelnen Werkzeugen, stellt aber, abgesehen von einem neuen Editor, noch keine prinzipiell neuen Bearbeitungsmöglichkeiten zur Verfügung. Zur Bearbeitung der diversen Elemente werden auch von der neuen Oberfläche die bereits aus älteren Versionen vorhandenen Werkzeuge benutzt.

Durch die gleichzeitige Darstellung von Navigation und Pflegewerkzeug in einem Fenster vergrößert sich der Platzbedarf im Vergleich zu älteren R/3-Versionen (z.B. 3.x) erheblich. Sie können daher mit der Funktionstaste ⇧-F12 den Navigationsbereich ein- und ausblenden.

3.1 Die Entwicklungswerkzeuge

Eine R/3-Anwendung kann sehr komplex werden und besteht aus verschiedenen Elementen. Zur Bearbeitung jedes dieser Elemente stehen eigene Werkzeuge bereit. Um dem Programmierer die Arbeit zu erleichtern, bietet der *Repository Browser* die Gesamtsicht auf alle Elemente einer Anwendung und erlaubt von einer einzigen Oberfläche aus den Zugriff auf alle notwendigen Pflegewerkzeuge. Unter Release 3.x wurde der Repository Browser als Object Browser bezeichnet. Beachten Sie bitte die Abhängigkeit der Begriffe vom Releasestand.

Das erste Programm wird ein einfacher Report sein. Der Begriff *Report* wurde bereits im Kapitel 2 erwähnt. Reports sind Programme, die Auswertungslisten erzeugen, diese auf dem Bildschirm darstellen und bei Bedarf drucken. In seiner Grundform arbeitet der Report nicht dialogorientiert. Allgemeiner ausgedrückt: Reports können nach dem Start ohne weitere Aktivität des Anwenders eine Ausgabe auf dem Bildschirm erzeugen.

Da Reports nicht interaktiv sind, benötigen sie weder Dynpros noch Oberflächen oder Ablauflogik. Ein Report besteht im einfachsten Fall nur aus einem einzigen Quelltext mit einer sehr einfachen Struktur. Hinzu kommt, dass Reports als einzige Programmart sofort aus der Entwicklungsumgebung heraus abgearbeitet werden können. Es bietet sich daher an, derartige Reports für den ersten Einstieg in die Programmierung und die Demonstration der wichtigsten ABAP-Anweisungen zu benutzten.

3.1.1 Das erste Programm: Hello World!

Fast alle Programmierhandbücher beginnen mit demselben Programm. Seine Aufgabe besteht darin, eine einfache Textzeile auf den Bildschirm zu schreiben. Meist lautet diese „Hello World!“. Der praktische Nutzen eines derartigen Programms und der Lerneffekt bezüglich der Sprache halten sich natürlich in Grenzen. Bedeutsam an dieser Art des Einstiegs ist vielmehr der erste Kontakt mit der Entwicklungsumgebung oder den Programmierwerkzeugen. Da diese unter R/3 recht komplex sind, spricht nichts dagegen, auf eben diese Weise zu beginnen.

Erzeugen eines Programms

Alle für die Anwendungsentwicklung erforderlichen Werkzeuge sind in einer so genannten Workbench zusammengefasst. Diese stellt sich dem Anwender als Bereichsmenü dar, das in älteren Versionen der R/3-Software vom R/3-Hauptmenü aus mittels der Menüfunktion WERKZEUGE | ABAP-WORKBENCH oder von beliebiger anderer Stelle mit dem Transaktionscode `S001` erreichbar ist. Abbildung 3.2 zeigt Ihnen dieses Bereichsmenü in einem aktuellen System. Zwecks besserer Navigationsmöglichkeit im Gesamtsystem wird es nicht mehr als herkömmliches Pulldown-Menü im Kopf des Fensters angezeigt sondern als Baum im Navigationsbereich der Arbeitsfläche.

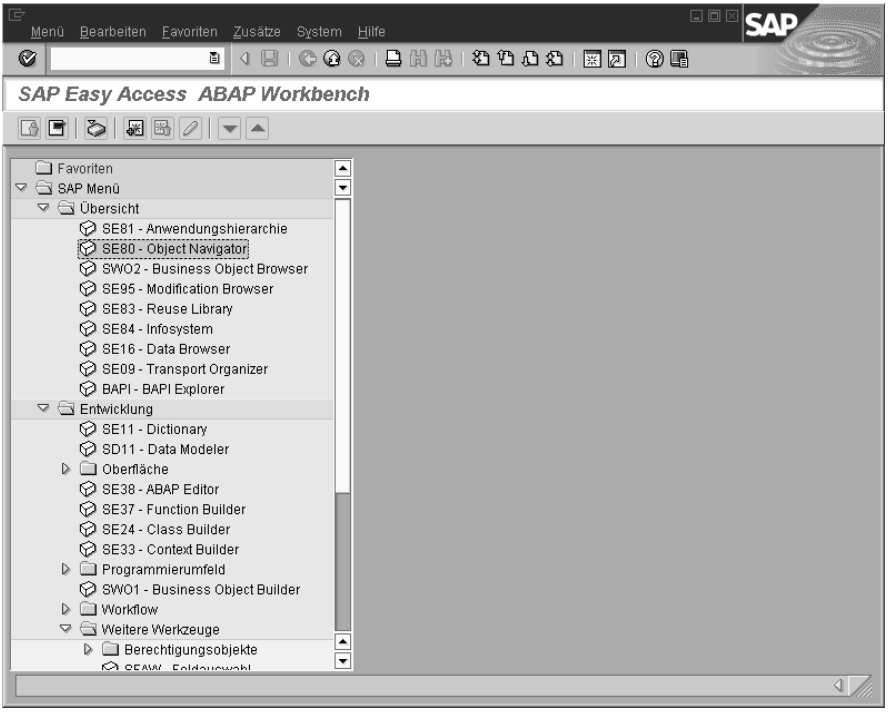

Abbildung 3.2 **© SAP AG**
Die Workbench in der Version 6.x

In der Workbench finden Sie alle Werkzeuge, die in irgendeiner Form mit der Programmierung von Anwendungen zu tun haben. Sie stellt somit eine funktionsorientierte Zugriffsmöglichkeit dar. Für die Entwicklungstätigkeit besser geeignet ist allerdings der *Object Navigator* (vergleichbar mit dem früheren Repository Browser). Er bietet eine Navigationsmöglichkeit über komplette Entwicklungsprojekte mit allen Bestandteilen. Die für jedes Entwicklungsobjekt erforderlichen Werkzeuge werden vom Object Navigator kontextsensitiv

bereitgestellt. Der Object Navigator ist über den entsprechenden Zweig der Workbench oder den Transaktionscode SE80 erreichbar. Die Oberfläche des Object Navigator (siehe Abbildung 3.3) bietet Ihnen wiederum einen zweigeteilten Arbeitsbereich. Die linke Seite dient zur Navigation innerhalb der unterschiedlichen Bereiche der Anwendung sowie der Auswahl des konkreten Entwicklungsobjekts. Im rechten Teil der Arbeitsfläche wird das Werkzeug zur Bearbeitung eines ausgewählten Entwicklungsobjekts eingeblendet.

Auf Grund der Komplexität des R/3-Systems sowie der Zahl der unterschiedlichen Elemente, Werkzeuge und entwicklungsbegleitenden Aufgaben muss im Object Navigator zunächst eine Auswahl aus einigen Rubriken getroffen werden. Im oberen Teil des Navigationsbereichs finden Sie fünf Drucktasten, von denen bis auf weiteres nur die Drucktaste REPOSITORY BROWSER betätigt werden sollte.

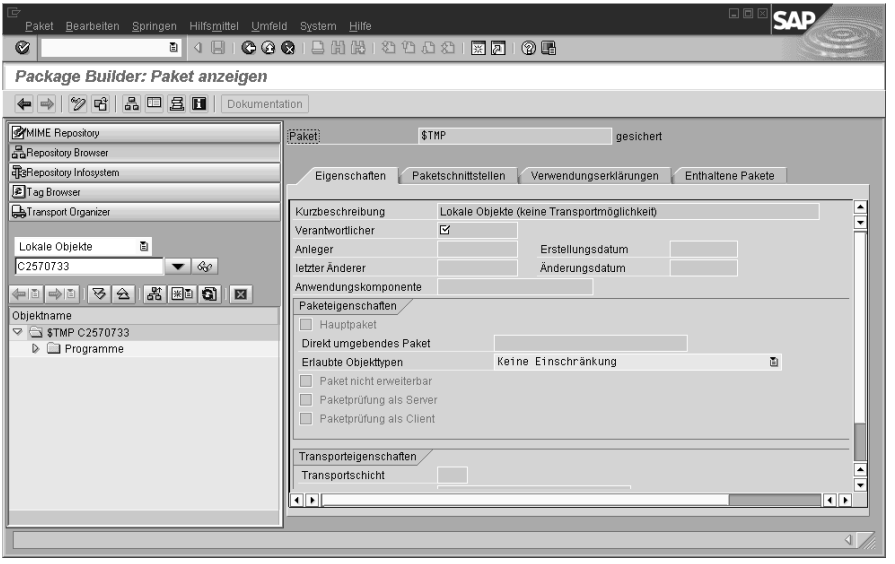

Abbildung 3.3 **© SAP AG**
Grundbild des Object Navigator

Zum Erzeugen eines neuen Programms ist noch eine weitere Vorgabe notwendig, die für alle Beispiele in diesem Kapitel gilt, sofern nichts anderes erwähnt wird. In dem direkt unter den Drucktasten liegenden Drop-Down-Auswahlfeld sollten Sie die Auswahl markieren, sofern sie nicht bereits aktiv ist (vergleiche Abbildung 3.4). Die Erläuterung der einzelnen Begriffe erfolgt später.

Daraufhin sollte im zweiten Eingabefeld Ihr Benutzername erscheinen. Falls nicht, sollten Sie ihn eintragen und mit der ↵-Taste bestätigen. Die Betätigung der ↵-Taste bzw. die des Refresh-Symbols führt zum Neuaufbau der Objektliste im unteren Teil des Navigationsbereiches. Die Oberfläche des Object Navigator sollte nun der Abbildung 3.3 entsprechen.

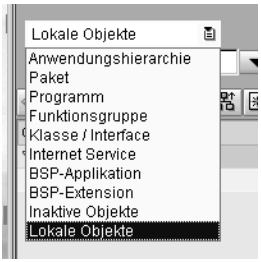

Abbildung 3.4
Auswahlmöglichkeiten für Entwicklungsbereiche im Object Navigator, © SAP AG

Um ein neues Programm anzulegen, markieren Sie durch einen einfachen Mausklick den obersten Eintrag der Objektliste. Dieser besteht aus dem Kürzel „$TMP" als Kennzeichen für lokale Objekte und Ihrem Benutzernamen. Dann rufen Sie durch einen Klick mit der rechten Maustaste das hierarchische Kontextmenü auf und selektieren den Menüeintrag ANLEGEN | PROGRAMM. Abbildung 3.5 zeigt den Object Navigator mit dem aufgeklappten Kontextmenü.

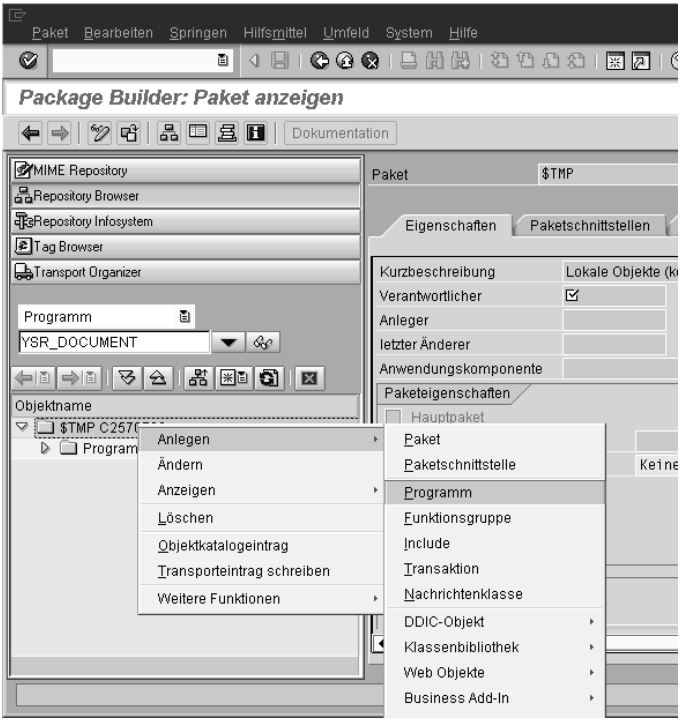

Abbildung 3.5 **© SAP AG**
Kontextmenü zum Anlegen eines neuen Programms

Das System blendet daraufhin ein kleines Popup ein (Abbildung 3.6), in dem Sie den Namen des Programms eintragen müssen.

Abbildung 3.6 © SAP AG
Festlegen des Namens des neuen Programms

An dieser Stelle ist vor allem auf einen korrekten Programmnamen zu achten. Das SAP-System generiert an vielen Stellen selbsttätig Programme, wobei deren Namen nach feststehenden Regeln gebildet werden. Weiterhin unterscheidet die Entwicklungsumgebung zwischen Entwicklungssystemen bei SAP und Kundensystemen. Für beide Arten von Systemen existieren unterschiedliche Vorgaben bezüglich der verwendbaren Namen. Für alle Elemente (nicht nur Reports) prüft die Entwicklungsumgebung einige Namenskonventionen. Auf so genannten Kundensystemen dürfen die Namen neuer Programme nur mit den Zeichen *Y* oder *Z* beginnen. Auf diese Weise sollen eigene Entwicklungen des Kunden eindeutig von denen des Hauses SAP unterschieden werden. Dementsprechend können in den Entwicklungssystemen direkt bei SAP keine Objekte angelegt werden (zumindest nicht ohne eindrückliche Warnung), die mit diesen beiden Buchstaben beginnen. Eine weitere Differenzierung ist durch einen optionalen Präfix vor dem eigentlichen Namen möglich. Dieser Präfix stellt die Bezeichnung eines Namensraums dar. Darauf soll an dieser Stelle allerdings noch nicht eingegangen werden.

Um Konflikte mit den automatisch generierten Namen zu vermeiden und mehreren Anwendern in Ihrem System die Abarbeitung der Demoprogramme zu ermöglichen, sollten Sie einige Konventionen beachten. Für die Beispiele in diesem Buch werden Namen der Form `Yiikknnm` benutzt. Für `ii` setzen Sie bitte Ihre eindeutigen Initialen ein. Damit ist sichergestellt, dass mehrere Benutzer diese Beispiele an einem einzigen System nachvollziehen können. In diesem Buch werden als Initialen die Zeichen `Z4` verwendet. Die Zeichenfolge `kk` steht als Platzhalter für die Kapitelnummer (der zweiten Gliederungsebene), `nn` ist eine laufende Nummer innerhalb des Kapitels. Zu einigen der vorgestellten Programme können bzw. sollen Sie eigene Modifikationen anlegen. Diese Varianten zum Programm werden mittels der letzten Stelle des Namens unterschieden. Die Ursprungsversion erhält immer das Zeichen 0. Das erste Programm bekommt daher den Namen `YZ431010`.

Umfangreiche Programme können Sie in mehrere Teildateien (Includes) aufspalten. Die Entwicklungsumgebung unterstützt diese Programmierweise durch automatisches Anlegen eines Rahmenprogramms, das nur aus Include-

Anweisungen besteht. An dieser Stelle ist diese komplexe Programmstruktur allerdings noch nicht notwendig. Sie sollten daher das Flag MIT TOP INCLUDE deaktivieren. Danach können Sie dieses Popup mit der ↵-Taste oder durch Betätigen des entsprechenden Symbols verlassen. Gegebenenfalls erscheint nun ein Popup, dass Sie darüber informiert, dass das neue Programm im Kundennamensraum liegt. Dies ist allerdings von der konkreten Systemeinstellung abhängig. Sie können es ohne weiteres beenden.

Das neue Programm wird nun vom System angelegt. Dazu benötigt es aber eine Reihe zusätzlicher Informationen, die in einer Eingabemaske abgefragt werden. Diese erscheint wiederum als Popup (Abbildung 3.7). Viele Elemente einer Anwendung, so auch ein einfacher Report, besitzen einige Verwaltungsinformationen. Von besonderer Bedeutung sind dabei zwei Felder, in die auf jeden Fall eine Eingabe erfolgen muss. Bei diesen Feldern handelt es sich um eine Kurzbeschreibung des Programms und den Programmtyp.

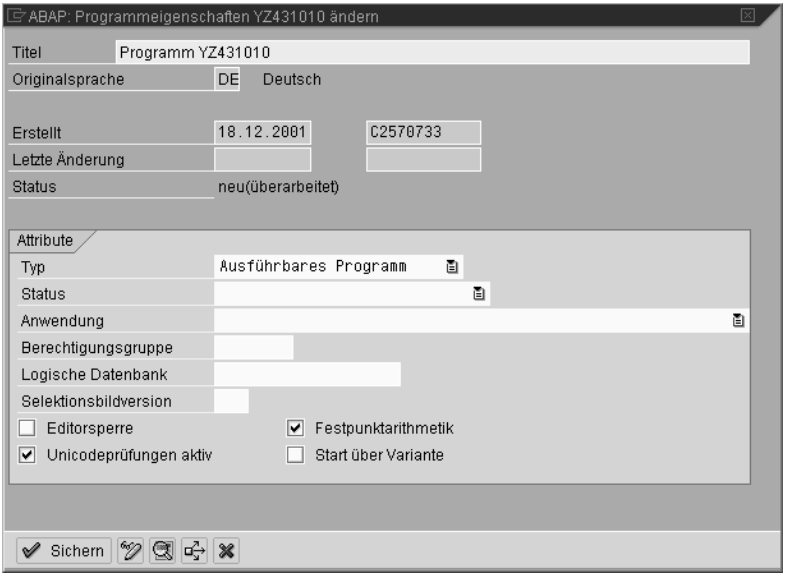

Abbildung 3.7 **© SAP AG**
Pflege der Attribute eines neuen Programms

Die Kurzbeschreibung im Feld TITEL wird von neueren Versionen der R/3-Software automatisch vorbelegt. Sie können diese Vorgabe jederzeit überschreiben, z. B. mit „Hello-World“. Diese Kurzbeschreibung ist in der Praxis unentbehrlich. Wegen der teilweise vorgeschriebenen Programmnamen, deren Länge zudem begrenzt ist und der großen Anzahl von Programmen im SAP-System gibt nur dieser Titel einen echten Hinweis auf die Funktion des Programms. Der Titel erscheint übrigens auch in den automatisch generierten Standardüberschriften der vom Report erzeugten Liste.

In Kapitel 2 wurden bereits verschiedene Programmtypen erwähnt, z. B. Modul-Pools oder die hier verwendeten Reports. Im Feld TYP wird eben dieser Typ des Programms festgelegt. Diese Typangabe ist zwingend erforderlich. Reports besitzen immer den Typ AUSFÜHRBARES PROGRAMM. Nur Programme mit diesem Typ können direkt abgearbeitet werden! Mit dem hier benutzten Werkzeug können durch Wahl des entsprechenden Typs neben Online-Reports auch andere Programmtypen angelegt werden. Abbildung 3.8 zeigt den zur Verfügung stehenden Wertevorrat in Form einer Eingabehilfe.

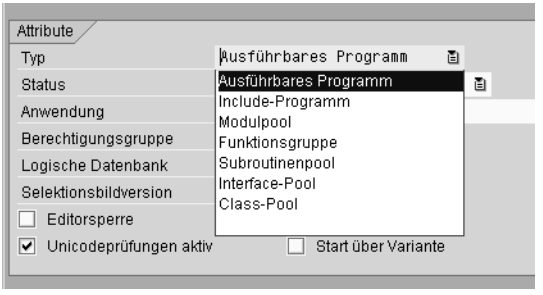

Abbildung 3.8 **© SAP AG**
Eingabehilfe für den Programmtyp

Alle anderen Eingabefelder nehmen optionale Informationen auf. Sie können die vom System getroffenen Vorgaben übernehmen. Nach Eintragen der Attribute für das Programm müssen Sie diese Werte abspeichern. Das geschieht entweder mit der ↵-Taste oder dem entsprechenden Icon. Beim erstmaligen Sichern eines neuen Objekts erfragt das System in einem zum Korrektur- und Transportwesen gehörenden Popup (Abbildung 3.9) den Namen eines so genannten Pakets.

Abbildung 3.9 **© SAP AG**
Popup des Korrektur- und Transportwesens

Pakete dienen zur Gruppierung der verschiedenen Entwicklungsobjekte innerhalb des R/3-Systems. Logisch zusammengehörende Elemente werden einem gemeinsamen Paket zugeordnet. Ein Paket ist eine Weiterentwicklung der aus älteren Systemen bekannten Entwicklungsklasse. Die Eigenschaften eines Pakets bestimmen beispielsweise auch, ob Objekte aus dem aktuellen System heraus in andere Systeme transportiert werden können oder nicht. Üblicherweise werden im R/3-System Programme in einem eigenständigen Testsystem entwickelt. Nach der Entwicklung und einem Programmtest werden die Programme dann mittels des Korrektur- und Transportwesens auf das gewünschte Zielsystem übertragen. Die korrekte Pflege der Eigenschaften der diversen Pakete und die Zuordnung der Anwendungen zu diesen Paketen ist somit von entscheidender Bedeutung für die Weitergabe von Anwendungen an nachfolgende Systeme. Zu dieser Problematik folgt später ein etwas ausführlicherer Abschnitt.

Damit nicht jedes kleine Test- oder Hilfsprogramm in die offizielle Entwicklung eingeht, können Entwicklungsobjekte auch als so genannte *Lokale Objekte* gekennzeichnet werden. Im dargestellten Popup dient dazu eine spezielle Drucktaste. So deklarierte Objekte werden durch das Transportwesen nicht erfasst. Sie können daher auch nicht in andere Systeme übertragen werden. Alle Demoprogramme sollten als lokale Objekte angelegt werden. Sie können das Popup also ohne weitere Eingabe durch Betätigen der Schaltfläche LOKALES OBJEKT beenden. Gleichbedeutend mit diesem Schalter ist die Eingabe des Pakets „$TMP“ im dafür vorgesehenen Eingabefeld. Da im Grundbild des Object Navigator die lokalen Objekte als Arbeitsvorrat ausgewählt wurden, erscheint im Popup der entsprechende Paketname $TMP bereits als Vorgabe.

Dieses Popup wird im weiteren Verlauf der Programmierung für jedes neue Objekt erscheinen. Es ist immer auf die erwähnte Weise zu behandeln. Im weiteren Verlauf des Buches wird nicht mehr auf diesen Arbeitsschritt eingegangen.

Nach Auswahl des Pakets befinden Sie sich wieder im Object Navigator. Das neue Programm wurde in die Objektliste aufgenommen. Im rechten Teil des Bildschirms erscheint nun der Programmeditor (Abbildung 3.10). Er zeigt das momentan nur aus einer einzigen echten Anweisung und einigen Kommentarzeilen bestehende Programm an. Bei Online-Reports generiert das System automatisch eine Programmzeile, die aus dem Schlüsselwort `REPORT` und dem Programmnamen besteht.

Beim Editor handelt es sich um den so genannten Frontend-Editor. Dieser wird durch ein Control im SAPGUI gebildet, die gesamte Programmlogik liegt somit auf dem lokalen Rechner. Dieses vorgehen stellt eine Abkehr vom Editor der älteren R/3-Versionen dar, der auf den Dialogelementen des SAPGUI und der herkömmlichen Dynpro-Technik beruhte. Der optisch überarbeitete, Serverbasierte Editor ist weiterhin verfügbar. Für die Abarbeitung dieses Beispiels ist es bedeutungslos, welcher Editor benutzt wird.

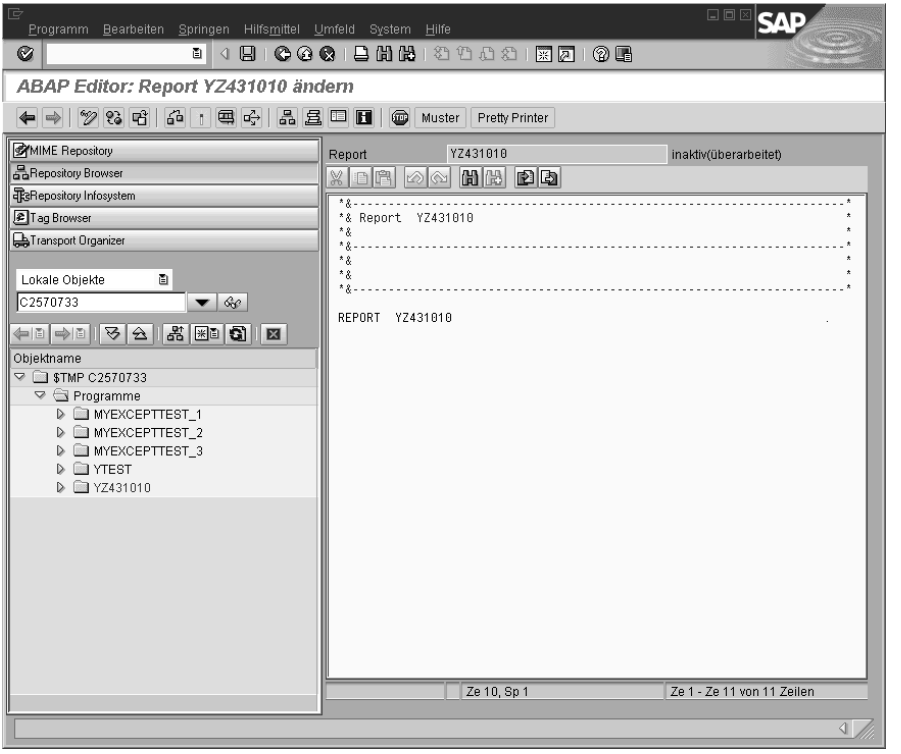

Abbildung 3.10 **© SAP AG**
Editor mit Programmkopf

An dieser Stelle können Sie das Programm erfassen. Dazu müssen Sie nur eine einzige Zeile ergänzen:

```
REPORT YZ331010.
WRITE 'Hello world!'.
```

Durch einen einfachen Mausklick können Sie den Cursor auf die Zeile unter der `REPORT`-Anweisung platzieren. Dann tragen Sie die zusätzliche Anweisung ein. Innerhalb des Editors können Sie mit der ↵-Taste Zeilenumbrüche einfügen. In der ersten Zeile wird dem System mittels des Schlüsselwortes `REPORT` mitgeteilt, dass es sich bei den folgenden Anweisungen um ein eigenständiges Programm handelt. Völlig äquivalent zu `REPORT` ist übrigens die alternativ verfügbare Anweisung `PROGRAM`. Die zweite Zeile schreibt einen Text in das Ausgabefenster.

ABAP-Anweisungen können formatfrei eingegeben werden, also über mehrere Zeilen reichen oder beliebig eingerückt werden. Dies erfordert, das Ende eines ABAP-Kommandos eindeutig zu kennzeichnen. Im ABAP dient dazu ein Punkt.

Nun können Sie die Anwendung direkt aus dem Editor heraus ausführen. Dies erfolgt mittels der Menüfunktion PROGRAMM | TESTEN | DIREKT oder der Funktionstaste F8. Falls sich keine Tippfehler eingeschlichen haben, wird das Editor-Dynpro durch die Programmausgabe ersetzt. Es sollte sich das in Abbildung 3.11 dargestellte Ergebnis darbieten. Die Bildschirmausgabe wird von den Reports nicht direkt durch Ausgabe an das Fenster erzeugt. Vielmehr schreibt ein Report seine Ausgabe in einen Puffer, der erst im Ausgabefenster angezeigt wird, wenn der Report komplett abgearbeitet wurde. Bei komplexeren Reports kann dies einige Zeit dauern!

Abbildung 3.11 **© SAP AG**
Bildschirmausgabe des ersten Programms

Das Puffern der Ausgabe ermöglicht es, längere Listen nach ihrer Erstellung am Bildschirm beliebig zu rollen. Außerdem ist es möglich, den Pufferinhalt und damit die Ausgabeliste zu drucken, ohne dass im Report dazu spezielle Anweisungen stehen müssen.

Neben der erwarteten Ausschrift „Hello world!“ finden Sie in der Ausgabe außerdem den Titel des Reports und eine Seitennummer wieder. Diese Elemente werden automatisch vom System erzeugt, da sie normalerweise auf jeder Druckliste unentbehrlich sind. Vom Ausgabefenster des Reports gelangen Sie mit der Funktionstaste F3 oder dem Symbol mit dem grünen Pfeil zurück zum Editor.

Die Anweisung `REPORT` oder `PROGRAM` kann mit einigen Zusätzen versehen werden, die Auswirkungen auf das Erscheinungsbild der erzeugten Liste haben. Sofort ausprobieren können Sie den Zusatz `NO STANDARD PAGE HEADING`. Er schaltet den automatisch generierten Listenkopf ab.

Umfangreiche Programme werden erst durch Kommentare verständlich. Unter ABAP sind zwei Möglichkeiten zur Kennzeichnung von Kommentaren verfügbar: Ein Stern in der ersten Spalte einer Zeile ist die erste und recht auffällige Variante. Kommentare, die nicht in der ersten Spalte beginnen sollen, also z.B. solche, die in einer Zeile mit einem ABAP-Kommando hinter diesem stehen sollen, beginnen mit dem Anführungszeichen. Die nachfolgenden Zeilen demonstrieren die bisher erläuterten Änderungen. Das Einfügen der zusätzlichen Zeichen ist eine gute Übung für den Umgang mit dem Editor.

```
*&---------------------------------------------------------*
*& REPORT YZ431010                                         *
*&                                                         *
*&---------------------------------------------------------*
*& This program creates a short text                       *
*&                                                         *
*&---------------------------------------------------------*
REPORT yz431010 NO STANDARD PAGE HEADING.
WRITE 'Hello world!'. " text output
```

Zum Abschluss ist ein weiterer Arbeitsschritt, die Aktivierung, notwendig. Bei der Bearbeitung vorhandener Programme müssen oft mehrere Elemente (Dynpros, Includes) bearbeitet werden. Dadurch kann es zwischenzeitlich zu inkonsistenten Zuständen kommen, die eine Ausführung der Anwendung verhindern würden. Aus diesem Grund werden ab Version 4.5A alle Änderungen zunächst in inaktiver Form gespeichert und erst nach Abschluss der Entwicklung aktiviert. Dadurch werden alle Änderungen auf einmal wirksam. Bis zur Aktivierung der neuen Version ist die alte Version der Anwendung aktiv. Ein Programmtest ohne Aktivierung würde daher lediglich die alte Version der Anwendung ausführen. Eine Ausnahme bildet dabei der hier benutzte Aufruf aus dem Editor heraus, der sofort auf den neuen Quelltext zugreift. Das gilt aber nur für das momentan im Editor befindliche Programm, also auch nicht für dessen Includes und andere Elemente!

Die Aktivierung eines Programms erfolgt mit der Menüfunktion PROGRAMM | AKTIVIEREN oder der Funktionstaste ⇧ F8. Sofern, wie in diesem Fall, nur ein inaktives Element existiert, wird es ohne weitere Rückfrage sofort aktiviert. Falls es mehrere inaktive Elemente gibt, müssen Sie in einem Popup die zu aktivierenden Objekte markieren (siehe Abbildung 3.12). Im gezeigten Beispiel existieren zwei zu aktivierende Objekte, der eigentliche Quelltext (Objekt REPS) sowie zusätzliche Textressourcen (Objekt REPT).

Während der Aktivierung findet eine Syntax-Prüfung statt. Eventuelle Syntaxfehler werden durch das System gemeldet. Je nach Art des Fehlers erfolgt dies durch eine einfache Meldung im Dynpro oder ein Popup, das mittels spezieller Drucktasten sofort den Sprung zur Position des Fehlers einleiten kann. Sofern Sie mit einer älteren Version arbeiten, bei der noch keine Aktivierung erforderlich ist, erfolgt die Syntax-Prüfung beim Start der Anwendung.

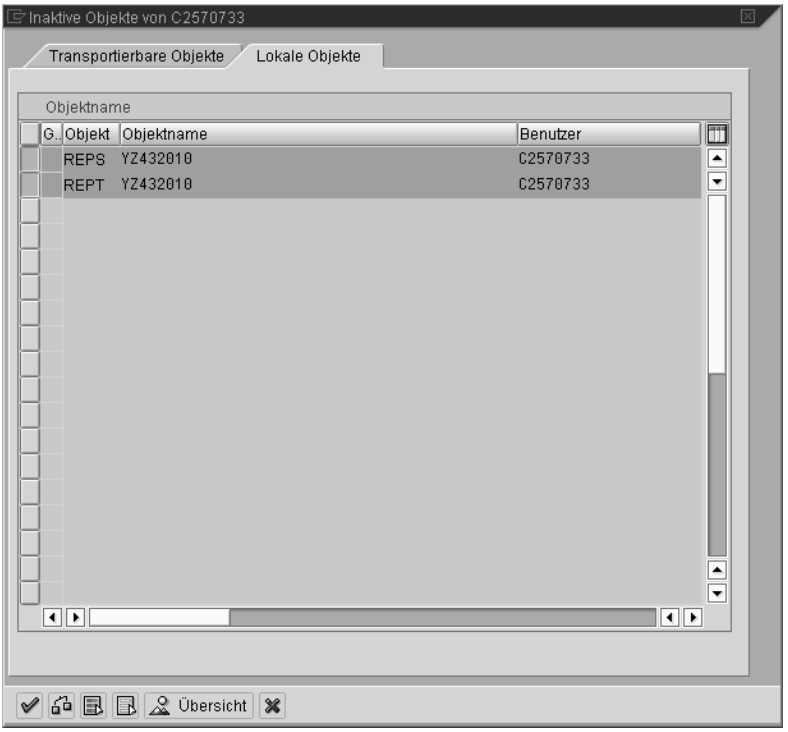

Abbildung 3.12 **© SAP AG**
Aktivieren von Objekten

Die Aktivierung darf nicht mit der Generierung einer Anwendung verwechselt werden. Durch die Aktivierung legen Sie fest, welche Quelltextversion die gültige Version ist. Bei der Generierung hingegen erzeugt das System aus dem aktiven Quelltext einen Zwischencode, der schneller ausgeführt werden kann.

Ausgabe einer Textzeile

Viele ABAP-Kommandos können mit mehreren Parametern aufgerufen werden. Im Falle des einzigen bisher benutzten Kommandos, der `WRITE`-Anweisung, bedeutet dies, dass mehrere Zeichenketten gleichzeitig mit einer Anweisung auf die Ausgabeliste ausgegeben werden können. Dazu ist allerdings eine kleine Erweiterung der Syntax nötig. Ein Doppelpunkt unmittelbar hinter dem eigentlichen Schlüsselwort leitet die Parameterliste ein; die einzelnen Parameter sind durch Kommata voneinander zu trennen.

```
WRITE: 'Das','sind','mehrere','Zeichenketten'.
```

Die Doppelpunktvariante existiert auch für viele andere Kommandos, nicht nur für WRITE. Leider existieren in den offiziellen Syntaxbeschreibungen keine Hinweise darauf, welche Kommandos die Doppelpunkt-Schreibweise unterstützen.

Mehrere fest kodierte Texte als einzelne Parameter anzugeben macht natürlich wenig Sinn. Allerdings wird aus diesem Beispiel, sofern Sie es ausführen, ersichtlich, dass mehrere Parameter bei der Ausgabe jeweils durch ein automatisch eingefügtes Leerzeichen voneinander getrennt werden. Interessanter wird die Parameterfähigkeit der WRITE-Anweisung erst mit zusätzlichen Argumenten zur Formatierung der Ausgabe, insbesondere zur gezielten Positionierung der auszugebenden Zeichenketten innerhalb einer Zeile. Mittels vorangestellter Zahlenwerte der Form

```
column(length)parameter
```

wird der Parameter in der angegebenen Spalte in einem Feld mit der in Klammern eingetragenen Länge ausgegeben. Beide Angaben sind wahlfrei. Ein führender Schrägstrich bewirkt einen Zeilenwechsel. Auch dazu ein Beispiel:

```
WRITE: / 'Das',/5'sind',/10(4)'mehrere',/(7)'Zeichenketten'.
```

Neben der Positionierung erlauben weitere Optionen eine sehr leistungsfähige Aufbereitung der auszugebenden Werte, z. B. die Formatierung gemäß einer Maske oder die Ausrichtung innerhalb einer Spalte. Diese Optionen an dieser Stelle zu beschreiben würde aber mehr Verwirrung stiften als Nutzen bringen. Nähere Informationen finden Sie im Abschnitt zur Report-Programmierung. Datenfelder und Ausdrücke

Die Möglichkeit, einfache Zeichenketten auszugeben, reicht für echte Anwendungen natürlich nicht aus. Unbedingt erforderlich ist die Möglichkeit, mit Daten umzugehen und den Programmablauf steuern zu können. Die dazu notwendigen Anweisungen werden Sie in den folgenden Abschnitten detailliert kennen lernen. Als kleiner Einstieg soll das folgende Programm dienen, das in einer Tabelle das kleine 1x1 ausgibt. Im weiteren Verlauf dieses Abschnitts dient dieses Programm auch als Testprogramm für den Debugger.

```
REPORT yz431030 NO STANDARD PAGE HEADING.
DATA:
  row TYPE i VALUE 1,
  col TYPE i,
  res TYPE i.

DO 10 TIMES.
  IF row = 1.
    WRITE: (5) space.
    col = 1.
    DO 10 TIMES.
      WRITE: (5) col.
      col = col + 1.
```

```
    ENDDO.
    WRITE: / sy-uline(80).
  ENDIF.
  col = 1.
  WRITE: /(5) row.
  DO 10 TIMES.
    res = col * row.
    WRITE: (5) res.
    col = col + 1.
  ENDDO.
  row = row + 1.
ENDDO.
```

Dieses Programm deklariert mittels der Anweisung `DATA` die drei Datenfelder `col`, `row` und `res`. Alle drei Felder sind vom Typ `INTEGER`. Der Wert für `ROW` wird durch den Zusatz `VALUE` mit 1 vorbelegt.

Die Anweisung `DO 10 TIMES` eröffnet eine Schleife, die genau zehnmal durchlaufen wird. Die Anweisung `ENDDO` am Ende des Programms beendet diese Schleife.

Während des ersten Durchlaufs durch diese Schleife wird zunächst ein Tabellenkopf ausgegeben. Als Kennzeichen dafür dient der Wert für `row`, der während des ersten Durchlaufs 1 ist. Geprüft wird dieser Zustand durch die Anweisung `IF`. Diese Anweisung prüft einen logischen Ausdruck (`row = 1`). Ist dieser Ausdruck wahr, dann werden die Anweisungen zwischen `IF` und `ENDIF` ausgeführt. Für die Erstellung des Tabellenkopfes wird zunächst mit

```
WRITE: (5) SPACE
```

ein 5 Stellen breites leeres Feld ausgegeben. Unter diesem Feld stehen in den anderen Zeilen die Bezeichner der Tabellenzeilen. Anschließend wird `col` mit dem Wert 1 für die erste Spalte belegt. In einer zweiten Schleife wird der Wert für `col` in einem 5 Zeichen breiten Ausgabefeld ausgegeben und anschließend um 1 erhöht. Nach Ausgabe aller 10 Spaltenbezeichnungen erfolgt ein Zeilenvorschub und die Ausgabe einer durchgehenden Linie zur Abgrenzung des Tabellenkopfes vom Tabellenrumpf. Dies erfolgt mit der Anweisung

```
WRITE: / sy-uline(80).
```

Das Feld `sy-uline` ist eines der vordefinierten Systemfelder. Es enthält das Zeichen zum Erzeugen einer horizontalen Trennlinie.

Alle nun folgenden Anweisungen werden für jeden Durchlauf durch die äußere Schleife ausgeführt. Der Wert für `col` wird wieder auf 1 gesetzt. Anschließend wird die Zeilennummer in einem 5 Zeichen breiten Feld ausgegeben. Das „/"-Zeichen in der `WRITE`-Anweisung sorgt dafür, dass diese Ausgabe am Anfang einer neuen Zeile beginnt.

Es schließt sich wieder eine DO-Schleife an, in der aus Zeilen- und Spaltennummer das Produkt errechnet wird. Dieses Ergebnis wird dann ausgegeben. Hier fehlt das „/"-Zeichen, da alle Ausgaben in einer Zeile stehen sollen. Die Verwendung des Hilfsfeldes res ist notwendig, da das WRITE-Kommando keine Ausdrücke als Parameter verarbeiten kann. Vor dem abschließenden ENDDO wird die Zeilennummer um 1 erhöht.

Übungen

1. Ergänzen Sie das erste Programm mit Kommentaren.
2. Legen Sie eigene Programme an, um die beschriebenen Varianten der WRITE-Anweisung zu testen.
3. Testen Sie das 1x1-Programm.

3.1.2 Die Entwicklungsumgebung

Der vorangegangene Abschnitt demonstrierte alle Schritte zur Erstellung einer einfachen Anwendung. Bevor Sie weitere Programme erzeugen, in denen die diversen ABAP-Kommandos näher beschrieben werden, lernen Sie in diesem Abschnitt die Entwicklungsumgebung etwas genauer kennen.

Objektlisten

Der bisher benutzte Repository Browser stellt als Teil des Object Navigator zwar den zentralen Teil der Entwicklungsoberfläche dar, neben ihr existieren aber weitere wichtige Werkzeuge. Daher erfolgt der Einstieg in die Programmentwicklung nicht unbedingt über den Object Navigator und den Repository Browser sondern in bestimmten Fällen auch über das bereits erwähnte Bereichsmenü der Workbench. Der Object Navigator ist selbst wieder eine Zusammenfassung von Entwicklungswerkzeugen. Er nimmt innerhalb der Entwicklungsoberfläche eine Sonderstellung ein. Während die anderen Werkzeuge jeweils ein Element einer bestimmten Art bearbeiten, bietet der Object Navigator bzw. dessen Teilbereiche einen Überblick über alle Bestandteile einer Anwendung. Über ihn sind daher indirekt auch fast alle anderen Werkzeuge erreichbar. In der Praxis erfolgt der Einstieg in die Entwicklung daher fast immer über dieses Werkzeug, das nicht nur über die Menüs der Workbench, sondern auch durch den direkten Aufruf der Transaktion SE80 gestartet werden kann.

Die Elemente der Entwicklungsumgebung werden, wie bereits erwähnt, so genannten Paketen zugeordnet. Ein Paket nimmt alle Objekte auf, die zu einem Projekt oder einer komplexen Anwendung gehören. Dabei kann es sich durchaus um mehrere Programme und weitere Objekte handeln. Das Paket bildet die oberste Ebene in der Hierarchie der Entwicklungsobjekte. Die Abbildung 3.13 zeigt Ihnen als Beispiel die Struktur der SAP-eigenen Entwicklungsklasse

SBF_WEB. Innerhalb der Objektliste des Repository Browsers wird unterschieden zwischen Elementen und Objektgruppen. Eine Objektgruppe ist kein real existierendes eigenständiges Objekt sondern fasst mehrere Elemente zusammen. Elemente wiederum sind die echten Bestandteile einer Anwendung. Sie werden auch als Entwicklungsobjekte bezeichnet. Die diversen Objektgruppen werden Sie nach und nach kennen lernen, sie sollen hier nicht näher beschrieben werden.

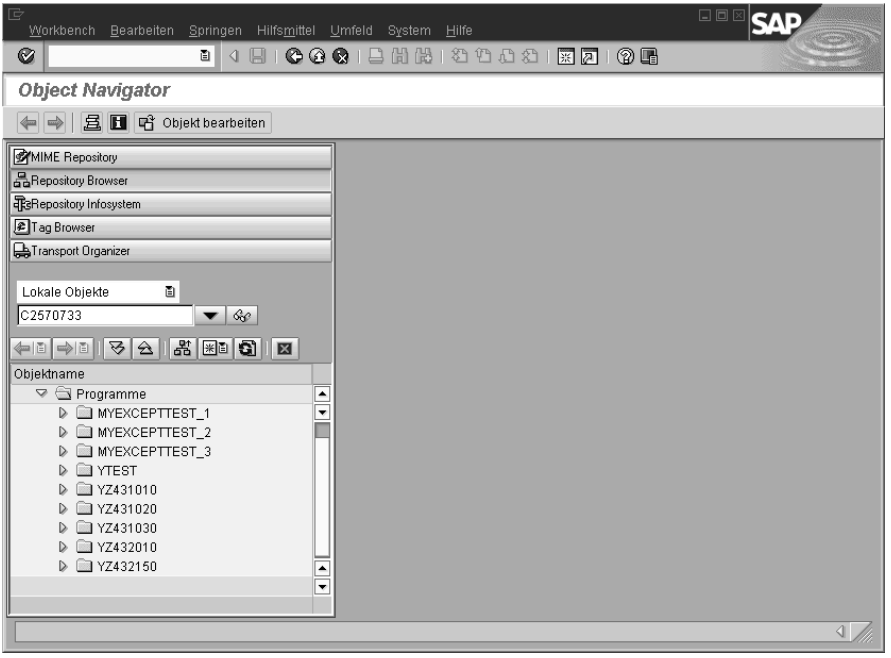

Abbildung 3.13 **© SAP AG**
Beispiel für die erste Ebene einer Objektliste

Durch einen einfachen Mausklick auf das Mappen-Symbol vor dem Namen der Objektgruppen können Sie die einzelnen Zweige der Objekthierarchie öffnen oder schließen. Bei einigen der Gruppen, z. B. Programmen oder Funktionsgruppen, erhalten Sie bereits in der nächsten Ebene die Auflistung der jeweiligen Elemente. Andere Gruppen wie die der DDIC-Objekte sind nochmals unterteilt.

Durch einen Doppelklick auf ein Element innerhalb der Objektliste, abgesehen von Programmen und Funktionsgruppen, starten Sie das Werkzeug zu dessen Bearbeitung. Bei Programmen und Funktionsgruppen führt Sie der Doppelklick nicht direkt zum Quelltexteditor. Vielmehr gelangen Sie in eine zweite Objektliste, in der dann nur die Teilelemente des gewählten Programms oder der Funktionsgruppe enthalten sind. Da Programme und Funktionsgruppen selbst sehr komplexe Objekte darstellen, dient diese Form der Darstellung der besseren Übersichtlichkeit. Gleichzeitig wird das Menü ausgetauscht, um Funktionen anzubieten, die auf die Programm-Objekte angepasst sind.

Anlegen neuer Elemente

Der Repository Browser verfügt über einen sehr großen Funktionsumfang. So können Sie auch innerhalb der Objektlisten beliebige neue Objekte anlegen. Dazu platzieren Sie den Cursor auf ein Element oder eine Objektgruppe und rufen durch einen Klick mit der rechten Maustaste das Kontextmenü auf. Wenn der Repository Browser die Art des neuen Elements anhand des Objekts an der Cursorposition ermitteln kann, so blendet er sofort ein Popup ein, in dem Sie das neue Element oder Teilelement genauer spezifizieren können. Stand der Cursor beim Aufruf der ANLEGEN-Funktion beispielweise auf einem Programm oder der Objektklasse für Programme dann bezieht sich das Popup auf Programmobjekte. Dies wurde beim ersten Beispiel bereits deutlich.

Falls der Repository Browser beim Versuch, ein neues Objekt anzulegen, die erforderliche Objektklasse nicht bestimmen kann, stehen die diesbezüglichen Auswahlmöglichkeiten im Kontextmenü zur Verfügung.

Grundeinstellungen

Innerhalb des Repository Browsers, aber auch in vielen anderen Werkzeugen, steht Ihnen die Menüfunktion HILFSMITTEL | EINSTELLUNGEN zur Verfügung. Abhängig vom Ursprung des Aufrufs gelangen Sie immer an ein Popup, das Ihnen Einstellmöglichkeiten für die aktuelle Anwendung anbietet. Im Fall der Entwicklungsumgebung erhalten Sie ein Popup, das mehrere, durch Reiter (Tab Strips) wählbare Teilbilder enthält, die selbst wieder in Teilbereiche untergliedert sein können.

In Abbildung 3.14 sehen Sie die Einstellungen für den Quelltext-Editor. Nähere Erläuterungen dazu finden Sie im nächsten Abschnitt.

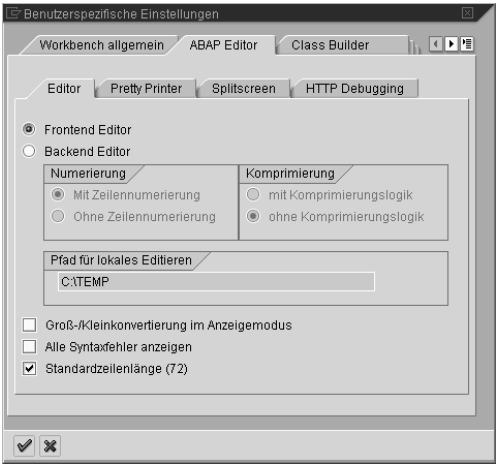

Abbildung 3.14 **© SAP AG**
Grundeinstellungen für die Entwicklungsumgebung

Funktionen auf Elemente

Innerhalb der Objektliste des Repository Browsers können Sie diverse Funktionen auf die Elemente anwenden. Allerdings können nicht alle zur Verfügung stehenden Funktionen auf alle Elemente angewendet werden. Außerdem ist die Reaktion des Browsers nicht nur von der gewählten Funktion, sondern auch vom jeweiligen Element abhängig. Die meisten der elementbezogenen Funktionen stehen in der aktuellen Version der Entwicklungsumgebung in einem, ggf. mehrere Ebenen umfassenden Kontextmenü bereit. Um eine Funktion auf ein Element anzuwenden, platzieren Sie den Cursor auf den Namen des Objekts und rufen das Kontextmenü durch einen Klick mit der rechten Maustaste auf. Das Neuanlegen und Ändern wurde bereits erwähnt. Darüber hinaus können Sie Objekte löschen, umbenennen oder kopieren. Beim Kopieren und Umbenennen bietet der Browser je nach Typ des gewählten Elements ein oder mehrere Popups an, in denen die Namen des Ziels sowie gegebenenfalls die ebenfalls zu kopierenden Teilelemente festgelegt werden.

Neben den echten Bearbeitungsfunktionen sind eine Reihe von Hilfsfunktionen wie Aktivieren, Transportieren oder Prüfen verfügbar. Diese sind im eigentlichen Menü zu finden.

Testen von Programmen

Innerhalb des Object Navigator besteht die Möglichkeit, neu erstellte Anwendungen sofort aus der Entwicklungsumgebung heraus zu testen. Dazu dient die Menüfunktion TESTEN | DEBUGGING. Sie haben dabei sogar die Möglichkeit, zwischen dem normalen Ablauf und dem Start im Debug-Modus zu wählen (Abbildung 3.15).

Die Ausführungsart *Direkt* übergibt die Kontrolle an das zu testende Programm. Es startet sofort und läuft im Vordergrund. Sie können wie gewohnt mit dem Programm arbeiten. Von erheblicher Bedeutung während der Programmentwicklung ist die Ausführungsart *Debugging*. Sie bewirkt, dass das auszuführende Programm in den Debugger geladen und unter dessen Kontrolle ausgeführt wird. Näheres zur Bedienung des Debuggers erfahren Sie im Abschnitt 3.1.4.

Der Testmodus steht nicht nur für Programme sondern auch für andere Elemente zur Verfügung. So können beispielsweise Funktionsbausteine separat aus der Entwicklungsumgebung heraus getestet werden, ohne dass für den Test eine eigene Anwendung geschrieben werden muss. Aber auch Elemente, die nicht aus ABAP-Quelltext bestehen, können über eine Testfunktion verfügen. Die Reaktion des Browsers hängt dabei stark vom gewählten Element ab. Bei der Anwendung auf Tabellen wird beispielsweise der Datenbrowser aufgerufen, der es Ihnen ermöglicht, den Inhalt der Tabelle zu analysieren.

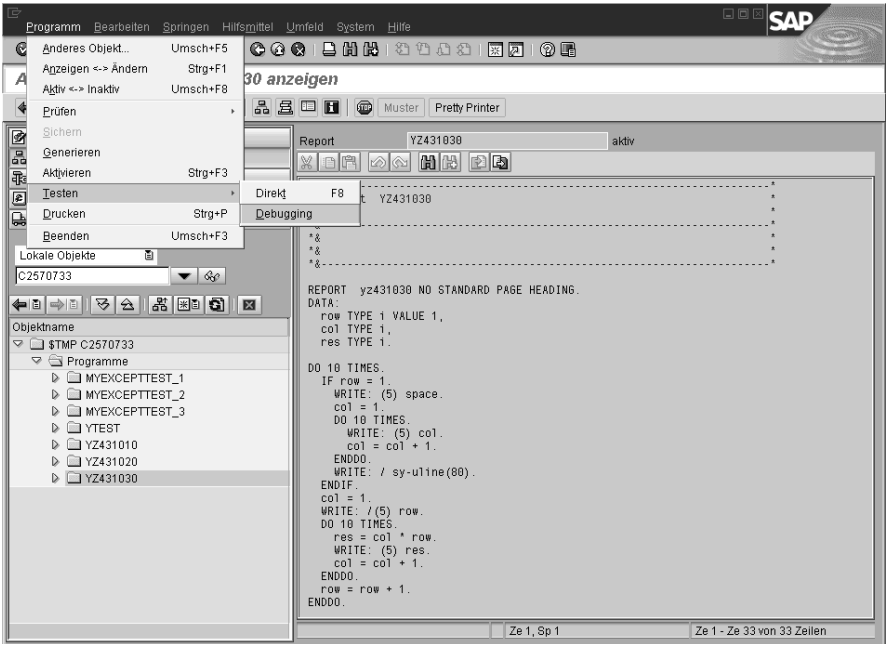

Abbildung 3.15 **© SAP AG**
Start eines Programms aus der Entwicklungsumgebung heraus

3.1.3 Der Editor

Die Entwicklungsumgebung des R/3-Systems unterstützt inzwischen zwei prinzipiell unterschiedliche Editoren. Die Auswahl ist über das bereits erwähnte Popup für die Grundeinstellungen möglich. Einer der Editoren beruht auf der elementaren Dynpro-Technik (Abbildung 3.16). Er ist ebenso wie viele andere Werkzeuge des Systems ein ABAP-Programm, die Oberfläche des Editors demzufolge ein Dynpro. Da ABAP in Dynpros momentan nur einzeilige Eingabefelder zur Verfügung stellt, arbeitet der Editor zeilenorientiert. Er nutzt jedoch alle Möglichkeiten, die Dynpros bieten, konsequent aus. Die Funktionalität des R/3-Programmeditors ähnelt daher weitgehend derjenigen modernerer, seitenorientierter Exemplare. Trotzdem sind einige Einschränkungen vorhanden, z.B. der fehlende automatische Zeilenumbruch.

Der eigentliche Arbeitsbereich besteht aus mehreren Zeilen. Jede Zeile ist durch eine Zeilennummer gekennzeichnet. Die Nummerierung wird vom System automatisch durchgeführt. Die Länge der Eingabezeile ist auf das sichtbare Maß (72 Zeichen) beschränkt. Es findet kein automatischer Zeilenumbruch statt; jede Eingabezeile wird für sich editiert.

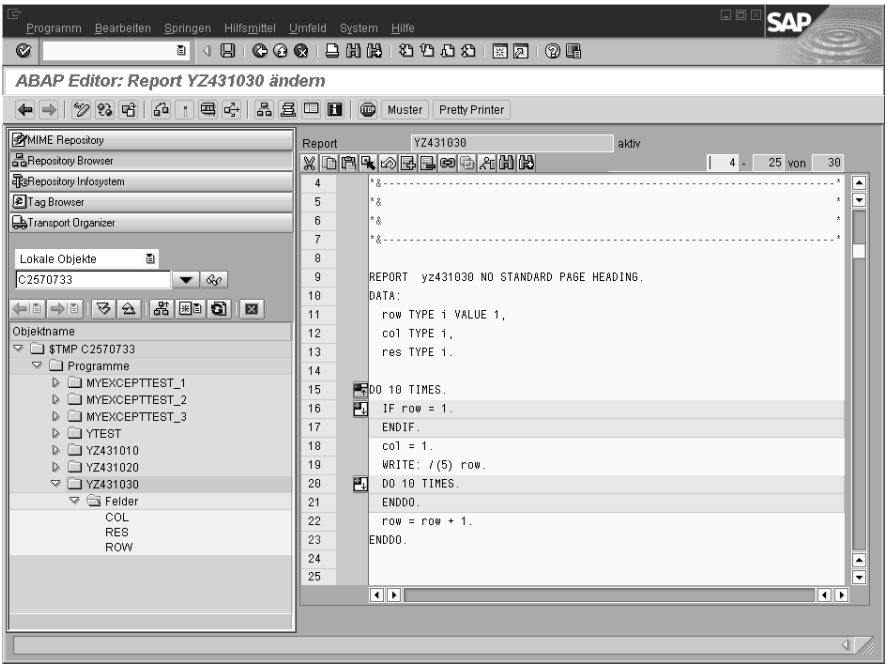

Abbildung 3.16 **© SAP AG**
Der Backend-Editor mit Komprimierungslogik

Befindet sich der Editor im Ändern-Modus, so sind die eingabebereiten Zeilen farbig so hervorgehoben wie eingabebereite Felder in einem Dynpro (Standard: schwarze Schrift auf weißem Hintergrund). Im Anzeigen-Modus erscheint der Text hingegen in derselben Darstellung wie die Schlüsselfelder eines Dynpros (Standard: schwarze Schrift auf grauem Hintergrund).

In der rechten unteren Ecke des Bildschirmfensters erscheinen die aktuellen Zeilennummern. Sie können direkt zu einer Zeile springen, in dem Sie die gewünschte Zeilennummer in das Eingabefeld eintragen und die ↵-Taste betätigen.

In der Statuszeile werden Statusinformationen angezeigt. Wichtig ist z.B. die Unterscheidung zwischen Einfüge- und Überschreibmodus.

Der Klick auf die rechte Maustaste blendet ein Menü mit der aktuellen Funktionstastenbelegung ein. Die Einträge können mit der Maus selektiert werden. Alternativ dazu ist die Betätigung der jeweiligen Funktionstaste möglich. R/3 benutzt bis zu 24 Funktionstasten. Da auf PC-Tastaturen meist nur 12 Funktionstasten zur Verfügung stehen, werden die Funktionstasten ab F13 durch die Kombination der Umschalttaste mit einer Funktionstaste nachgebildet. Die Kombination ⇧ F1 entspricht beispielsweise F13. Ebenso ist es möglich, an-

stelle der Funktionstasten bei gedrückter Control-Taste (Ctrl oder Strg) die zweistellige Nummer der Funktionstaste einzugeben. Auf diese Weise sind in Systemen ab Release 3.0 auch weitere Funktionen verfügbar. Einige der wichtigsten Funktionen sind auch über Symbole in der Drucktastenzeile verfügbar.

Der Cursor kann entweder mit der Maus oder aber mit den Cursortasten platziert werden. Dabei wird im Text nicht vertikal geblättert, der Cursor springt von der untersten in die oberste Zeile und umgekehrt. Mit dem Tabulator bzw. Umschalt-Tabulator wird zwischen den Eingabeelementen des Programmeditors (Quelltextfelder, Zeilennummernfeld) hin- und hergewechselt. Zum Editieren des Programmtextes können Sie zunächst die in Tabelle 3.1 aufgeführten Tasten benutzen.

Taste	Funktion
INS (Einfg)	Umschalten zwischen Einfüge- und Überschreibmodus.
DEL (Entf)	Löschen des Zeichens rechts neben dem Cursor.
<-	Löschen des Zeichens links neben dem Cursor.
Enter	Teilt die Zeile an der Cursorposition bzw. fügt neue Zeile ein.
CTRL-63	Löscht Zeile, in welcher der Cursor steht.
F5	Dupliziert Zeile, in welcher der Cursor steht bzw. den markierten Block.
F7	Verbindet die aktuelle und die nachfolgende Zeile, falls der Platz reicht.
F9	Markiert Zeile bzw. Block.

Tabelle 3.1
Wichtige Editortasten

Mittels der Menüfunktion PROGRAMM | SICHERN oder dem Sichern-Symbol wird ein Programm abgespeichert. Die Funktion PROGRAMM | PRÜFEN hingegen prüft ein Programm auf Syntaxfehler. Dabei stehen unterschiedliche Prüfmechanismen zur Verfügung, wobei die einfache Syntaxprüfung die am meisten verwendete Funktion ist. Eine derartige Prüfung sollte immer ausgeführt werden, bevor ein Programm gespeichert wird. Programme, die Syntaxfehler enthalten, sollten niemals abgespeichert werden! Derartige Fehler bereiten in den hier vorgestellten Demoprogrammen zwar keine Probleme, allerdings kann das Abspeichern von echten Anwendungsprogrammen, die Syntaxfehler enthalten, unter ungünstigen Umständen zur Funktionsunfähigkeit ganzer Anwendungsgruppen führen.

Hilfestellung zu ABAP-Kommandos erhalten Sie entweder durch eine kontextsensitive Hilfe mittels der F1-Taste oder den Aufruf des Hilfesystems mit der Menüfunktion HILFSMITTEL | HILFE ZU... .

ABAP-Programme unterscheiden nicht zwischen Klein- und Großschreibung. Beim Abspeichern eines Programms werden alle Buchstaben, die nicht in Zeichenkettenkonstanten oder Kommentaren stehen, in Großbuchstaben umgesetzt. Die Darstellung innerhalb des Editors kann wahlweise in Groß- oder Kleinbuchstaben erfolgen. Die Auswahl der Schreibweise erfolgt in dem Popup, in dem auch der Editormodus gewählt wird. In einigen R/3-Versionen ist auch die Wahl zwischen Großschreibung für ABAP-Schlüsselworte und Kleinschreibung für alle anderen Anweisungen möglich.

Eine sehr interessante Funktion ist die Komprimierungslogik. Wenn diese Funktion eingeschaltet ist, erscheinen rechts neben allen Anweisungen, die einen Anweisungsblock einleiten, kleine Symbole (siehe Abbildung 3.15). Durch Betätigen dieser Symbole können Sie den Anweisungsblock ein- und ausblenden. Auf diese Weise ist es möglich, alle nicht interessierenden Teile eines Programms zu verbergen und nur den momentan interessierenden Teil des Quellcodes ausführlich darzustellen. Da die erste und letzte Zeile des Anweisungsblocks (oft eine `IF-`, `CASE-`, `LOOP-` oder `SELECT-`Anweisung) immer dargestellt werden, erhalten Sie so einen sehr guten Überblick über die Struktur einer Anwendung. Sie können die Übersichtlichkeit noch weiter verbessern, wenn Sie vor derartigen Anweisungen einen Kommentar einfügen, der die Aufgaben des folgenden Anweisungsblocks beschreibt.

Bei der anderen Variante des Editors, dem so genannten Frontend-Editor, handelt es sich um ein Control, das im SAPGUI enthalten ist. Dessen Funktionalität entspricht der einfacher Editoren aus dem Windows-Umfeld. Die beim Backend-Editor erwähnten Probleme (Zeilenumbruch, Scrollen) sind hier nicht zu beobachten. Allerdings fehlt die Komprimierungslogik.

3.1.4 Der Debugger

Während der Programmentwicklung erleichtert ein Debugger die Fehlersuche erheblich. Die ABAP-Entwicklungsumgebung verfügt natürlich über ein solches Werkzeug. Mit ihm kann eine Anwendung auf Quelltextebene schrittweise abgearbeitet werden.

Starten des Debuggers

Der Debugger kann durch unterschiedliche Kommandos gestartet werden. Es stehen zum direkten Aufruf entsprechende Menüfunktionen in der Entwicklungsoberfläche sowie statische und dynamische Breakpoints zur Verfügung.

Beim Start einer Anwendung über die Entwicklungsumgebung bietet Ihnen das System verschiedene Ausführungsarten. Eine der Ausführungsvarianten startet das auszuführende Programm im Debugger.

Mitunter ist es nicht notwendig, die Anwendung von Anfang an zu debuggen. Gerade bei Dialoganwendungen interessiert meist das Verhalten eines bestimm-

ten Dynpros. In diesem Fall kann der Debugger auch nachträglich gestartet werden. Dies erfolgt durch Eingabe des Kommandos „/h“ im OK-Feld. Nach betätigen der Datenfreigabetaste informiert eine Ausschrift in der Statuszeile darüber, dass der Debug-Modus eingeschaltet wurde. Die nächste Aktion, die zur Abarbeitung eines Kommandos führt, also jede, mit der die Eingabe in das Dynpro beendet und der PAI-Teil ausgeführt wird, startet den Debugger. Anstelle des nächsten Dynpros erscheint dann die Oberfläche des Debuggers.

Für Funktionsbausteine existiert eine eigene Testumgebung, in der unterschiedliche Ablaufarten gewählt werden können, zu denen auch ein Debug-Modus gehört.

Das Kommando „/h“ kann natürlich auch in Reports benutzt werden. Das ist bei dieser Programmart aber nur im Selektionsbild oder nach der Anzeige einer Teilliste im interaktiven Reporting sinnvoll, da bei einfachen Reports die Liste erst nach Abarbeitung des gesamten Programms angezeigt wird.

Beide bisher erwähnten Möglichkeiten des Debugger-Aufrufs lassen keine exakte Wahl des Startpunktes zu. Die direkte Platzierung eines oder mehrerer Haltepunkte (Breakpoints) im Programm ermöglicht eine solche Wahl. Für die Definition eines Breakpoints existieren wiederum unterschiedliche Varianten. Zunächst kann im Programm die Anweisung

```
BREAK-POINT.
```

notiert werden. Beim Erreichen dieser Anweisung wird der Debugger aktiviert. Diese Anweisung darf natürlich nicht in produktiven Anwendungen enthalten sein, sondern ist nur in der Entwicklungsphase sinnvoll. Eine etwas abgeschwächte Variante dieser Anweisung ist

```
BREAK user.
```

Dieser Breakpoint ist nur wirksam, wenn der Name des aktuellen Benutzers mit dem Argument der BREAK-Anweisung übereinstimmt. Alle anderen Anwender können das Programm ohne Unterbrechung ausführen. Die beiden genannten Anweisungen können in beliebiger Anzahl eingesetzt werden.

Breakpoints können allerdings nicht nur durch hart programmierte Programmanweisungen, sondern auch dynamisch innerhalb des Editors oder des Debuggers gesetzt werden. Im Programmeditor ist eine Menüfunktion HILFSMITTEL | BREAKPOINTS | SETZEN/LÖSCHEN, in einigen Editormodi auch eine entsprechende Drucktaste verfügbar. Sie setzt in der Zeile, in der sich der Cursor befindet, einen Breakpoint. Auch diese Variante kann beliebig oft benutzt werden. Die so gesetzten Breakpoints bleiben allerdings nur während einer Anwendersitzung erhalten.

Innerhalb des Debuggers werden Breakpoints durch einen Doppelklick vor dem ersten Zeichen einer Programmzeile gesetzt oder gelöscht. Ab Release 3.0 werden sie durch ein Symbol (Stoppzeichen) kenntlich gemacht. Ältere Releasestände benutzen dazu den Kleinbuchstaben „b”.

Einige Sonderformen der Breakpoints werden dynamisch ausgewertet. Sie sind nicht an spezielle Zeilennummern, sondern an Ereignisse geknüpft. Als auslösende Ereignisse sind der Sprung in ein Unterprogramm, das Erreichen eines ausgewählten Kommandos, die Wertänderung in einem Datenfeld oder ein Wert ungleich 0 für SY-SUBRC möglich. Die Einstellungen dazu können nur innerhalb des Debuggers mit der Menüfunktion BREAKPOINT | BREAKPOINTS BEI erfolgen.

Bedienung des Debuggers

Beim Sprung in den Debugger ist zunächst dessen Vorzugsmodus zu sehen (Abbildung 3.17). Dieser Modus ermöglicht die Darstellung einzelner Feldinhalte.

Innerhalb dieses Modus erscheint im mittleren Bereich des Bildschirms ein Ausschnitt aus dem gerade abgearbeiteten Programm. Die als nächstes auszuführende Anweisung ist mit einem nach rechts gerichteten Pfeil gekennzeichnet. In der obersten Zeile des Arbeitsbereichs, also unmittelbar unter der Drucktastenzeile, erscheinen einige Schaltflächen, die den Wechsel zwischen den verschiedenen Debugger-Modi ermöglichen. Unter der Zeile mit den Schaltflächen finden Sie zwei Zeilen mit Statusinformationen. Unterhalb des Programmbereichs befindet sich eine vierzeilige Tabelle, die zur Darstellung von Feldinhalten dient. Darunter finden Sie die Darstellung dreier wichtiger Systemfelder.

Der Debugger kann in verschiedenen Anzeigemodi arbeiten. Die in diesen Modi verfügbaren Kommandos bezüglich der weiteren Programmabarbeitung sind nahezu identisch. Die Debugger-Modi unterscheiden sich vor allem in den dargestellten programmbezogenen Informationen. Der Vorzugsmodus, der beim Sprung in den Editor aktiv ist, ist der mit der Anzeigemöglichkeit für Datenfelder.

Vor der Beschreibung der unterschiedlichen Darstellungsmodi des Debuggers sollen die allgemein in allen Modi verfügbaren Kommandos beschrieben werden. Sie sind bis auf wenige Ausnahmen im Interesse einer einfachen Bedienung über Drucktasten in der Drucktastenzeile oder Funktionstasten verfügbar. Zu allen Tasten existieren natürlich auch Menüfunktionen. Effektive Arbeit erfordert allerdings die Verwendung der Druck- oder der Funktionstasten. Die zu beschreibenden Funktionen stehen auch in älteren Releaseständen zur Verfügung, sind dort aber nicht unbedingt auf dieselbe Art und Weise aufzurufen.

Mit der Funktion (F5) wird die markierte Anweisung abgearbeitet. Dabei wird in Unterprogramme oder Funktionsbausteine verzweigt, diese können also auch Anweisung für Anweisung abgearbeitet werden. Die Funktion AUSFÜHREN (F6) hingegen führt das markierte Kommando aus, ohne dabei in Modularisierungseinheiten zu verzweigen. Mit WEITER (F8) wird die schrittweise Abarbeitung beendet, das Programm läuft bis zum nächsten Breakpoint, zur Position des Cursors oder bis zum Programmende weiter. Es gilt jeweils der erste erreichte Unterbrechungspunkt. Mit RETURN (F7) ist die Beendigung der Abarbeitung eines Unterprogramms oder Funktionsbausteins möglich. Der Debugger arbeitet das Unterprogramm komplett ab und hält im übergeordneten Programm, also an der Anweisung nach dem Unterprogrammaufruf, wieder an.

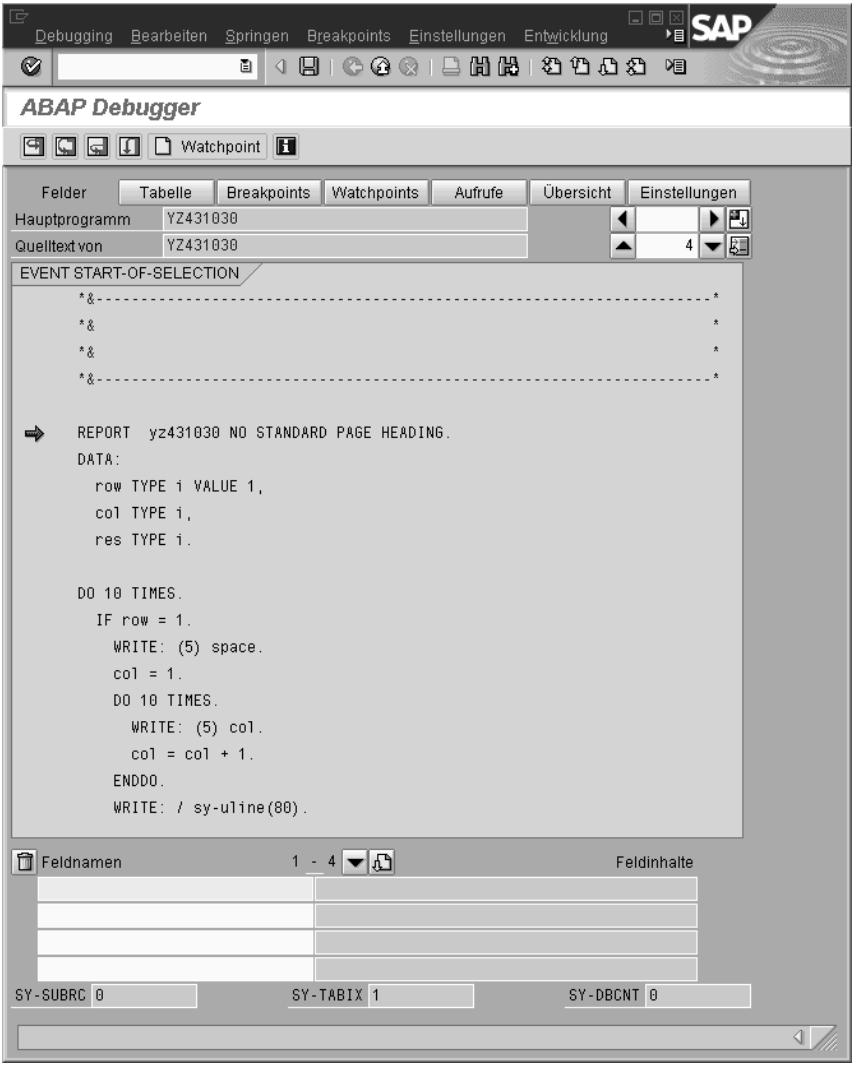

Abbildung 3.17 **© SAP AG**
Vorzugsmodus des Debuggers

Neben den eigentlichen Menüfunktionen bzw. den Icons im Debuggerfenster können die erwähnten sowie einige weitere Funktionen auch in einem Kontextmenü (erreichbar über die rechte Maustaste, Abbildung 3.18) ausgewählt werden.

Die zweite in Abbildung 3.17 zu sehende Taste TABELLE [Ctrl]-[F4] schaltet den Anzeigemodus auf die Darstellung von internen Tabellen um. Dieser Anzeigemodus ist neben dem Standardmodus der am häufigsten verwendete. In ande-

ren Anzeigemodi ist die Drucktaste FELDER (F12) enthalten, mit der zum Standardmodus zurückgekehrt werden kann.

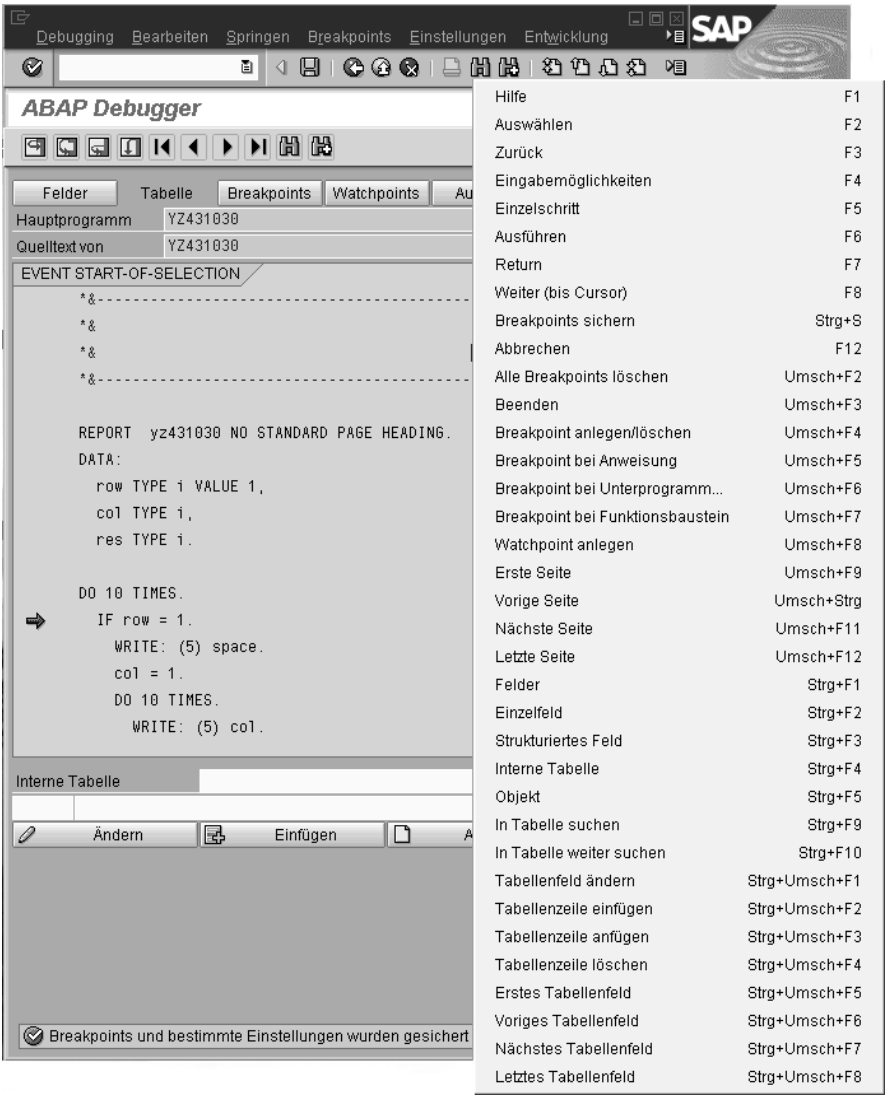

Abbildung 3.18 **© SAP AG**
Kontextmenü im Debugger

Zum Test einer Anwendung werden oft Testdaten aufgebaut, die durch die Anwendung verändert werden. Da jeder Bildwechsel einer Anwendung ein Commit Work und damit das Festschreiben von Datenbankänderungen auslöst, müssten die Testdaten nach jedem Lauf des Debuggers erneut aufgebaut werden. Um dies zu vermeiden, kann über die Menüfunktion DEBUGGING | DATENBANK | ROLLBACK ein Rollback der Datenbank ausgelöst werden.

Am äußeren rechten Ende der ersten Statuszeile ist eine weitere kleine, nur in Form eines Symbols dargestellte Drucktaste zu finden. Das Symbol stellt entweder ein grünes Plus-Zeichen oder ein rotes Minus-Zeichen dar. Diese Taste vergrößert den Anzeigebereich für den Quelltext auf die gesamte Dynpro-Fläche oder stellt den Normalzustand (Quelltext mit zusätzlichen Informationen, je nach Anzeigemodus) wieder her.

Insgesamt stehen etwa ein Dutzend Anzeigemodi zur Verfügung, von denen die sieben wichtigsten mittels der Drucktasten in der ersten Dynpro-Zeile aktiviert werden können. Alle Anzeigemodi sind natürlich auch über Menüfunktionen verfügbar. In den verschiedenen Modi stehen unter Umständen weitere Funktionen zur Verfügung. Nachfolgend sollen die wichtigsten Anzeigemodi mit eventuell vorhandenen Sonderfunktionen beschrieben werden.

Felder

Dieser Modus ist der bereits erwähnte Standardmodus. Er zeigt neben dem Quelltext vier Feldpaare zur Anzeige des Inhaltes von Datenfeldern (einfache Felder oder Feldleisten, also auch Kopfzeilen von Tabellen) an. Im jeweils linken Feld wird der Name des Datenfeldes eingetragen, im rechten Feld erscheint dann der aktuelle Inhalt des Feldes. Feldleisten werden als Feld vom Typ C interpretiert. Der Inhalt der Anzeigefelder wird in der PAI-Verarbeitung des Debugger-Dynpros aktualisiert. Nach dem manuellen Eintragen eines Feldnamens in der linken Spalte muss daher erst die Datenfreigabetaste betätigt werden, bevor der Feldinhalt erscheint. Es ist auch möglich, darzustellende Felder mit einem Doppelklick im Quelltext auszuwählen. Sie werden automatisch in das nächste freie Feld im Anzeigebereich eingetragen. Es ist möglich, mehr als vier Felder zu überwachen. In diesem Fall ermöglichen einige Symbole oberhalb der kleinen Tabelle das Rollen in der Liste der zu überwachenden Felder.

In diesem Modus können Feldinhalte nicht nur ausgewertet, sondern auch dauerhaft verändert werden. Dazu ist der neue Wert in der rechten Spalte einzutragen – dies ist schließlich auch ein Eingabefeld – und anschließend die äußere Drucktaste mit dem Stift-Symbol neben dem Eingabefeld zu drücken.

Tabellen

Dieser Modus ist der nach dem Standardmodus wohl am häufigsten verwendete. Er stellt den Inhalt einer internen Tabelle dar. In diesem Modus stehen für den Quelltextbereich nur wenige Zeilen zur Verfügung (siehe Abbildung 3.19). Er ist daher weniger geeignet, den Programmablauf zu verfolgen.

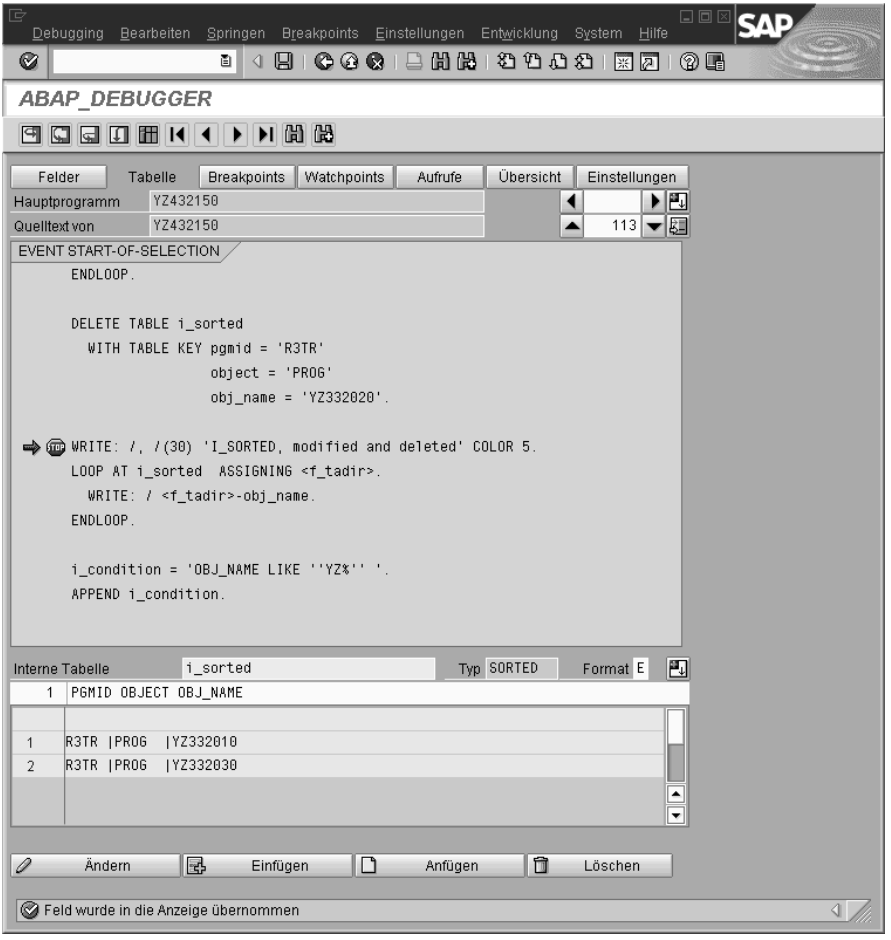

Abbildung 3.19 **© SAP AG**
Darstellung von internen Tabellen im Debugger

In der ersten Zeile unterhalb des Programmbereichs finden Sie zwei Eingabefelder. Eines dient zur Eingabe des Namens der darzustellenden internen Tabelle, das andere bestimmt den Darstellungsmodus (siehe Tabelle 3.2). Ein Ausgabefeld zwischen den beiden Eingabefeldern gibt Auskunft über den Typ der inter-

nen Tabelle. Ein Symbol rechts außen dient zum Vergrößern bzw. zum Verkleinern des Darstellungsbereichs für den Tabelleninhalt. Es wird als grünes Plus- bzw. als rotes Minus-Zeichen dargestellt.

Darstellungsart	Funktion
C	Tabellenzeile als unstrukturiertes Char-Feld
E	Tabellen mit Aufbereitung der Spalten
X	Hexadezimale Darstellung

Tabelle 3.2
Darstellungsarten für interne Tabellen im Debugger

Die zweite Zeile besteht ebenfalls aus zwei Eingabefeldern. Das erste erlaubt die Eingabe einer Datensatznummer, zu der gesprungen wird. Das zweite Feld dieser Zeile nimmt die Spaltenüberschriften auf. Durch Überschreiben der Spaltennamen können Sie die Reihenfolge der angezeigten Spalten verändern.

In der dritten Zeile beginnt die Darstellung des Tabelleninhalts. Die erste der aufgelisteten Zeilen ist die Kopfzeile der internen Tabelle. Sie wird durch ein Symbol markiert. Die übrigen, mit einer Zeilennummer versehenen Zeilen stellen den Inhalt der internen Tabelle dar.

Mittels weiterer vier Drucktasten unter dem Tabellenbereich können Sie neue Spalten hinzufügen, vorhandene löschen oder einzelne Feldwerte ändern.

Ähnlich wie bei einzelnen Feldern gibt es auch in der Tabellenanzeige alternative Möglichkeiten zur Übernahme des Tabellennamens. Wenn der Anzeigemodus für Tabellen bereits aktiv ist, reicht ein Doppelklick auf den Tabellennamen im Programmbereich. Andernfalls setzen Sie den Cursor auf den Tabellennamen und schalten durch Betätigen der Schaltfläche TABELLE in diesen Modus um.

Einzelfeld

In diesem Modus liefert der Debugger Informationen zu einem Feld bzw. einer Feldleiste. Es werden für elementare Felder der Datentyp, die definierte und die Ausgabelänge sowie die Zahl der Dezimalstellen dargestellt. Bei Feldleisten erzeugt der Debugger eine Tabelle mit den Inhalten aller Teilfelder. Interessant ist dies beispielsweise für die Struktur `SYST`. In diesem Fall werden die Inhalte aller Systemfelder angezeigt (Abbildung 3.20).

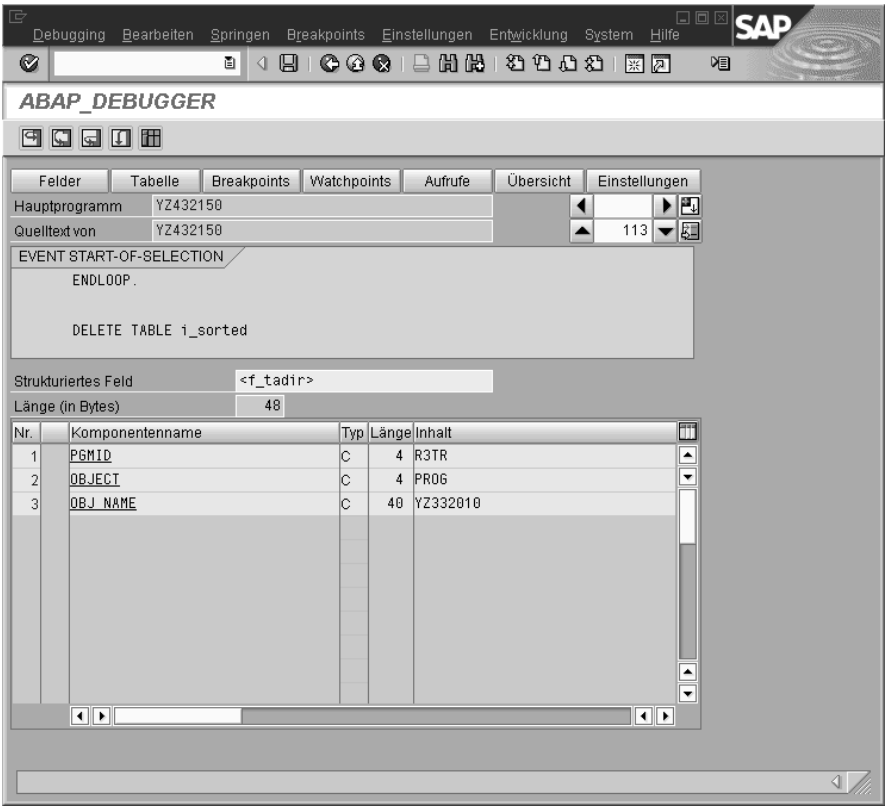

Abbildung 3.20 **© SAP AG**
Darstellung von Feldleisten im Modus Einzelfeld

Übersicht

In diesem Modus zeigt der Debugger die Struktur des aktuellen Programms mit Modulen, Ereignissen und Unterprogrammen an.

Aufrufe

Dieser Modus stellt die Aufrufreihenfolge der diversen Unterprogramme und Funktionsbausteine sowie der Ereignisse dar.

Einstellungen

Dieses Bild liefert keine Informationen zum aktuellen Programm, sondern ermöglicht die Modifikation der Einstellungen des Debuggers.

Watchpoints

Dieser Modus erlaubt die Definition von Watchpoints. Diese dienen dazu, Feldinhalte permanent mit einem vorgegebenen Muster zu vergleichen und bei Erfüllung einer vorgegebenen Bedingung den Programmablauf zu unterbrechen. Angelegt werden Watchpoints durch eine Schaltfläche in der Drucktastenzeile oder ein Symbol innerhalb des Watchpoint-Bereichs (siehe Abbildung 3.21).

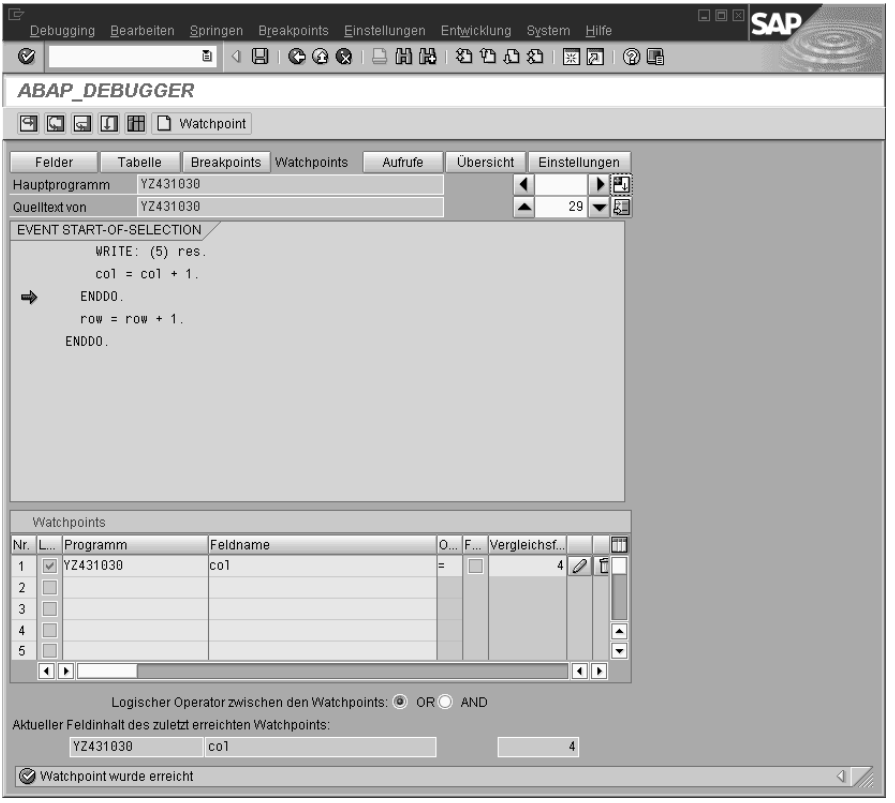

Abbildung 3.21 **© SAP AG**
Darstellung von Watchpoints

Die für einen Watchpoint erforderlichen Angaben sind in einem kleinen Popup (Abbildung 3.22) zu erfassen. Der Programmname ist bereits vom System vorgegeben. Im Feld darunter tragen Sie den Namen des zu überwachenden Feldes ein. Danach legen Sie den Vergleichsoperator fest. Abschließend tragen Sie im Eingabefeld VERGLEICHSFELD/-WERT das Vergleichsmuster ein. Dabei kann es sich um einen Direktwert oder einen Feldinhalt handeln. Im letztgenannten Fall müssen Sie außerdem das Flag VERGLEICHSFELD markieren.

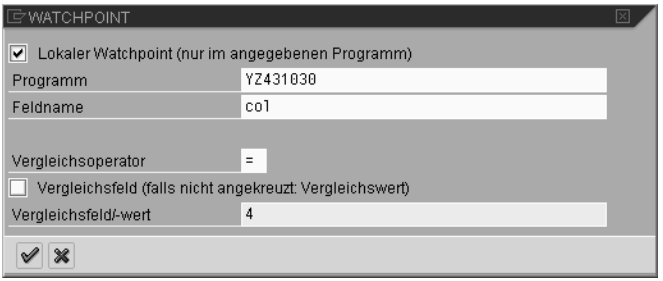

Abbildung 3.22 **© SAP AG**
Anlegen eines Watchpoint

3.1.5 Bezeichner und Namensräume

Bis zur Version 3.x unterliegen die Namen von Elementen der Entwicklungsumgebung Beschränkungen bezüglich der Länge und teilweise des Aufbaus der Namen. Diese Beschränkungen verhinderten beispielsweise die Vergabe von sprechenden Namen für Anwendungen. Außerdem mussten SAP-Kunden bei Eigenentwicklungen die Einhaltung des Kundennamensraumes berücksichtigen, um Datenverlust nach SAP-Upgrades zu verhindern.

Ein neues Konzept zur Vergabe von Bezeichnern soll einen Großteil der Probleme beseitigen. Kern dieses Konzeptes sind längere Bezeichner für die meisten Elemente der Entwicklungsumgebung sowie die komfortablere Vergabe von Namensräumen.

Angaben zur maximalen Länge der Bezeichner finden Sie in Tabelle 3.3.

Element	Länge in Zeichen
DDIC-Tabelle	30
Datenelement	30
Domäne	30
Programm	40
Funktionsbaustein	30
Funktionsgruppe	26
Entwicklungsklasse	30
Logische Datenbank	20
Get/Set-Parameter	20
Modulnamen	30

Element	Länge in Zeichen
Unterprogrammnamen	30
Suchhilfen	30
Bereichsmenüs	20
Transaktion	20

Tabelle 3.3
Maximale Länge von Bezeichnern ab Release 4.0

Die Namen können einen in Schrägstriche „/" eingeschlossenen Präfix erhalten, zum Beispiel `/MATZKE/TESTPROG1`. Diese Präfixe stellen die Bezeichnung eines Namensraumes dar, der von der SAP AG verwaltet und an Drittentwickler vergeben werden kann. Auf diese Weise sind Namenskonflikte in Kundensystemen und Probleme beim Upgrade vermeidbar.

3.2 Grundlegende Kommandos

In diesem Abschnitt lernen Sie Kommandos kennen, die nahezu unabhängig vom Programmtyp (Report oder Dialoganwendung) eingesetzt werden können. Zur Demonstration der Kommandos dienen einfache Reports. Der Einsatz von Reports erfolgt, weil sich diese Programme, wie bereits gezeigt, mit wenig Aufwand erzeugen und abarbeiten lassen.

3.2.1 Textsymbole und Überschriften

Die in diesem Unterabschnitt beschriebenen Sprachelemente sind keine ABAP-Kommandos im engeren Sinne, sondern reservierte Bezeichner, die über spezielle Transaktionen gepflegt und in vielen Anwendungen benutzt werden. Dies zeigt, dass die Programmierung im R/3-System mehr Wissen erfordert als nur die Kenntnis der diversen Kommandos.

Beachten Sie bitte, dass die nachfolgend beschriebenen Elemente ebenso wie der eigentliche Quelltext nach Änderungen aktiviert werden müssen, damit die neue Version wirksam werden kann.

Textsymbole

Standardsoftware muss über Sprachgrenzen hinweg verwendbar sein. Dies bedeutet unter anderem, dass sich Ausschriften, Fehlernachrichten, Feldbezeichnungen in Dynpros usw. leicht in andere Sprachen übersetzen lassen müssen. Es ist daher unüblich, Zeichenketten direkt im Programm zu notieren. Vielmehr

stellt ABAP so genannte Textsymbole zur Verfügung, die einem Programm zugeordnet werden. Die Textsymbole erhalten einen dreistelligen Bezeichner. Das System identifiziert Textsymbole anhand dieses Bezeichners und der Anmeldesprache. Die Sprache wird beim Anmelden durch den Benutzter gewählt und ab diesem Zeitpunkt in einer Systemvariablen aufbewahrt. Die Übersetzung der Textsymbole in verschiedene Zielsprachen kann völlig unabhängig von der eigentlichen Programmentwicklung erfolgen. In früheren Versionen der R/3-Software wurden Textsymbole als nummerierte Textelemente bezeichnet, da als Identifikatoren nur Ziffern zugelassen waren.

Im Programm werden Textsymbole durch das Schlüsselwort `TEXT-` in Verbindung mit einem dreistelligen Bezeichner definiert. Die Pflege der Textsymbole (und einiger anderer übersetzungsrelevanter Elemente) erfolgt in einer eigenen Transaktion `SE32`. Diese kann vom Bereichsmenü der Entwicklungsumgebung durch die Menüfunktion ENTWICKLUNG | PROGRAMMIERUMFELD | TEXTELEMENTE aufgerufen werden. Bei dieser Form des Aufrufs muss im Startbild der Transaktion (Abbildung 3.23) der Name des Programms bzw. der Klasse, für das Textelemente gepflegt werden sollen, angegeben werden. Mit der Drucktaste ABGLEICHEN erzeugen Sie einen Arbeitsvorrat, den Sie anschließend über die Drucktaste ÄNDERN bearbeiten können. Innerhalb des Programmeditors erreichen Sie dieses Werkzeug über den Menüeintrag SPRINGEN | TEXTELEMENTE | TEXTSYMBOLE.

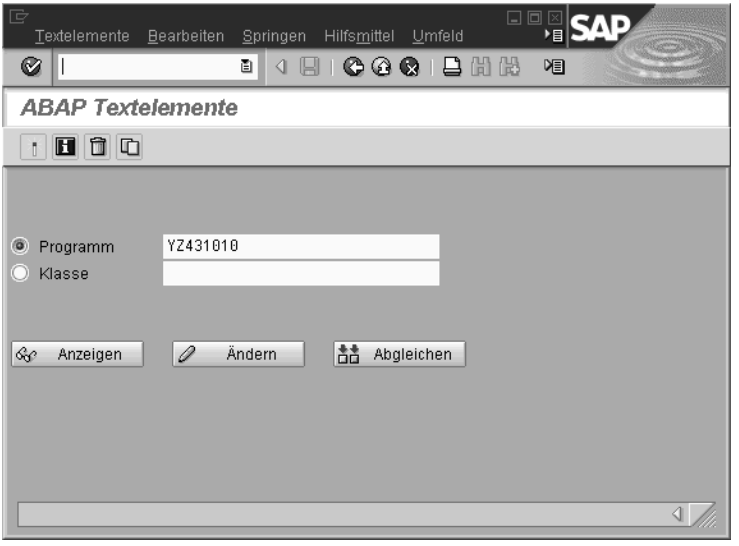

Abbildung 3.23 **© SAP AG**
Startbild der Transaktion SE32 zur Pflege der Textsymbole

Während der Programmentwicklung ist es für gewöhnlich recht lästig, den Editor zu verlassen, eine andere Transaktion aufzurufen, um die Textelemente zu pflegen, und dann wieder zum Editor zurückzukehren. Dies ist auch nicht notwendig. Die Entwicklungsumgebung verfügt über ein leistungsfähiges Navigationskonzept, mit dem Sie viele Elemente direkt vom Editor aus pflegen können. Der Navigationsmechanismus wird im Quelltext durch einen Doppelklick auf die diversen Begriffe ausgelöst. Dieselbe Wirkung erzielt das Positionieren des Cursors auf ein Element mit anschließendem Druck auf die Funktionstaste F2. Die Reaktion des Editors hängt vom selektierten Element ab. Beim Klick auf existierende Elemente wie Datenfelder, Namen von Unterprogrammen usw. wird zu der Stelle im Quelltext gesprungen, an der dieses Element definiert wurde oder zu dem Werkzeug, mit dem dieses Element gepflegt werden kann. Falls ein selektiertes Element noch nicht existiert, bietet das System Werkzeuge an, um dieses Element zu erzeugen. Der Navigationsmechanismus funktioniert auch über Programmgrenzen hinweg!

Die praktische Demonstration erfordert zunächst, dass Sie folgendes Programm erzeugen:

```
REPORT yz432010.
SET TITLEBAR 'T01'.
WRITE / TEXT-001.
WRITE / TEXT-002.
WRITE / TEXT-ABC.
```

Sichern Sie das Programm, und führen Sie dann mit der Maus einen Doppelklick auf der Zeichenkette `TEXT-001` aus. Da dieses Textsymbol noch nicht existiert, müssen Sie zunächst in einem Popup bestätigen, dass Sie dieses Textsymbol wirklich anlegen möchten. Die Transaktion `SE32` zur Pflege der Textelemente (siehe Abbildung 3.24) wird aufgerufen. Dabei wird das Startbild übersprungen, Sie gelangen sofort zur Pflege der Textsymbole. Dort können Sie einen Text für dieses Element eintragen. Zuvor müssen Sie in der Spalte MLEN die maximale Länge für den Text eintragen. Dabei ist zu berücksichtigen, dass bei einer eventuellen Übersetzung längere Texte entstehen können. Danach speichern Sie die Eingabe durch Ctrl-S oder das Sichern-Symbol und kehren mit F3 oder dem Zurück-Symbol (grüner Pfeil) zum Editor zurück. Führen Sie diesen Schritt dann auch für die anderen beiden Texte aus, und überprüfen Sie das Ergebnis durch Ausführen des Programms. Die Texte können Sie beliebig wählen.

Auch Textsymbole müssen aktiviert werden. Dazu stehen in der Pflegetransaktion ein Symbol in der Drucktastenzeile bzw. die Menüfunktion TEXTELEMENTE | AKTIVIEREN zur Verfügung. Falls Sie die Textsymbole nicht aktivieren, sind die Änderungen nicht wirksam.

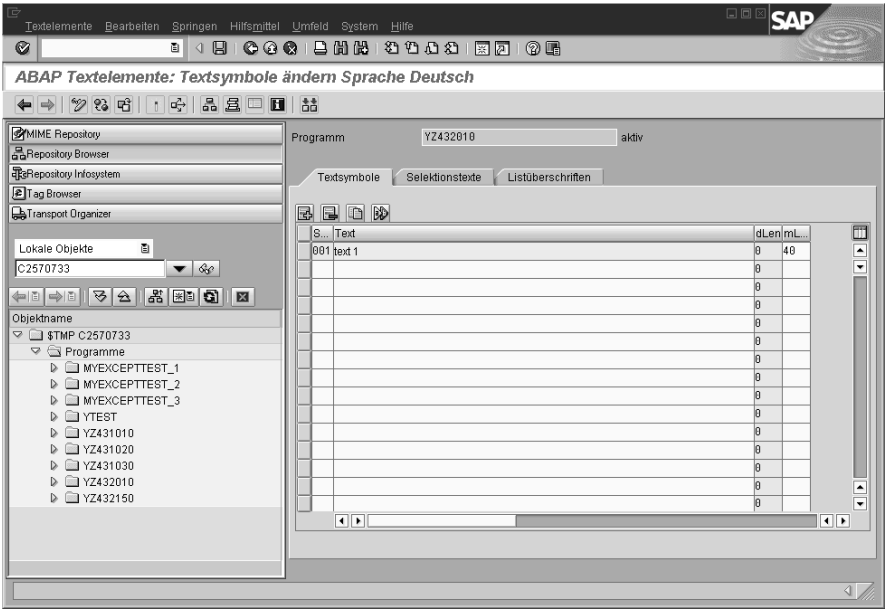

Abbildung 3.24 **© SAP AG**
Pflege der Textsymbole

Allerdings kann die Aktivierung auch außerhalb der eigentlichen Pflegetransaktion erfolgen. Die Entwicklungsumgebung vermerkt alle inaktiven Objekte bzw. Teilobjekte in einer Liste. Wenn innerhalb des Programmeditors die Funktion zum aktivieren eines Programms aufgerufen wird und mehrere inaktive Objekte existieren, dann werden diese in einem Popup aufgelistet. Dort kann die Auswahl der tatsächlich zu aktivierenden Elemente erfolgen (vergleiche Bild 3.12).

Textsymbole werden immer in der Anmeldesprache gesucht. Steht in dieser Sprache noch keine Übersetzung zur Verfügung, kann auch kein Text ausgegeben werden. Solche Fehler sollen nach Möglichkeit vermieden werden. Durch die Schreibweise:

```
string(identifier)
```

kann im Programm für ein Textsymbol ein Defaultwert vorgegeben werden. Dieser Defaultwert wird nur dann wirksam, wenn in der aktuellen Sprache das Textsymbol nicht zur Verfügung steht. Eine Anweisung in einem Programm könnte also lauten

```
WRITE / 'Date:'(017).
```

Im weiteren Verlauf dieser Einführung werden Sie keines dieser Textsymbole mehr finden. In den gedruckten Programmbeispielen ist es für die Verständlichkeit des Quelltextes besser, die jeweiligen Texte direkt aufzunehmen, auch wenn dies nicht den Konventionen für R/3-Anwendungen entspricht.

Überschriften

Neben den Textsymbolen, die sowohl in Reports als auch Dialoganwendungen eingesetzt werden können, existieren weitere Elemente, die nur für Reports sinnvoll sind. Es handelt sich dabei um die Überschriften für die gesamte Liste und die Spalten. Diese Elemente werden ebenfalls in der Transaktion SE32 gepflegt. Sie müssen dort lediglich den Reiter (Tab Strip) LISTÜBERSCHRIFTEN aktivieren oder beim Aufruf der Transaktion über einen Menüeintrag aus dem Editor heraus die entsprechende Auswahl treffen.

Die Listenüberschrift wird in der ersten Zeile der Liste ausgegeben, die Spaltenüberschrift erscheint, farblich hervorgehoben, darunter.

Selektionstexte

Für Reports existieren zwei Kommandos (SELECT-OPTIONS und PARAMETERS), mit denen Sie die Eingabe von Selektionswerten durch den Anwender ermöglichen können. Diese Kommandos, deren genaue Beschreibung Sie im Abschnitt 3.4.3 finden, erzeugen ein Dynpro mit Eingabefeldern. Die erläuternden Bezeichnungen für diese Felder pflegen Sie innerhalb der Textelementepflege unter der Rubrik SELEKTIONSTEXTE. Dort stehen aber nur dann eingabebereite Felder zur Verfügung, wenn in Ihrem Report die erwähnten Kommandos enthalten sind.

Titel

Zusätzlich zu den Texten, die innerhalb der Arbeitsfläche angeboten werden, ist es möglich, den Inhalt der Kopfzeile des Bildschirmfensters vom ABAP-Programm aus zu setzten. Dazu dient das Kommando

```
SET TITLEBAR title [ OF PROGRAM program ] [ WITH g1 g2 ... gn ].
```

Wenn der Titel nicht gesetzt wird, trägt das System den Kurztext aus den Programmattributen ein. Die Titel ähneln den nummerierten Texten, werden aber mit völlig anderen Werkzeugen bearbeitet. Der Titelidentifikator ist eine dreistellige Zeichenkette. Gepflegt werden Titel entweder über Funktionen der Workbench oder am einfachsten mittels des schon bei den Textelementen demonstrierten Doppelklick-Mechanismus. Eine syntaktisch korrekte SET TITLEBAR-Anweisung wird im Quelltext eingetragen, z. B.

```
SET TITLEBAR 'T01'.
```

Dann wird der Cursor auf die Zeile positioniert und die Funktionstaste F2 betätigt oder ein Doppelklick mit der Maus auf den Titelidentifikator durchgeführt. Das System stellt anschließend, gegebenenfalls nach einer Sicherheitsabfrage, ein Popup-Fenster (Abbildung 3.25) zur Pflege des Titels bereit. Diese Titel werden ebenso wie die nummerierten Texte übersetzt, stehen also in der jeweiligen Anmeldesprache bereit.

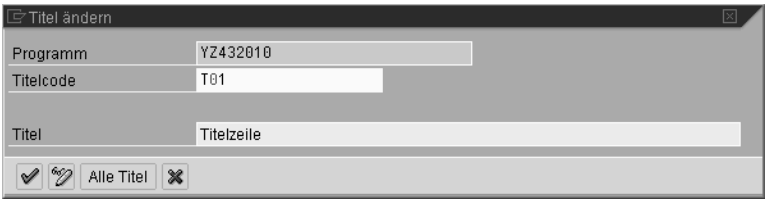

Abbildung 3.25 **© SAP AG**
Bearbeiten eines Titels

Die Titel müssen ebenso wie der Programmtext aktiviert werden, damit sie wirksam sind. Betätigen Sie dazu im Pflegepopup die Drucktaste ALLE TITEL. Das darauf folgende Dynpro listet alle Titel des Programms auf. Die Aktivierung erfolgt über die Menüfunktion OBERFLÄCHE | AKTIVIEREN oder ein Symbol in der Drucktastenzeile (Abbildung 3.26).

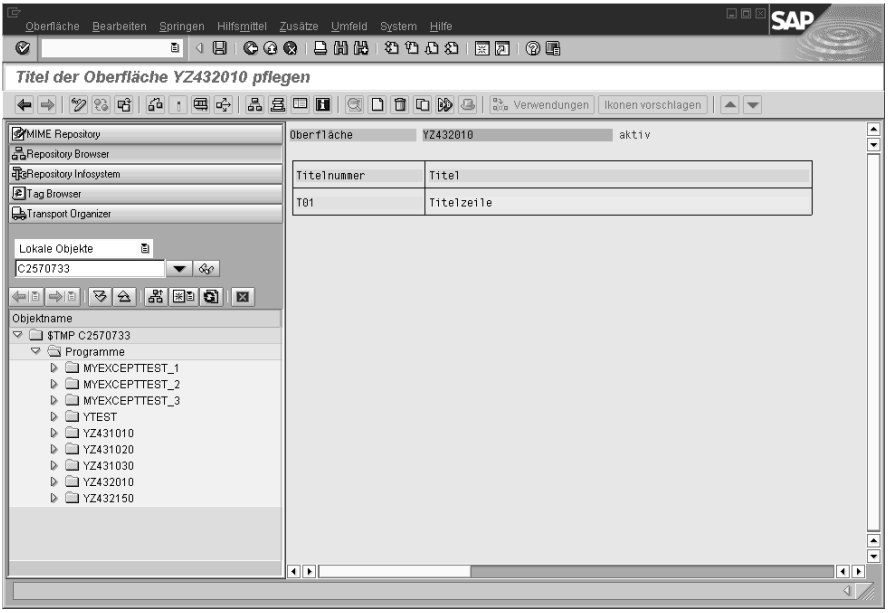

Abbildung 3.26 **© SAP AG**
Pflege aller Programmtitel

Die so erzeugten Titel haben nichts mit dem Titel der Textelementepflege zu tun. Sie stellen völlig eigenständige Elemente dar, die alternativ zum Standardtitel benutzt werden können. Der Titelidentifikator kann entweder – wie oben dargestellt – als Direktwert oder als Feldinhalt an die `SET TITLEBAR`-Anweisung übergeben werden.

Im Titel können die Platzhalter „&1“ bis „&9“ vorhanden sein. Diese Platzhalter werden durch die Option `WITH` mit Werten versorgt. Falls Sie den Titel nicht aus dem Textvorrat des aktuellen Programms sondern aus einem anderen Programm entnehmen möchten, so können Sie mit dem Zusatz `OF PROGRAM` den Namen des entsprechenden Programms festlegen. Notwendig oder zumindest wünschenswert ist dies z. B., wenn Sie in einem allgemein verwendbaren Funktionsbaustein ein Dynpro aufrufen und in diesem Dynpro den Titel des aktuellen Hauptprogramms setzen möchten.

Falls Ihre Anwendung über eine eigene Oberfläche verfügt, so können Sie die Titel auch innerhalb des Screen Painter pflegen.

Übungen

❶ Legen Sie das Programm `YZ432010` und die drei Textsymbole an.

❷ Pflegen Sie eine List- und eine Spaltenüberschrift zu diesem Programm.

❸ Pflegen und setzen Sie einen Programmtitel.

3.2.2 Datenobjekte und Datentypen

Die bisherigen Programme dienten zum ersten Einstieg in die Programmentwicklung. Die Fähigkeit, Texte auszugeben, reicht für eine ernstzunehmende Programmiersprache natürlich nicht aus. Erforderlich ist der flexible Umgang mit Daten der verschiedensten Typen und die Möglichkeit, auf diese Daten diverse Operationen anzuwenden. Wie andere Programmiersprachen verfügt daher auch ABAP über die Möglichkeit, Variablen zu verwenden. Unter ABAP wurde für Variablen oft der Begriff Datenfeld oder kurz Feld benutzt. Diese Bezeichnungen besitzen auch heute noch ihre Berechtigung. Da es aber verschiedene Arten von Variablen gibt und zwischenzeitlich ein ausgefeiltes Typkonzept existiert, ist es angebrachter, allgemein von Datentypen und Datenobjekten zu sprechen. Datenobjekte sind die Instanzen von Datentypen. Datentypen beschreiben somit die Eigenschaften von Datenobjekten. Zur Datenspeicherung können nur Datenobjekte benutzt werden.

Im SAP-System existieren drei Arten von Datenobjekten:

- Einzelfelder
- Feldleisten (strukturierte Datenfelder, Zeilentyp)
- Tabellen

Einzelfelder beruhen direkt oder indirekt auf den vordefinierten Datentypen des SAP-Systems. Sie können jeweils einen Wert aufnehmen. Der Begriff Feldleisten ist äquivalent zum Begriff Datensatz. Eine Feldleiste fasst mehrere andere Datenobjekte zu einem neuen, komplexen Objekt zusammen. Für Datentypen, mit denen Feldleisten definiert werden, kommt oft die Bezeichnung *Zeilentyp* zum Einsatz. Interne Tabellen fassen mehrere gleichartige Einzelfelder oder Feldleisten zu einer Tabelle zusammen, die aber nicht auf der Datenbank sondern nur im Hauptspeicher des Computers existiert. Datentypen zur Beschreibung von internen Tabellen werden auch als *Tabellentypen* bezeichnet. Darüber hinaus kann die DATA-Anweisung auch benutzt werden, um den Zugriff auf Datenbanktabellen zu ermöglichen. Diese Verwendungsmöglichkeit soll hier zunächst unberücksichtigt bleiben. Sie wird im Abschnitt 3.2.9 über die Verwendung von Datenbanktabellen näher erläutert.

In diesem Abschnitt finden Sie zunächst die Erläuterung der Begriffe Definition und Deklaration. Anschließend werden die vordefinierten Datentypen vorgestellt. Den Hauptteil bildet die Beschreibung der drei genannten Datenobjekt-Gruppen, wobei Definition und Deklaration im Zusammenhang behandelt werden.

Definition und Deklaration

Im Zusammenhang mit Datentypen und Datenobjekten sind zwei Anweisungen von Interesse. Mit dem Kommando

```
TYPES datatype typedefinition.
```

definieren Sie einen Datentyp. Ein Datentyp kann selbst keine Daten speichern, sondern er beschreibt lediglich die Eigenschaften eines Datenobjekts. Die Typdefinition kann sehr komplex werden, da vor allem Tabellentypen eine Vielzahl von Eigenschaften besitzen, die in der Definition festgelegt werden können.

Um ein Datenobjekt zu erzeugen, benutzen Sie das Kommando DATA. Durch die Deklaration wird Speicherplatz reserviert. Nur so erhalten Sie ein Datenobjekt, in dem Sie Daten speichern können. Die allgemeine Form dieses Kommandos lautet:

```
DATA dataobject TYPE datatype.
```

Eine etwas speziellere Form der DATA-Anweisung ermöglicht es, bei komplexen Datenobjekten auf die vorherige separate Definition eines Datentyps zu verzichten. Die Deklaration enthält sowohl den Namen des Datenobjekts als auch die Typdefinition (Strukturbeschreibung). Diese Form der Deklaration wurde vor Einführung der TYPES-Anweisung benutzt. Sie kann nicht für alle Datenobjekte eingesetzt werden. Die prinzipielle Syntax ist:

```
DATA dataobject complex_definition.
```

Obwohl diese Form der Deklaration weiterhin unterstützt wird, ist der Einsatz eines separaten Datentyps empfehlenswert. Dies erleichtert die Übersicht und ermöglicht eine saubere Typisierung der Parameter von Unterprogrammen und Funktionsbausteinen.

Neben der direkten Definition von Datentypen in einem Programm ist es möglich, so genannte Typgruppen (Type-Pools) zu erzeugen. Dabei handelt es sich um Quelltexte, die lediglich Typdefinitionen und Konstantendeklarationen enthalten dürfen. Sie werden mit der Anweisung

```
TYPE-POOLS type_group.
```

in Anwendungen eingebunden. Erstellt werden Type-Pools innerhalb der Dictionary-Pflege (Transaktion SE11).

Vordefinierte Datentypen

In ABAP existieren einige vordefinierte Datentypen. Tabelle 3.4 zeigt eine Übersicht. Ein Datentyp bestimmt zunächst, wie die im jeweiligen Feld gespeicherten Informationen interpretiert werden. Außerdem sind die Standardlänge eines Datenfeldes und dessen Initialwert vom Datentyp abhängig.

Datentyp	Standard-Länge	Mögliche Länge	Initialwert	Beschreibung
C	1	1 -65.535	Leerzeichen	Text (Zeichenkette)
N	1	1- 65.535	'0...0'	Nummerische Zeichenkette
T	6	6	'000000'	Zeitpunkt (HHMMSS)
D	8	8	'00000000'	Datum (JJJJMMTT)
F	8	8	'0.0'	Gleitpunktzahl
I	4	4	0	Ganze Zahl
P	8	1 – 16	0	Gepackte Zahl
X	1	1 – 65.535	X'00'	Hexadezimal
STRING	0	Beliebig	Null-String	Char-Feld variabler Länge
XSTRING	0	Beliebig	Null	Byte-Feld variabler Länge

Tabelle 3.4
Vordefinierte Datentypen

Seit Version 4.0 können auch Sie Dictionary-Elemente (Tabellen und Datenelemente) als Datentypen verwenden. Dabei dienen Datenelemente als einfacher Typ, Tabellen definieren einen Zeilentyp. Voraussetzung für die Eindeutigkeit der Definitionen ist, dass Dictionary-Tabellen und Datenelemente in einem gemeinsamen Namensraum liegen. Es darf ab Version 4.0 keine Datenelemente mehr geben, die den selben Namen wie eine Tabelle besitzen. Diese Forderung bestand bis zur Version 3.x nicht.

Zeichenkette (Typ C)

Zeichenketten können beliebige Zeichen enthalten. Zeichenketten nehmen allgemeine Informationen wie Namen, Bezeichnungen usw. auf. Jedes Zeichen wird in einer Stelle des Datenfelds gespeichert.

Nummerische Zeichenkette (Typ N)

Eine nummerische Zeichenkette ähnelt der eben beschriebenen Zeichenkette. Allerdings kann sie nur Ziffern enthalten. Andere Zeichen werden bei Zuweisungen unterdrückt. Der Wert wird rechtsbündig eingetragen; leere Stellen werden mit '0' aufgefüllt. Nummerische Zeichenketten werden üblicherweise als Schlüsselwerte, z. B. Auftrags-, Material- oder Personalnummern etc., benutzt.

Zeitpunkt (Typ T)

Dieser Datentyp nimmt einen Zeitpunkt in der Form Stunde, Minute und Sekunde (HHMMSS) auf, wobei für jeden Wert zwei Stellen zur Verfügung stehen. Auf Trennzeichen zwischen den Bestandteilen wird verzichtet. Ein Feld vom Typ T besitzt somit eine feste Länge von 6 Zeichen. Sowohl bei der Zuweisung als auch bei der Ausgabe mit `WRITE` wird diese Darstellungsform ohne Trennzeichen verwendet. Andere Formatierungen bei der Ausgabe erfordern spezielle Optionen für das `WRITE`-Kommando. Beim Einsatz dieses Datentyps ist unbedingt eine Eigenheit zu beachten. Bei Bereichsüberschreitungen über den Betrag von 24 Stunden hinaus entsteht kein Laufzeitfehler oder eine anderweitige Warnung; vielmehr beginnt die Zählung wieder von vorn.

Datum (Typ D)

Ein Datumsfeld enthält das Datum in der Form Jahr, Monat, Tag (JJJJMMTT). Eine Trennung der Teilfelder entfällt. Mittels spezieller Formatierungsanweisungen, z. B. beim Kommando `WRITE`, kann ein Datum auf unterschiedliche Weise dargestellt werden. Die Darstellung mit `WRITE` ohne zusätzliche Angaben führt zu einer Umordnung der Teilwerte entsprechend der im Benutzerprofil eingestellten, meist länderabhängigen Datumsvariante. Datumsfelder besitzen stets die Länge 8, um alle Informationen aufnehmen zu können. Längere Datumsfelder wären unnötig, kürzere könnten keinen korrekten Wert aufnehmen.

Gleitpunktzahl (Typ F)

Eine Gleitpunktzahl wird in einer binären, nicht direkt darstellbaren Form gespeichert. Der Wert wird dabei in Exponent und Mantisse zerlegt, wobei natürlich das duale Zahlensystem zugrunde liegt. Dieser Datentyp entspricht damit in etwa dem Datentyp `float` aus C. Die Länge von acht Stellen hat indirekt Einfluss auf die mögliche Stellenzahl und Genauigkeit des Wertes. Unter ABAP reicht der darstellbare Wertebereich von -1E307 bis +1E307. Die Auflösung (kleinster darstellbarer Absolutwert) ist 1E-307. Die Genauigkeit beträgt etwa 15 Stellen.

Bei der Ausgabe von Gleitpunktzahlen wird standardmäßig die Exponentialdarstellung verwendet. Die Zuweisung kann allerdings in unterschiedlicher Form erfolgen, wobei der Wert aber immer in Apostrophe einzuschließen ist. Das Komma bei gebrochenen Zahlen wird, wie auch in anderen Programmiersprachen üblich, durch den Punkt dargestellt. Mögliche Werte sind z. B.:

```
'1'
'1.2'
'1.23E2'
'-123E-5'
```

Gleitpunktzahlen werden für Berechnungen verwendet, für die ein großer Wertebereich erforderlich ist und bei denen die beschränkte Genauigkeit in Kauf genommen werden kann.

Ganze Zahl (Typ I)

Ganze Zahlen werden ebenfalls in binärer Form gespeichert, die Länge des Feldes entspricht also nicht direkt der Stellenzahl des Wertes. Dieser Datentyp entspricht etwa dem unter C bekannten `longint`. Auf der üblicherweise benutzten Hardware lässt sich ein Wertebereich von -2**31 bis +2**31 darstellen (-2.147.483.648 bis +2.147.483.647). Der Datentyp I ist der bevorzugte Datentyp für Zählvariablen, Indizes usw.

Gepackte Zahl (Typ P)

Dieser Datentyp dient zur Speicherung von Festpunktzahlen. Durch den optionalen Zusatz `DECIMALS n` in der `DATA`-Anweisung wird die Zahl der Nachkommastellen für das jeweilige Datenfeld angegeben. Der Standardwert für `DECIMALS` ist 0! In gepackten Feldern werden jeweils zwei Ziffern des Wertes in einer Stelle (in einem Byte) abgespeichert, wobei zusätzlich eine Ziffer berücksichtigt werden muss, welche die Zahl der Nachkommastellen aufnimmt. Ein Datenfeld mit dem Typ P und der Länge L kann also, unabhängig vom Vorzeichen, Werte mit L * 2 - 1 Stellen aufnehmen. Dabei müssen die Nachkommastellen und ein eventuelles Vorzeichen mitgezählt werden! Ebenso wie bei Gleitpunktzahlen steht der Punkt als Trennzeichen zwischen ganzem Teil und Nachkommastellen des Wertes zur Verfügung. Bei Zuweisungen darf die Exponentialschreibweise nicht benutzt werden.

Der effektive Umgang mit gepackten Zahlen erfordert es, in den Attributen des Programms das Flag FESTPUNKTARITHMETIK zu markieren. Dann wird die Stellenzahl sowohl bei Berechnungen als auch bei der Ausgabe automatisch berücksichtigt. Üblicherweise ist dieses Flag beim Erzeugen eines neuen Programms standardmäßig gesetzt (vgl. Abbildung 3.7).

Gepackte Zahlen werden dort eingesetzt, wo Werte mit konstanter Zahl von Nachkommastellen auftreten, z.B. Preise, Mengen usw. Derartige Felder beziehen sich daher oft auf eine Währung oder eine Mengeneinheit. Die `WRITE`-Anweisung weist daher Zusätze speziell für Felder mit dem Typ P auf, mit deren Hilfe der Inhalt eines solchen Feldes entsprechend der Währung oder der Maßeinheit aufbereitet werden kann.

Hexadezimale Felder (Typ X)

Ähnlich wie bei gepackten Feldern nimmt eine Stelle im Datenelement zwei Hexadezimalzeichen auf. Die Zuweisung erfordert die Kennzeichnung des Hexadezimalwertes mit einem X., z.B. `X'0D0A'`. Derartige Felder sind üblicherweise nur in Sonderfällen von Bedeutung.

Zeichenketten mit variabler Länge (Typ STRING)

Dieser Datentyp ähnelt Zeichenketten vom Typ C. Allerdings verfügt er über eine variable Länge, diese muss bei der Deklaration nicht angegeben werden. Die reale Länge wird zur Laufzeit der Anwendung dynamisch bestimmt, wodurch sich im Vergleich zu Zeichenketten fest definierter Länge eine Speicherplatzersparnis ergibt.

Strings können momentan nicht in Dynpros oder Dictionary-Elementen benutzt werden.

Hexadezimale Felder variabler Länge (Typ XSTRING)

Der Datentyp `XSTRING` nimmt hexadezimale Werte auf. Ähnlich wie beim Datentyp `STRING` wird die benötigte Länge dynamisch bestimmt, so dass bei der Definition keine Längenzuweisung erfolgt.

Namen von Datenobjekten

Die Namen von Datenobjekten können bis zu 30 Zeichen lang sein. Bei zusammengesetzten Namen (z.B. Felder von Feldleisten) gilt dies für den gesamten Namen und nicht nur für eine Komponente. Außer den Sonderzeichen ( ) + . , und : können alle Zeichen, also auch Ziffern, benutzt werden. Der Name darf aber nicht nur aus Ziffern bestehen. Der Bindestrich "-" sollte allerdings nicht verwendet werden, obwohl er zur Gruppe der zulässigen Zeichen gehört. Er dient in so genannten Feldleisten zur Trennung der einzelnen Bestandteile eines Namens. Die zusätzliche Verwendung des Bindestrichs in Namen von Datenobjekten, die ebenfalls Bestandteile der Feldleisten sein können, würde zu unüber-

sichtlichen Namen führen. Sollte eine optische Trennung einzelner Begriffe im Namen eines Feldes notwendig sein, können Sie dazu den Unterstrich _ benutzen. Beispiele für zulässige Feldnamen sind folgende:

```
ANZAHL
M1
M123
BESTELL_NUMMER
LAGERMENGE_ALT
LAGERMENGE_NEU
```

Das System definiert einige Felder selbst. Die Namen dieser Felder dürfen ebenfalls nicht als Feldnamen verwendet werden. Dies betrifft insbesondere den Feldnamen `SPACE` und die eigentlichen Systemvariablen, die alle mit `SY-` oder `SYST-` beginnen.

Verschiedene ABAP-Anweisungen besitzen diverse Zusätze und Optionen. Die Namen dieser Zusätze können zwar als Feldnamen dienen, was aber in speziellen Fällen zu unübersichtlichen Anweisungen führt. Zu diesen möglichen Namen gehören beispielsweise die Bezeichner `TO`, `INTO` und `FROM`.

Datentypen und Datenobjekte werden vom System unabhängig voneinander verwaltet. Dies bedeutet, dass es jeweils einen Typ und ein Datenobjekt mit identischem Namen geben darf. Daraus können Missverständnisse resultieren, da sowohl Datentypen als auch Datenobjekte benutzt werden können, um in Deklarationen andere Datenobjekte zu beschreiben. Sie sollten daher stets für eindeutige Namen sorgen. Das kann z. B. durch Voranstellen eines Präfix („T_“) vor den Typnamen erfolgen.

In komplexen Anwendungen ist es ohnehin ratsam, durch systematischen Einsatz von Namenspräfixen die diversen Typen und Objekte zu kennzeichnen. Eine Richtlinie kann die Tabelle 3.5 sein. Sie zeigt empfehlenswerte Präfixe für die Namen von Datenobjekten. Diese Präfixe beschreiben die Art des Datenobjekts bzw. dessen Gültigkeitsbereich.

Präfix	Bedeutung
C	Konstante
G	Globales Datenobjekt (im Top Include deklariert)
L	Lokales Datenobjekt (innerhalb von Unterprogrammen oder Funktionsbausteinen)
P	Parameter
T	Datentyp

Tabelle 3.5
Art und Gültigkeitsbereich von Datenobjekten

Bei Bedarf können die Präfixe durch einen zweiten Buchstaben aus Tabelle 3.6 ergänzt werden. Dieser zweite Buchstabe beschreibt den Aufbau des Datenobjekts. Unter Umständen kann der entsprechende Buchstabe auch allein stehen.

Präfix	Bedeutung
F	Feldsymbol
I	Interne Tabelle
O	Objektreferenz
R	Datensatz (Feldleiste, Record)

Tabelle 3.6
Präfix zur Kennzeichnung der Struktur eines Datenobjekts

Schließlich können Parameter von Unterprogrammen oder Funktionsbausteinen näher spezifiziert werden. Diese Angaben in Tabelle 3.7 sind als Ergänzung zum Präfix P vorgesehen.

Präfix	Bedeutung
C	Changing-Parameter
E	Export-Parameter
I	Interne Tabelle
V	Value-Parameter

Tabelle 3.7
Zusätzliche Werte für Präfix P (Parameter von Funktionsbausteinen oder Unterprogrammen)

Nachfolgend (Tabelle 3.8) einige Beispiele für den Aufbau der Bezeichner.

Bezeichner	Bedeutung
ti_tadir	Typbezeichner für interne Tabelle (Tabellentyp)
tr_e070	Typbezeichner für Datensatz (Zeilentyp)
t_line	Typbezeichner für elementares Feld
c_yes	Konstante
i_tadir	interne Tabelle
g_fcode	globales Feld

Bezeichner	Bedeutung
lr_e071	lokale Feldleiste
p_line	Parameter
pv_line	Wertparameter
pi_tadir	Interne Tabelle als Parameter

Tabelle 3.8
Beispiele zur Namensbildung

3.2.3 Einfache Datenfelder

Einfache Datenfelder nehmen jeweils einen einzelnen Wert eines bestimmten Datentyps auf. Die Deklaration eines einfachen Datenfeldes ist durch vier Methoden möglich. Zunächst kann ein Feld direkt von einem der vordefinierten Datentypen abgeleitet werden. Dabei ist bei einigen Datentypen die Angabe einer Länge möglich:

```
DATA field[(length)] [ TYPE predefined_type].
```

Sofern bei der Deklaration eines einfachen Datenfelds nichts anderes angegeben wird, erzeugt ABAP ein Datenfeld vom Typ C mit der Länge 1. Die einfachste Variante der DATA-Anweisung lautet demzufolge

```
DATA field.
```

Längenangaben, sofern für den jeweiligen Datentyp möglich (nur für C, I, N, P und X), werden in runde Klammern eingeschlossen und stehen ohne Lücke direkt hinter dem Feldnamen.

```
DATA name(10).
```

Als zweite und dritte Variante kann der Bezug auf eine Typdefinition oder ein Dictionary-Datenelement erfolgen. Dabei darf keine Längenangabe benutzt werden, da die Länge bereits in der Typdefinition oder dem Datenelement vorgegeben wird.

```
DATA field TYPE datatype.
```

```
DATA field TYPE dataelement.
```

Als letzte Variante können Sie ein Feld deklarieren, dessen Eigenschaften einem bereits vorhandenen programminternen oder Dictionary-Feld (Tabellenfeld) entsprechen. Dazu verwenden Sie die Anweisung

```
DATA field LIKE field_2.
```

Diese Variante stellt sicher, dass ein Datenfeld immer exakt einem anderen Feld (meist einem Tabellenfeld) entspricht. Änderungen der Datenstruktur im Dictionary erfordern so keine Änderung im Programm. Sofern das Vergleichsfeld ein Tabellenfeld ist und nur in der Deklaration benutzt wird, muss die entsprechende Tabelle vor ihrer Verwendung in der Felddeklaration nicht im Programm deklariert worden sein. Der `LIKE`-Zusatz kann innerhalb der ABAP-Objects-Erweiterung allerdings nicht im Zusammenhang mit Dictionary-Datentypen benutzt werden. Bei allen Varianten der Deklaration kann durch den Zusatz `VALUE` ein Startwert gesetzt werden. Der Wert ist dabei in einfache Anführungsstriche einzuschließen:

```
DATA ... VALUE '<value>'.
```

Ohne eine derartige Angabe wird das Feld mit dem datentypabhängigen Initialwert belegt.

Das Kommando `DATA` unterstützt die Doppelpunktnotation zur Übergabe mehrerer Parameter:

```
DATA: field_1 [TYPE | LIKE ...], ...  field_n [TYPE | LIKE
...].
```

Die möglichen Typdefinitionen für einfache Datentypen ähneln den Deklarationsanweisungen. Eine vollständige Typdefinition ist zunächst möglich durch Bezug auf einen der vordefinierten Datentypen und optional die Angabe einer Länge:

```
TYPES datatype[(length)] [ TYPE predefined_type].
```

Falls der Bezug auf den Datentyp fehlt, wird wie bei der Deklaration automatisch der Datentyp C benutzt.

Falls ein Datentyp erzeugt werden soll, der den Eigenschaften eines existierenden Datenfelds entspricht, können Sie auch in der `TYPES`-Anweisung mit `LIKE` einen Bezug auf real existierende Felder herstellen:

```
TYPES datatype LIKE field.
```

Auch die beiden anderen Varianten, die Definition über einen anderen Datentyp und über ein Datenelement, können Sie verwenden, was aber wenig Sinn macht, da die beiden genannten Objekte selbst schon einen Datentyp darstellen.

```
TYPES datatype TYPE datatype.
```

```
TYPES datatype TYPE dataelement.
```

Beispiele für Definitionen und Deklarationen:

```
TYPES:
  t_name(30) TYPE C,
  t_kunnr    TYPE kunnr
  t_konto    LIKE kna1-konto.

DATA:
  is_changed,
  name(10),
  matnr(8) TYPE  N,
  c_ja     VALUE 'J',
  ok_code  LIKE sy-tcode,
  fname    TYPE t_name 'IXOS'.
```

Konstanten

Konstanten werden ähnlich wie die oben erwähnten Felder mit der Anweisung

```
CONSTANTS constant [TYPE | LIKE ...] VALUE [ value | IS
INITIAL ].
```

angelegt. Für `CONSTANTS` gelten dieselben Konventionen für den Namen wie bei `DATA`, außerdem können mit `LIKE` oder `TYPE` bzw. der expliziten Längenangabe mit `()` die relevanten Eigenschaften der Konstanten (Typ, Länge usw.) bestimmt werden. Auch diese Anweisung kann per Doppelpunkt mehrere Konstanten gleichzeitig deklarieren.

Für Konstanten muss natürlich bei der Deklaration zwingend ein Wert vorgegeben werden. Der Zusatz `VALUE` gehört somit zwangsweise zur `CONSTANTS`-Anweisung und ist nicht optional. Falls eine Konstante mit dem Initialwert des jeweiligen Datentyps angelegt werden soll, kann statt einem expliziten Wert der Zusatz `IS INITIAL` benutzt werden.

Beispiele:

```
CONSTANTS:
  c_yes VALUE 'X',
  c_no VALUE IS_INITIAL.
```

Datentypanhängige Besonderheiten bei der Ausgabe mit WRITE

Die verschiedenen vordefinierten Datentypen und somit auch die von ihnen abgeleiteten einfachen Felder werden bei der Ausgabe durch das Kommando `WRITE` unterschiedlich behandelt. Diese Eigenheiten müssen bei der Formatierung einer Liste berücksichtigt werden. Die zu beachtenden Unterschiede beziehen sich auf die voreingestellte Feldbreite bei Ausgaben und die Ausrichtung des Wertes innerhalb des Ausgabefeldes. Tabelle 3.9 zeigt die charakteristischen Eigenschaften.

Typ	Standardausgabelänge	Ausrichtung
C	Feldlänge	Linksbündig
D	8	Linksbündig
F	22	Rechtsbündig
I	11	Rechtsbündig
N	Feldlänge	Linksbündig
P	2*Feldlänge bzw. 2* Feldlänge +1	Rechtsbündig
T	6	Linksbündig
X	2* Feldlänge	Linksbündig

Tabelle 3.9
Standardformatierung bei Ausgabe mit WRITE

In vielen Fällen ist die Ausgabelänge von untergeordneter Bedeutung, da die Standardwerte meist brauchbare Ergebnisse erzeugen. Die Ausgabelänge wird vor allem in drei Fällen gesetzt:

- Zeichenkettenfelder können mitunter recht lang werden, wenn es sich um Bezeichnungen oder verbale Beschreibungen handelt. Diese Felder werden sehr häufig gekürzt.
- Bei Datums- und Zeitfeldern wird bei der Ausgabe üblicherweise immer eine größere Ausgabelänge (10 bzw. 8) gesetzt, damit auch die Trennzeichen korrekt ausgegeben werden. In den Datums- und Zeitfeldern mit Standardausgabelänge haben diese Zeichen keinen Platz, so dass nur die acht bzw. sechs Ziffern ausgegeben würden. Beachten Sie in diesem Zusammenhang bitte das Demoprogramm zur Typumwandlung und die Aufgaben am Ende dieses Abschnitts.
- Bei Gleitpunktzahlen wird sehr oft eine Längenbeschränkung in Verbindung mit einer zusätzlichen Formatierung der Ausgabe durchgeführt, um überflüssige Nachkommastellen abzuschneiden.

Zuweisungen

Wenn ein Datenfeld einen Wert erhalten soll, kann dies nur durch eine Zuweisung geschehen. Eine Möglichkeit der Zuweisung, die Verwendung der Klausel `VALUE` bei der Definition, wurde bereits erwähnt. Daneben gibt es noch weitere Möglichkeiten. An dieser Stelle sollen zunächst nur die Zuweisungen erwähnt

werden, die einzelne Datenfelder mit Werten füllen. Im Zusammenhang mit Feldleisten und Tabellen existieren noch weitere Varianten der Zuweisung.

Als universellster und sinnfälligster Zuweisungsoperator wird das Gleichheitszeichen eingesetzt. So notierte Zuweisungen erinnern sehr an Zuweisungen anderer Programmiersprachen. Die allgemeine syntaktische Beschreibung einer Zuweisung lautet:

```
field = expression.
```

wobei `expression` ein Direktwert, ein Feld oder ein komplexer Ausdruck sein kann. Dem angegebenen Feld wird der Wert des Ausdrucks zugewiesen. Dabei finden üblicherweise implizite Typumwandlungen statt, falls die beteiligten Felder unterschiedliche Datentypen besitzen. Selbstverständlich wird die Länge des Zielfeldes berücksichtigt; es werden maximal so viele Stellen übertragen, wie das Zielfeld aufnehmen kann. Bei Zuweisungen mit dem Gleichheitszeichen sind Mehrfachzuweisungen der Form

```
field_1 = field_2 = expression.
```

möglich. Diese werden dann von rechts nach links abgearbeitet. Nur ganz rechts darf anstelle eines Feldes auch ein Ausdruck stehen. Eine funktionell zum Gleichheitszeichen identische Zuweisung mit der Anweisung `MOVE` erinnert an die Assemblerprogrammierung, die z.B. innerhalb des mainframebasierten SAP-Systems R2 eine große Rolle spielt.

```
MOVE field_1 TO field_2.
```

Mit dieser Anweisung wird einem Feld der Wert eines anderen Feldes oder ein im Programm fest kodierter Wert zugewiesen. Die Verwendung eines Ausdrucks anstelle von `field_1` ist nicht möglich. Bei Zuweisungen zwischen Feldern unterschiedlicher Typen finden automatisch Typkonvertierungen statt. Interessant an dieser Variante ist, dass mittels des Doppelpunkts Mehrfachzuweisungen eingeleitet werden können:

```
MOVE: a TO b,
      c TO d,
      e TO f.
```

Als letzte Möglichkeit besitzt die `WRITE`-Anweisung eine spezielle Klausel, mit der die Ausgabe eines Feldes in ein anderes Feld umgeleitet werden kann. Dabei muss das Zielfeld vom Typ C sein.

```
WRITE field_1 TO field_2.
```

Bei dieser Form der Zuweisung findet eine Ausgabeaufbereitung statt, die der bei der Ausgabe der Werte in eine Liste entspricht. Diese Aufbereitung ist nicht mit der impliziten Typkonvertierung bei `MOVE` identisch. Dazu ein kleines Beispiel:

```
REPORT yz432020.
DATA:
  d TYPE D,
  s(10).
d = sy-datum.
MOVE d TO s.
WRITE: / s.
WRITE d TO s.
WRITE: / s.
```

Diese Form der WRITE-Anweisung ermöglicht es außerdem, im Gegensatz zu MOVE, den Namen des Quellfeldes indirekt anzugeben. Er steht dann nicht direkt in der Anweisung, sondern in einem dritten Feld, das in der WRITE-Anweisung in runde Klammern zu setzen ist.

```
WRITE (field_1) TO field_2.
```

Auch dazu ein kleines Beispiel:

```
REPORT yz432030.
DATA:
  name(10),
  s1(10) VALUE 'EINS',
  s2(10).

name = 'S1'.
WRITE  (name) TO s2.
WRITE / s2.
```

Mit der Anweisung CLEAR, die über den Doppelpunktmechanismus mehrere Parameter verarbeiten kann, werden Felder auf ihren Initialwert zurückgesetzt, z.B.:

```
CLEAR: name, s1, s2.
```

Offset- und Längenangaben

Bereits bei der einführenden Beschreibung der Anweisung WRITE wurde demonstriert, dass die Ausgabelänge von Feldern begrenzt werden kann. Diese Möglichkeit besteht nicht nur bei der Ausgabe, sondern auch bei anderen Feldverwendungen, allerdings nicht für alle Datentypen. Felder mit den Datentypen P, F und I sind von dieser Möglichkeit ausgeschlossen, da bedingt durch ihre interne Darstellung nur der Zugriff auf den Wert als Ganzes sinnvoll ist.

Die Angabe eines nummerischen Wertes in runden Klammern direkt hinter dem Feldnamen beschränkt den Zugriff auf die angegebene Stellenzahl. Zusätzlich kann ein Offset angegeben werden. Der Zugriff erfolgt dabei nicht ab der ersten Stelle des jeweiligen Feldes, sondern um die angegebene Stellenzahl versetzt. Ein Offset von 0 spricht also die erste Stelle des Datenfelds an, ein Offset von 1

die zweite Stelle usw. Interpretiert man den Offset als Index, so beginnt dieser innerhalb des Datenfelds mit 0.

Der Offset wird ebenfalls hinter dem Feldnamen, aber vor der Längenangabe eingetragen. Gekennzeichnet wird der Offset durch ein vorangestelltes Plus-Zeichen +. Zwischen den Bestandteilen dürfen keine Leerzeichen stehen! Die Syntax lautet demzufolge:

field+*offset*(*length*)

Die erwähnten Angaben wirken sowohl beim Lesen von Datenfeldern als auch bei Zuweisungen. Anstelle konstanter Werte kann zumindest innerhalb von Zuweisungen sowohl der Offset als auch die Länge dynamisch, also als Inhalt eines Feldes angegeben werden. An anderen Stellen, z.B. bei der Ausgabe mit `WRITE`, existiert die Möglichkeit der dynamischen Offset- und Längenangabe in dieser einfachen Form nicht. Zur Erläuterung der Definition, der Zuweisungen und der Offset- und Längenangaben soll folgendes Programmbeispiel dienen:

```
REPORT  YZ432040.
DATA: s1(4),
      s2(20) VALUE 'In aqua ',
      s3(20) VALUE 'vino veritas.',
      three TYPE I VALUE 3,
      four  TYPE I VALUE 4.

* Bei Zuweisungen wird die Länge der Datenfelder
* berücksichtigt, zu lange Werte werden abgeschnitten
s1 = s3.
WRITE / s1.
WRITE /.

* Positionsangaben können sowohl bei der Quelle
* als auch beim Ziel stehen
s2+8 = s3+5.
WRITE / s2.
WRITE /.

* mit einer Längenangabe kann der zu ersetzende Bereich
* genau bestimmt werden,
s2+three(four) = s1.
WRITE / s2.
WRITE /.

* denn Zuweisungen ohne Längenbeschränkung überschreiben
* ab der Startposition bis zum Ende
s2+3 = s1.
WRITE / s2.
```

Operationen und Funktionen

Bezüglich der Operationen unterscheidet ABAP solche, die durch Operationszeichen ausgeführt werden, und solche, für die spezielle, so genannte operationale Anweisungen zur Verfügung stehen. Letztere Art von Operationen ähneln den vordefinierten Funktionen, so wie sie aus anderen Programmiersprachen bekannt sind.

Bereits seit Release 3.0 stehen eine Reihe von Anweisungen bereit, welche die Arbeit mit Zeichenketten erleichtern (siehe Tabelle 3.10).

Anweisung	Funktion
CHARLEN	Liefert die Länge des ersten Zeichens eines zeichenartigen Feldes.
CONCATENATE	Verknüpfen mehrerer Zeichenketten.
CONDENSE	Eliminieren von Leerzeichen.
DBMAXLEN	Liefert für STRING und XSTRING die Dictionary definierte Maximallänge des Strings zurück.
NUMOFCHAR	Liefert die Anzahl der Zeichen eines zeichenartigen Feldes.
REPLACE	Zeichenweises Ersetzen.
SHIFT	Zeichenweises Verschieben.
SPLIT	Zerlegen von Zeichenketten.
STRLEN	Liefert die Anzahl der Bytes, die für die Zeichenkette benötigt wird.
TRANSLATE	Änderung der Schreibweise und Austausch von Zeichen.
XSTRLEN	Liefert die Länge eines byteartigen Feldes (Typ X oder XSTRING).

Tabelle 3.10
Anweisungen zur Stringverarbeitung

Einige der Anweisungen verhalten sich unterschiedlich, je nach dem, ob das zu Grunde liegende R/3-System Unicode-fähig ist und welcher Zeichensatz in nicht Unicode-fähigen Systemen benutzt wird. Dies hat z.B. Auswirkung auf die Funktionen STRLEN und NUMOFCHAR. Während NUMOFCHAR immer die Zahl der Zeichen liefert, kann STRLEN unter bestimmten Umständen einen Wert zurückgeben, der Auskunft über die Anzahl der Bytes liefert, die für die Speicherung der Zeichenkette benötigt werden. Manipulationen von Zeichenketten müssen diesen Unterschied berücksichtigen.

Das folgende Programm zeigt ein Beispiel für die Arbeit mit einigen Zeichenketten-Funktionen.

```
REPORT yz432050.
DATA: sc(30),
      ss TYPE string,
      sx TYPE xstring,
      l TYPE i.

sc = 'abcäöü'.
ss = 'xyz321'.
sx = 'abcyyz'.

l = strlen( sc ).
WRITE: / 'strlen( sc ) = ', l.
l = strlen( ss ).
WRITE: / 'strlen( ss ) = ', l.
l = numofchar( ss ).
WRITE: / 'strlen( ss ) = ', l.
l = xstrlen( sx ).
WRITE: / 'xstrlen( sx ) = ', l.

l = dbmaxlen( ss ).
WRITE: / 'dmbaxlen( sx ) = ', l.

l = charlen( sc ).
WRITE: / 'charlen( sc ) = ', l.

CONCATENATE sc ':' ss INTO sc.
WRITE: / 'concatenate sc ss = ', sc.

TRANSLATE ss USING 'xUyVzW'.
WRITE: / 'translate ss = ', ss.

REPLACE ALL OCCURRENCES OF 'ä' IN sc WITH 'ae'.
WRITE: / 'sc nach replace = ', sc.

SPLIT sc AT ':' INTO sc ss.
WRITE: / 'sc nach split = ', sc.
WRITE: / 'ss nach split = ', ss.

sc+8(1) = 'X'.
WRITE: / 'sc nach einfügen = ', sc.
```

Neben der Verwendung der diversen Zeichenkettenfunktionen demonstriert dieses Beispiel noch eine andere wichtige Eigenschaft der Char-Datenfelder. Obwohl das Feld `sc` nur mit sechs Zeichen gefüllt ist, kann mit der Zuweisung `sc+8` die 9. Stelle des Feldes angesprochen werden. In manch anderen Programmiersprachen würde der Versuch, eine Position außerhalb der aktuellen Länge des Strings anzusprechen, zu einem Laufzeitfehler führen. Unter ABAP hingegen

sind alle Zuweisungen erlaubt, sofern sie innerhalb der definierten Feldlänge liegen, unabhängig vom aktuellen Inhalt des Feldes. Auch bei Ausgaben wird standardmäßig die definierte Feldlänge verwendet, und nicht die Länge des aktuellen Wertes.

Einige weitere Funktionen erfordern eine detailliertere Beschreibung. Die Funktion `TRANSLATE` übersetzt einzelne Zeichen in ein jeweils anderes. Im Parameterstring werden paarweise das zu ersetzende und das Ersetzungszeichen angegeben, wobei mehrere solcher Paare aufeinander folgen können.

Ebenfalls zum Ersetzen von Zeichen durch andere dient die Funktion `REPLACE`. Sie ersetzt aber nicht ausschließlich einzelne Zeichen. Mit ihr können ganze Zeichenketten durch andere ersetzt werden.

Die Funktion `SPLIT` teilt eine Zeichenkette unter Verwendung einer Trennzeichenkette in mehrere Teile. Die trennende Zeichenkette ist in den entstehenden Substrings nicht mehr enthalten.

Das Verbinden von Zeichenketten ist mit der Funktion `CONCATENATE` möglich.

Alle Zeichenkettenfunktionen können durch diverse Kommando-Zusätze in ihrer Funktion beeinflusst werden. Diese Zusätze sind aber oft von der konkreten R/3-Version abhängig, so dass Sie hier die Systemdokumentation zu Rate ziehen sollten.

Arithmetische Operationen werden vorzugsweise mit Operationszeichen dargestellt. Es stehen die Zeichen `+`, `-`, `*` und `/` für die Grundrechenarten zur Verfügung. Mit `**` kann eine Potenzierung ausgeführt werden. Mit `DIV` ist eine ganzzahlige Division möglich. Der Operator `MOD` hingegen liefert den Rest einer Division.

Die Abarbeitungsreihenfolge in komplexen Ausdrücken kann durch Klammern bestimmt werden. Allgemein gilt, dass zunächst Funktionen, dann die Potenzierung, anschließend Multiplikation und Division (einschließlich `DIV` und `MOD`) und zum Schluss Addition und Subtraktion ausgeführt werden. Wichtig ist, dass Operationszeichen und Operatoren stets durch Leerzeichen voneinander getrennt werden müssen. Nachfolgend einige Beispiele für Ausdrücke:

```
a = 1 + 3.
c = 8 / ( 3 + 1 ).
b = a - c
d = c+3. "Dangerous! No numeric expression!
```

Des weiteren unterstützt ABAP die in Tabelle 3.11 aufgeführten mathematischen Funktionen.

Name	Bedeutung	Zugelassene Datentypen
EXP	Exponentialfunktion	F
LOG	Natürlicher Logarithmus	F
SIN	Sinusfunktion	F
COS	Cosinusfunktion	F
SQRT	Quadratwurzel	F
DIV	Ganzzahlige Division	F
MOD	Rest	F
ACOS	Arcus-Cosinus	F
ASIN	Arcus-Sinus	F
TAN	Tangens	F
COSH	Hyperbelcosinus	F
SINH	Hyperbelsinus	F
TANH	Hyperbeltangens	F
LOG10	Logarithmus zur Basis 10	F
ABS	Absolutwert	F, I, P
SIGN	Vorzeichen (-1, 0, 1)	F, I, P
CEIL	Kleinster ganzzahliger Wert >= x	F, I, P
FLOOR	Größter ganzzahliger Wert <= x	F, I, P
TRUNC	Ganzzahliger Teil	F, I, P
FRAC	Dezimalteil	F, I, P

Tabelle 3.11
Mathematische Funktionen

Die Zuweisungen mit dem =-Zeichen stellen eine vereinfachte Sonderform von ABAP-Zuweisungen dar, da alle Anweisungen üblicherweise mit einem Schlüsselwort beginnen. Für die beschriebenen Zuweisungen existieren daher auch Langformen, die durch ein Schlüsselwort eingeleitet werden. So lautet die vollständige Syntax für Zuweisungen eigentlich

```
COMPUTE field = expression.
```

Dabei ist das Schlüsselwort `COMPUTE` optional und wird daher in der Praxis kaum verwendet. Alternativ zu den Operationszeichen existieren die operationalen Anweisungen `ADD TO`, `SUBTRACT FROM`, `MULTIPLY BY` und `DIVIDE BY`. Nachfolgend ein Beispiel zur Anwendung dieser Anweisungen.

```
ADD value TO sum.
```

Nach den erwähnten Schlüsselwörtern muss eine Konstante oder ein Feld stehen; ein Ausdruck wird nicht verarbeitet und erzeugt eine Fehlermeldung beim Syntaxcheck. Für die erwähnten Anweisungen existieren noch einige Sonderformen, deren Anwendung nur in Spezialfällen, z.B. im Zusammenhang mit Feldleisten, interessant ist.

Typumwandlungen

Da es mittlerweile 10 vordefinierte Datentypen gibt, bestehen bei Zuweisungen 100 Möglichkeiten zur Kombination der verschiedenen Typen und damit theoretisch 100 verschiedene Regeln zur Typkonvertierung. ABAP unterstützt fast alle denkbaren Konvertierungen automatisch. An dieser Stelle sollen nicht alle Regeln erläutert werden. Sie sind in der Online-Hilfe des Systems ausführlich beschrieben. Wichtig sind vor allem folgende Grundregeln:

- Das Quellfeld muss einen Wert besitzen, der in einen Wert vom Typ des Zielfeldes umgewandelt werden kann. Hat beispielsweise das Quellfeld den Typ C und das Zielfeld den Typ F, so muss das Quellfeld die Darstellung einer Gleitpunktzahl enthalten.
- Die Reaktionen auf eine nicht ausreichende Stellenzahl im Zielfeld sind unterschiedlich. Sie reichen vom einfachen Abschneiden über das Löschen des Inhalts (Füllen des Zielfeldes mit dem Zeichen *) bis hin zum Laufzeitfehler.
- Datums- und Zeitangaben werden unter Umständen in Zeitabstände (in Tagen oder Sekunden) ab einem bestimmten Bezugszeitpunkt umgewandelt. Auch die Umwandlung in entgegengesetzter Richtung (Zeitraum zu Datum) ist möglich. Dies ermöglicht Datumsrechnungen der Form:

  ```
  date = date + integer_value.
  ```

Das folgende Beispiel zeigt die Auswirkungen einiger Typumwandlungen. Es kann als Grundlage für eigene Experimente dienen. Es sei hier angemerkt, dass in ABAP-Anwendungen überwiegend die Datentypen C, N, D und P benutzt werden, diesbezügliche Typumwandlungen also von besonderem Interesse sind.

```
REPORT yz432060.
DATA: s(20),
      n TYPE I,
      d TYPE D,
      x TYPE F.
```

```
s = '123'.
n = s.
WRITE: / '1.   CHAR -> INT  ', 25(10) s, (20) n.

n = s + 1.
WRITE: / '2.   CHAR -> INT  ', 25(10) s, (20) n.

n = s+1.
WRITE: / '3.   CHAR -> INT  ', 25(10) s, (20) n.
WRITE /.

s = '123.789'.
x = s.
WRITE: / '4.   CHAR -> FLOAT', 25(10) s,
         (20) x DECIMALS 5 EXPONENT 0.

n = x.
WRITE: / '5.   FLOAT-> INT  ',
         25(10) x DECIMALS 5 EXPONENT 0, (20)n.
WRITE /.
WRITE /.

s = '19900601'.
d = s.
WRITE: / '6.   CHAR -> DATUM', 25(10) s, (10) d.

s = '900601'.
d = s.
WRITE: / '7.   CHAR -> DATUM', 25(10) s, (10) d.
WRITE /.

d = '19991231'.
s = d.
WRITE: / '8.   DATUM -> CHAR', 25(10) d, (10) s.

n = d.
WRITE: / '9.   DATUM -> INT', 25(10) d, (10) n.
```

Dieses kurze Programm erzeugt die in Abbildung 3.27 dargestellte Ausgabe.

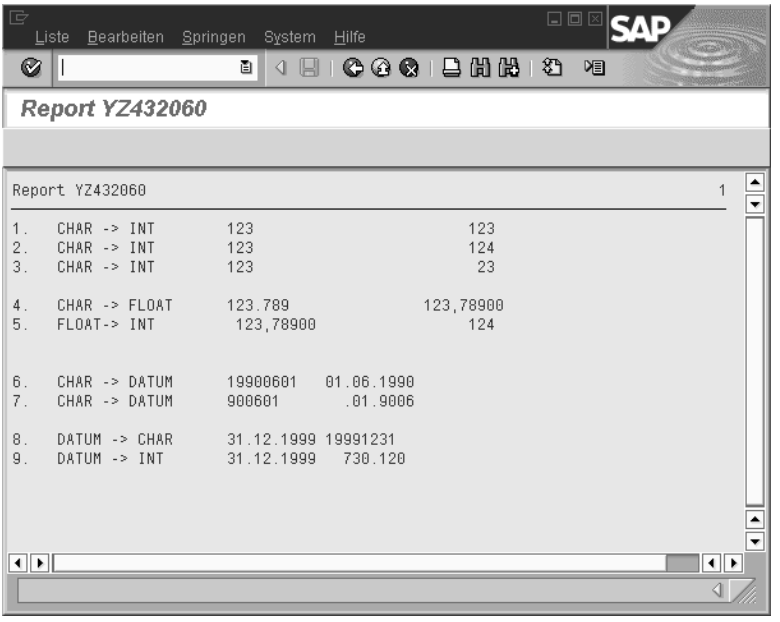

Abbildung 3.27 **© SAP AG**
Typumwandlungen

Die ersten drei Zeilen demonstrieren die Zuweisung von Zeichenketten zu nummerischen Feldern. Zunächst wird die unveränderte Zeichenkette einem Feld vom Typ I zugewiesen. Sie wird erwartungsgemäß korrekt in eine Zahl umgewandelt. Die nächste Zuweisung demonstriert die Verwendung einer Zeichenkette in einem nummerischen Ausdruck. Auch hier erfolgt eine korrekte Umwandlung in eine Zahl und deren Verwendung bei der Auswertung des Ausdrucks.

Die Verwendung von Zeichenketten in nummerischen Ausdrücken kann eine recht heimtückische Falle für den Programmierer sein. Die dritte Zuweisung ist syntaktisch korrekt, stellt aber trotz der bis auf die Leerzeichen zum vorangegangenen Ausdruck identischen Schreibweise keinen nummerischen Ausdruck dar. Hier wird der Inhalt der Zeichenkette ab Offset 1 (also ab der zweiten Stelle) für die Zuweisung zum nummerischen Feld benutzt. Dieses erhält also nicht den Wert 124, sondern lediglich 23. Dabei kommt den Leerzeichen eine erhebliche Bedeutung zu. Derartige logische Fehler werden durch keinen Syntaxcheck gefunden, mitunter sind derartige Zuweisungen sogar im Sinne des Programmierers. In solchen Fällen sollte nach Möglichkeit ein Kommentar auf die Korrektheit und den Zweck der Anweisung hinweisen.

In Zuweisung 4 wird eine gebrochene Zahl in Zeichenkettendarstellung einem Gleitpunktfeld zugewiesen. Bedeutsam sind hier die unterschiedlichen Trennzeichen zwischen Vor- und Nachkommastellen. Während die Ausgabe des Gleitpunktwerts mit `WRITE` ein echtes Komma verwendet, muss dieses in der Zeichenkettendarstellung des Wertes durch einen Punkt ersetzt werden. Die Schreibweise

```
s = '123,789'.
x = s.
```

führt zu einem Laufzeitfehler, da die Zeichenkette nicht als Zahl interpretiert werden kann. Bei der Zuweisung einer Gleitpunktzahl zu einem Integer-Feld werden Nachkommastellen gerundet.

Die Zuweisungen von Zeichenketten zu Datumsfeldern (6) erfordern eine spezielle Schreibweise. Diese ist auch einzuhalten, wenn die Zuweisung nicht über ein Zeichenkettenfeld, sondern eine fest kodierte Konstante erfolgt. Das Datum ist stets in der Form JJJJMMTT zu notieren. Auch hier unterscheiden sich die Formate bei der Zuweisung und der Ausgabe mit `WRITE`. Fehlen Zeichen bei der Zuweisung, so entstehen, wie in Bildschirmzeile 7 demonstriert, ungültige Datumswerte. Ähnliches gilt für Zeitangaben (Typ T).

Die letzten beiden Zuweisungen demonstrieren noch einmal die Umwandlung von Datumswerten in Zeichenketten oder Integer-Werte. Bei der Zuweisung zu einer Zeichenkette ergibt sich das bekannte JJJJMMTT-Format. Der Integer-Wert gibt hingegen die Zahl der vergangenen Tage seit Beginn der Zeitrechnung (01.01.0001) an.

Die implizite Typumwandlung in Ausdrücken und Zuweisungen birgt einige Fallen, da das Endergebnis von der Reihenfolge der Umwandlungen abhängen kann. Der folgende Ausschnitt aus einem Programm soll das verdeutlichen:

```
REPORT yz432070.
DATA d TYPE D.
d = '19991225' + 7.
WRITE / d.
d = '19991225'.
d = d + 7.
WRITE / d.
```

Wenn Sie dieses Programm ausführen, werden Sie zwei unterschiedliche Ergebnisse erhalten, obwohl die ausgeführten Operationen scheinbar identisch sind. Der Grund liegt darin, dass während der Berechnung der Ausdrücke die Zwischenergebnisse unterschiedliche Datentypen aufweisen. Im ersten Ausdruck wird zunächst eine nur aus Ziffern bestehende Zeichenkette mit einem numerischen Wert verknüpft. Innerhalb des Ausdrucks ist nicht bekannt, dass später die Zuweisung zu einem Datumsfeld erfolgen soll und die Zeichenkette daher ein Datum enthält. Die Zeichenkette wird daher zunächst in einen Integer-Wert

gewandelt und um den Wert sieben erhöht. Erst dann erfolgt die Umwandlung in ein Datum. Auf der rechten Seite der Zuweisung steht nun aber ein Integer-Wert und nicht eine Zeichenkette. Dem Datumsfeld wird daher ein Datum zugewiesen, dass 19.991.232 Tage nach dem 1.1.0001 liegt. Dabei entsteht ein ungültiger Wert, der zur Zuweisung von 0 zum Datumsfeld führt.

Im zweiten Teil des Programms wurde die Zuweisung und der Ausdruck auf zwei Zeilen aufgespalten. Die Zuweisung zum Datumsfeld führt nun zum korrekten Ergebnis, da das Laufzeitsystem eindeutig den Datentyp von Quellwert und Zielfeld bestimmen kann. Auch die nachfolgende Operation ist eindeutig hinsichtlich der Datentypen, daher ergibt sich das Ergebnis 1.1.2000.

Neben der impliziten Typumwandlung in Ausdrücken und Zuweisungen können Sie durch einige Varianten des Kommandos `ASSIGN` (Abschnitt 3.2.5) auch explizite Typumwandlungen durchführen.

Übungen

1. Testen Sie die bisher vorgestellten Anweisungen. Erzeugen Sie zu den unvollständig aufgeführten Beispielen eigene Rahmenprogramme.
2. Testen Sie die Typumwandlungen zwischen verschiedenen Datentypen. Analysieren Sie das Verhalten bei Überschreitungen des Wertebereichs.
3. Schreiben Sie ein Programm, das Felder aller vordefinierten Typen mit Werten belegt und diese ausgibt. Verändern Sie die Ausgabelänge auf Werte kleiner und größer der Standardlänge, und werten Sie das Ergebnis aus.
4. Üben Sie den Umgang mit gepackten Zahlen. Achten Sie dabei auf das Rundungsverhalten und die Rechengenauigkeit bei Zwischenergebnissen.
5. Testen Sie die Datumsrechnung, insbesondere die Addition und Subtraktion zwischen Datumsfeldern untereinander und zwischen Datums- und Integer-Feldern. Weisen Sie das Ergebnis sowohl Datums- als auch Integer-Feldern zu.
6. Testen Sie die Ausgabe von Datums- und Zeitfeldern mit und ohne expliziter Angabe einer Ausgabelänge, die für die Darstellung des Wertes mit Trennzeichen ausreicht.

3.2.4 Feldleisten

Relationale Datenbanksysteme fassen mehrere einzelne Felder zu einem Datensatz zusammen. Entsprechende Konstrukte, oft als *Record* oder *Struktur* bezeichnet, sind aus herkömmlichen Programmiersprachen nicht mehr wegzudenken. Selbstverständlich kennt auch ABAP vergleichbare Elemente. Sie werden hier *Feldleiste* genannt. Datentypen, die derartigen Feldleisten definieren, werden auch als *Zeilentypen* bezeichnet.

Erzeugt werden Feldleisten auf zwei verschiedene Arten. Zum einen wird für jede Datenbanktabelle, die in einem Programm benutzt wird, automatisch eine Feldleiste angelegt. Über diese Feldleiste findet dann der Datenaustausch zwischen ABAP-Anwendung und Datenbanksystem statt.

Die zweite Variante besteht in der expliziten Deklaration von Feldleisten oder Feldleistentypen durch die Kommandos `DATA` oder `TYPES`. Dabei stehen verschiedene Optionen zur Verfügung, um die Struktur der Feldleiste zu definieren. Zunächst können Sie mit `BEGIN OF` und `END OF` die Struktur direkt und unabhängig von anderen Elementen definieren:

```
DATA: BEGIN OF record,
  declarations
END OF record.
```

oder

```
TYPES: BEGIN OF record,
  declarations
END OF record.
```

Bei den Deklarationen zwischen `BEGIN OF` und `END OF` handelt es sich um alle mit `DATA` eingeleiteten Deklarationen, nicht nur um die, die bereits beschrieben wurden. Das bedeutet, dass eine Feldleiste selbst Feldleisten oder sogar interne Tabellen enthalten kann. Ein Beispiel für die Definition eines Feldleistentyps in einem Programm könnte folgendermaßen aussehen:

```
TYPES: BEGIN OF t_address,
  name(20),
  street(30),
  city(20),
  zipcode(6),
END OF t_address.
```

Wie bei der Grundform von `DATA` für einfache Felder kann auch in Feldleistendeklarationen und -definitionen mit `LIKE` Bezug auf ein bereits existierendes Element (Feld oder Feldleiste) oder mit `TYPE` auf einen Typ genommen werden. Dabei existieren keine Einschränkungen hinsichtlich der Struktur des eingefügten Elements. Demnach können Feldleisten ihrerseits wiederum Feldleisten enthalten:

```
DATA: BEGIN OF invoice,
  customer TYPE t_address,
  value TYPE P DECIMALS 2,
END OF invoice.
```

Innerhalb der `DATA`- und der `TYPES`-Anweisung ermöglichen zwei neue Typisierungsvarianten den Bezug auf interne Tabellen. So können Sie mit den Anweisungen

```
[DATA | TYPES] record TYPE LINE OF itabtype.
```

bzw.

```
[DATA | TYPES] record LIKE LINE OF itab.
```

eine Feldleiste erzeugen, deren Aufbau der Struktur der nach `LINE OF` angegebenen Tabellenzeile entspricht. Diese Variante ist unter anderem in Unterprogrammen hilfreich, um dort einen Arbeitsbereich für interne Tabellen ohne Kopfzeile zu erzeugen. Näheres zu internen Tabellen erfahren Sie in Abschnitt 3.2.7.

Eine ähnliche Form erlaubt es, Feldleisten oder entsprechende Typen zu erzeugen, deren Struktur dem Datensatz einer Dictionary-Tabelle entspricht:

```
[DATA | TYPES] record LIKE database_table.
```

Der Zusatz `LINE OF` ist hier nicht erforderlich.

Beim Zugriff auf Feldleisten ist zwischen dem Zugriff auf die Feldleiste als Ganzes und dem Zugriff auf einzelne Felder zu unterscheiden. Für die Benutzung eines Teilfeldes einer Feldleiste wird der Name aus dem Namen der Feldleiste und dem eigentlichen Feldnamen gebildet. Dabei werden beide durch einen Bindestrich getrennt. Sind in einer Feldleiste weitere Feldleisten enthalten, so besteht der Name aus mehr als zwei Teilen. Bezogen auf die vorangegangenen Deklarationen sind gültige Feldnamen z. B. die folgenden:

```
invoice-value
invoice-customer-city
```

Die zweite Zugriffsmöglichkeit ist etwas riskant, in bestimmten Fällen aber unersetzbar. Sie funktioniert allerdings nur in Systemen, die nicht mit Unicode-Zeichensätzen arbeiten, da in diesen Systemen der für ein Zeichen benötigte Speicherplatz variieren kann. Für die Probleme, die mit der nachfolgend beschriebenen Zugriffsmöglichkeit gelöst werden können, existieren inzwischen auch andere Programmiertechniken. Für Neuentwicklungen sollte diese Zugriffsvariante auf Feldleisten daher nicht mehr benutzt werden, allerdings ist ihre Kenntnis zum Verständnis älterer Anwendungen mitunter notwendig.

Eine Feldleiste ist in ABAP zunächst ein einzelnes Feld des Typs C, dessen Länge durch die enthaltenen Felder bestimmt wird. Dabei kann die Länge der Feldleiste größer sein als die Summe der Längen der Einzelfelder, da diese möglicherweise durch automatisch eingefügte Zeichen auf Wortgrenzen ausgerichtet werden. Auf eine Feldleiste kann daher ohne Rücksicht auf die innere Struktur mit Offset- und Längenangaben zugegriffen werden. Dabei ist indirekt auch der Zugriff auf Felder möglich, die sonst nicht per Offset- und Längenangabe bearbeitet werden können. Diese Eigenschaft macht es auch möglich, einer Feldleiste eine Zeichenkette zuzuweisen, die dann gemäß der Anzahl der Zeichen auf die Teilfelder verteilt wird. Dazu zunächst ein Beispiel:

```
REPORT yz432080.
DATA: BEGIN OF fl,
  c1(2) VALUE '##',
  c2(2) VALUE '##',
  n TYPE I,
END OF fl.

WRITE: / fl-c1, / fl-c2, / fl-n.
fl = 'ABCDEFGH'.
WRITE: / fl-c1, / fl-c2, / fl-n.
```

Mit dieser Anweisung werden alle drei Felder der Feldleiste `fl` mit neuen Werten gefüllt. Während die Zuweisung zu `c1` und `c2` durchaus sinnvoll sein kann (Aufspalten eines Feldes in Teilfelder), erhält auch `n` einen Wert, der von der Interpretation des zugewiesenen Bitmusters entsprechend dem Datentyp abhängt. Derartige Zuweisungen sind mit Sicherheit unerwünscht.

Die Benutzung einer Feldleiste als Ganzes ist auch bei Zuweisungen von Feldleiste zu Feldleiste oder von Feldleiste zu einfachem Feld möglich. Auch bei diesen Zuweisungen werden die einzelnen Zeichen ohne Typumwandlung von der Quelle zum Ziel übertragen und dort unter Umständen neu interpretiert. Einzige Einschränkung bei dieser Art der Feldleistenverwendung ist hier nur die Länge des Zielfeldes, die auf jeden Fall berücksichtigt wird. Zuweisungen zwischen Feldleisten bringen also nur dann ein korrektes Ergebnis, wenn beide Feldleisten dieselbe Struktur aufweisen. In ABAP-Anwendungen besteht aber sehr oft der Wunsch, zwischen Feldleisten unterschiedlicher Struktur die Werte mehrerer Felder mit identischen Namen zu übertragen. Im folgenden Programm sollen beispielsweise Daten einer Bestellung und die Bezeichnung eines Artikels in eine dritte Feldleiste kopiert werden. Damit nicht für jedes Teilfeld eine eigene Zuweisung geschrieben werden muss, führt die Anweisung

```
MOVE-CORRESPONDING source_record TO target_record.
```

die Zuweisung zwischen Feldern gleichen Namens zweier Feldleisten aus:

```
REPORT yz432090.
TYPES t_wert TYPE P DECIMALS 2.

DATA: BEGIN OF mat,
  matnr(10) TYPE N VALUE '1234567890',
  matbz(20) VALUE 'Schreibtischlampe',
  preis TYPE t_wert VALUE '123.45',
END OF mat.

DATA: BEGIN OF best,
  matnr LIKE mat-matnr,
  stck TYPE I VALUE '2',
  summe TYPE t_wert,
END OF best.
```

```
DATA: BEGIN OF rech,
  matnr LIKE mat-matnr,
  matbz LIKE mat-matbz,
  stck LIKE best-stck,
  preis LIKE mat-preis,
  summe LIKE best-summe,
END OF rech.

MOVE-CORRESPONDING best TO rech.
MOVE-CORRESPONDING mat TO rech.
rech-summe = rech-stck * rech-preis.

WRITE: / rech-matnr, rech-matbz, rech-preis,
         rech-stck, rech-summe.
```

Ältere Releasestände der R/3-Software (kleiner als 3.0) verfügen nicht über das Typkonzept. Auch der Bezug auf interne Tabellen oder Dictionary-Tabellen mit `LIKE` ist dort nicht möglich. Daher sind die Möglichkeiten, Feldleisten anzulegen, eingeschränkt. Insbesondere ist es nicht möglich, Unterstrukturen auf die oben beschriebene Art einzubinden. Dies muss vielmehr über die Anweisung

```
INCLUDE STRUCTURE structure.
```

erfolgen. Diese Anweisung soll hier der Vollständigkeit halber beschrieben werden, da sie in älteren Programmen möglicherweise noch existiert, obwohl sie inzwischen durch andere Anweisungen abgelöst werden könnte. Die `INCLUDE`-Anweisung ist eine eigenständige Anweisung, die durch einen Punkt abgeschlossen werden muss. Sie kann also nicht ohne weiteres in die Parameterliste der `DATA`-Anweisung eingefügt werden. Vielmehr ist für `BEGIN OF` und `END OF` eine eigene `DATA`-Anweisung nötig, wie das folgende Beispiel zeigt.

```
DATA: BEGIN OF invoice.
  INCLUDE STRUCTURE adr.
DATA: value TYPE P DECIMALS 2,
END OF invoice.
```

Dabei muss `adr` entweder eine Struktur aus dem Data Dictionary oder eine programminterne, mit `DATA` definierte Struktur (Feldleiste oder interne Tabelle) sein. Die `INCLUDE`-Anweisung fügt die Elemente der Struktur als einzelne Felder ein, löst die Struktur `adr` also auf. Dadurch können Feldleisten älterer Releasestände nicht mehrfach verschachtelt werden; es existiert nur eine Feldebene.

Die eben gezeigte Deklarationsanweisung ist syntaktisch korrekt, allerdings etwas unübersichtlich. Das liegt daran, dass die Feldleiste durch zwei scheinbar unabhängige `DATA`-Anweisungen erzeugt werden muss. Die für die Datenstruktur entscheidenden Schlüsselwörter `BEGIN` und `END` stehen mitten in der Anweisung und können vor allem in komplexeren Deklarationsanweisungen leicht

übersehen werden. Die Ursache für solche Konstruktionen liegt in der nachträglichen Erweiterung der Syntax von ABAP, wobei wegen der unbedingt erforderlichen Kompatibilität mit älteren Releaseständen kein kompletter Neuentwurf möglich ist. Ergänzungen der Sprache unter vollständiger Gewährleistung der Kompatibilität werden oft durch komplizierte und mitunter missverständliche Syntax erkauft. Etwas abmildern lassen sich die Folgen einer solch komplizierten Syntax durch eine saubere Schreibweise mit Einrückungen. Die folgende Anweisung ist, trotz einer weiteren (redundanten) DATA-Anweisung, völlig kompatibel zum vorangegangenen Beispiel. Entscheidend bei der Deklaration von Feldleisten sind nicht die DATA-Anweisungen, die in nahezu beliebiger Anzahl benutzt werden dürfen, sogar für jedes Teilfeld der Feldleiste einzeln, sondern nur die Schlüsselworte BEGIN OF und END OF.

```
DATA: BEGIN OF invoice.
  INCLUDE STRUCTURE adr.
  DATA value TYPE P DECIMALS 2.
DATA: END OF invoice.
```

Eine andere Variante der INCLUDE-Anweisung ermöglicht es, anstelle einer existierenden Feldleiste oder Tabelle die Typdefinition einer Struktur einzubinden und dabei ebenfalls aufzulösen. Diese Variante der Anweisung lautet

```
INCLUDE TYPE type.
```

Für ihre Anwendung gelten dieselben Anmerkungen wie für INCLUDE STRUCTURE.

Im Gegensatz zur bisherigen Aussage, dass durch INCLUDE die Struktur aufgelöst wird, besteht ab Version 4.6 die Möglichkeit, die eingefügten Elemente weiterhin als eigenständiges Objekt zu behandeln. Dabei treten einige syntaktische Feinheiten zu Tage, die einer genaueren Erläuterung bedürfen.

Mit dem Zusatz

```
AS name
```

können Sie der eingefügten Struktur einen Namen zu geben. Obwohl die Struktur eigentlich aufgelöst wird, können Sie durch Verwendung des so zugewiesenen Namens so auf die Elemente zugreifen, als wäre name eine eigenständige Substruktur. Darüber hinaus können Sie beim Einfügen einer Komponente eine Zeichenkette angeben, mit der die Namen der eingefügten Felder automatisch ergänzt werden. Auf diese Weise ist es möglich, eine Substruktur mehrfach einzufügen und trotzdem eindeutige Feldnamen zu erhalten. Der entsprechende Zusatz zur INCLUDE-Anweisung lautet

```
RENAMING WITH SUFFIX suffix
```

Die Wirkung der beiden neuen Optionen verdeutlicht das folgende Programm.

```
 1 REPORT  yz432100 NO STANDARD PAGE HEADING.
 2 TYPES: BEGIN OF trec1,
 3   f11,
 4   f12,
 5   f13,
 6 END OF trec1.
 7
 8 TYPES: BEGIN OF trec2,
 9   f21,
10   f22,
11   f23,
12 END OF trec2.
13
14 DATA: BEGIN OF rec1.
15   INCLUDE TYPE trec1 AS a.
16   INCLUDE TYPE trec2 AS b.
17 DATA: END OF rec1.
18
19 DATA: BEGIN OF rec2.
20   INCLUDE TYPE trec1 AS a RENAMING WITH SUFFIX low.
21   INCLUDE TYPE trec1 AS b RENAMING WITH SUFFIX high.
22 DATA: END OF rec2.
23
24 rec1-f11   = '1'.
25 rec1-f21   = '4'.
26 rec1-a-f12 = '2'.
27 rec1-b-f22 = '5'.
28 WRITE: / '1. ', rec1.
29 WRITE: / '2. ', rec1-a.
30 WRITE: / '3. ', rec1-b.
31
32 rec2-f11low  = 'A'.
33 rec2-f13low  = 'C'.
34 rec2-f11high = 'D'.
35 rec2-f13high = 'F'.
36 rec2-a-f12   = 'B'.
37 rec2-b-f12   = 'E'.
38 WRITE: / '4. ', rec2.
39
40 CLEAR rec2.
41 MOVE-CORRESPONDING rec1 TO rec2.
42 WRITE: / '5. ', rec2.
43
44 CLEAR rec2.
45 MOVE-CORRESPONDING rec1 TO rec2-a.
```

```
46 MOVE-CORRESPONDING rec1 TO rec2-b.
47 WRITE: / '6. ', rec2.
```

Zu Beginn des Programms werden zwei kurze Feldleistentypen definiert. Die Zeilen 14 bis 17 zeigen, wie diese beiden Typen innerhalb der Deklaration der Feldleiste `rec1` benutzt und mit einem Strukturnamen (hier `a` und `b`) versehen werden. In den Zeilen 19 bis 22 wird eine zweite Feldleiste deklariert, die aber zweimal die Struktur `trec1` includiert. Der `RENAMING`-Zusatz sorgt dafür, dass durch Anfügen der Zeichenketten `low` und `high` trotzdem eindeutige Feldnamen entstehen.

Der folgende Bereich des Programms zeigt die unterschiedlichen Zugriffsmöglichkeiten auf die Teilfelder. Die Anweisungen in den Zeilen 24 und 25 entsprechen dem herkömmlichen Zugriff auf flache Strukturen. Sie weisen nach, dass durch die `INCLUDE`-Anweisung eine flache Struktur entsteht. In den Zeilen 26 und 27 wird zusätzlich der Name der Substruktur benutzt, um auf die Teilfelder zuzugreifen. Das bringt an dieser Stelle noch keinerlei Vorteile sondern demonstriert lediglich die erforderliche Schreibweise. Über die beiden Strukturnamen `a` und `b` ist lediglich der Zugriff auf die drei Felder jeder Substruktur möglich. Die Anweisung

```
rec1-a-f22 = 'X'.
```

würde somit einen Syntaxfehler verursachen, obwohl das Teilfeld `f22` in `rec1` enthalten ist.

Die nachfolgenden `WRITE`-Anweisungen zeigen einen möglichen Einsatzzweck für die Substrukturnamen. Es ist nun der einfache Zugriff auf eine Teilmenge der Struktur möglich.

Der `RENAMING`-Zusatz erstellt eine Feldleiste, in deren flacher Darstellung die Namen der Teilfelder durch einen Suffix eindeutig werden. Die Zeilen 32 bis 35 zeigen dies. Wenn allerdings unter Verwendung der Strukturnamen auf die Teilfelder zugegriffen wird, dann gelten die alten Namen ohne Suffix (siehe Zeile 36 und 37). Diese Eigenheit ist z. B. bei Zuweisungen durch `MOVE-CORRESPONDING` zu beachten, da dort die Feldnamen relevant sind. In Zeile 41 erfolgt die Zuweisung von `rec1` zu `rec2`. Da nur der Name der eigentlichen Struktur benutzt wird, verwendet `die MOVE-CORRESPONDING`-Anweisung die flache Struktur. In dieser Sichtweise sind die Namen der Teilfelder durch Anfügen der Suffixe in `rec2` natürlich unterschiedlich, es findet keine Übertragung von Feldinhalten statt. Das ändert sich, wenn durch Verwendung der Substrukturnamen die ursprünglichen Feldnamen sichtbar sind. In den Zeilen 45 und 46 ist dies der Fall. Nun ist eine Zuweisung der Teilfelder `f11` bis `f13` möglich.

3.2.5 Feldsymbole

Die bisher beschriebenen Zuweisungen und Zugriffsverfahren erlauben zwar einen recht freizügigen Umgang mit Daten, haben aber dort ihre Grenzen, wo systemnahe Probleme gelöst werden müssen. Dies ist z.B. bei dynamischen Datenzugriffen der Fall. So gibt es Unterprogramme, die mit unterschiedlich strukturierten Daten arbeiten müssen, also keine fest kodierten Feldnamen verwenden können. Für derartige spezielle Probleme stellt ABAP so genannte Feldsymbole bereit. Ein Feldsymbol ist entfernt mit einem Pointer aus C verwandt. Ihm wird der Bezug auf ein existierendes Feld zugewiesen. Im weiteren Verlauf kann mit dem Feldsymbol gearbeitet werden, als ob es sich um das Originalfeld handelt.

Grundform

Deklariert werden Feldsymbole mit der Anweisung

```
FIELD-SYMBOLS <field_symbol>.
```

Die spitzen Klammern gehören zum Namen des Feldsymbols. Für die Zuweisung eines Feldes zu einem Feldsymbol ist die Anweisung `ASSIGN` zuständig. Ihr Aufbau entspricht in etwa dem der Anweisung `MOVE`:

```
ASSIGN field[+offset[(length)]] TO <field_symbol>.
```

Bei dieser Form der Zuweisung übernimmt das Feldsymbol den Datentyp des Quellfeldes. Wenn Sie einem Feldsymbol nacheinander Felder unterschiedlichen Typs zuweisen, dann wechselt der Datentyp des Feldsymbols. Das Feldsymbol übernimmt dabei auch andere Eigenschaften des Quellfelds wie z.B. dessen Länge, sofern Sie nicht mit einer expliziten Längenangabe arbeiten. Dies könnte mit hoher Wahrscheinlichkeit zum unerwünschten Überschreiben anderer Daten führen, wenn bei Zuweisungen auf das Bezugsfeld über einen Offset zugegriffen wird. Das muss durch Angabe einer korrekten Längenangabe oder der Angabe von `(*)` verhindert werden. Der letztgenannte Zusatz verhindert bei Zuweisungen das Überschreiten von Feldgrenzen.

```
REPORT yz432110.
FIELD-SYMBOLS <fs>.

DATA: BEGIN OF f,
  c1(3),
  c2(2),
END OF f,
len TYPE i.

f-c2 = 'xx'.
```

```
ASSIGN f-c1 TO <fs>.
DESCRIBE FIELD <fs> LENGTH len IN CHARACTER MODE.
WRITE: / 'Länge des Feldsymbols: ', len.
<fs> = '123456'.
WRITE: / 'Zuweisung ohne Längenangabe:', f-c1, f-c2.

CLEAR f-c1.
ASSIGN f-c1+1(2) TO <fs>.
DESCRIBE FIELD <fs> LENGTH len IN CHARACTER MODE.
WRITE: / 'Länge des Feldsymbols: ', len.
<fs> = '123456'.
WRITE: / 'Zuweisung mit Längenangabe :', f-c1, f-c2.

CLEAR f-c1.
ASSIGN f-c1+2(*) TO <fs>.
DESCRIBE FIELD <fs> LENGTH len IN CHARACTER MODE.
WRITE: / 'Länge des Feldsymbols: ', len.
<fs> = '123456'.
WRITE: / 'Zuweisung mit *           :', f-c1, f-c2.
```

In diesem Beispiel wird zunächst ein Feldsymbol vom Feld `f-c1` abgeleitet. Da dieses Feld lediglich eine Länge von drei Stellen hat, werden bei einer Zuweisung zum Feldsymbol auch nur drei Stellen berücksichtigt. Die erste `WRITE`-Anweisung zeigt das.

Bei den beiden anderen Zuweisungen zum Feldsymbol wird ein Offset benutzt, der eine zusätzliche Längenangabe erzwingt, damit das Feldsymbol nicht die Grenzen des Zielfelds überschreitet. Die Ausgabe der Länge des Feldsymbols zeigt, dass sie durch das System an den verfügbaren Platz im Zielfeld angepasst wird.

In älteren Systemen existiert dieser Zugriffsschutz nicht. Hier kann durch unvorsichtigen Umgang mit Offsets möglicherweise ein anderes Feld überschrieben werden.

Die bisher benutzte Grundform des `ASSIGN`-Kommandos kann durch zusätzliche Optionen in zwei Richtungen erweitert werden. Es sind dies die Typisierung und die dynamische Angabe des Quellfeldes. Durch diese Ergänzungen können Sie explizite Typumwandlungen durchführen sowie generische Anweisungen konstruieren, die mit beliebigen Datenstrukturen umgehen können. Die nachfolgenden Optionen können größtenteils miteinander kombiniert werden. Zu Details vergleichen Sie bitte die Systemdokumentation.

Typisierung

Bei der Grundform der `ASSIGN`-Zuweisung richtet sich der Datentyp des Feldsymbols nach dem Datentyp des Bezugsfelds. Von dieser Vorgabe können Sie abweichen. Dazu haben Sie verschiedene Möglichkeiten. Zunächst können Sie ein typisiertes Feldsymbol anlegen:

```
FIELD-SYMBOLS <field_symbol> TYPE type.
```

Bei Ausführung einer ASSIGN-Anweisung für ein getyptes Feldsymbol wird geprüft, ob der Datentyp und andere Eigenschaften des Bezugsfeldes wie z.B. die Länge zum Datentyp des Feldsymbols passen. Falls dies nicht der Fall ist, entsteht bei der Programmprüfung ein Syntaxfehler. Die Typ-Prüfungen finden somit schon vor der Abarbeitung der Anwendung statt. Da zu diesem Zeitpunkt alle Typinformationen verfügbar sind, ist diese exakte Prüfung möglich. Die Prüfungen sind zum Teil wesentlich strenger als die bei anderen Zuweisungen. So ist für diese Form der Prüfung der Datentyp C inkompatibel zum Datentyp N, obwohl zwischen beiden problemlos Zuweisungen möglich sind.

Weitere Typisierungsmöglichkeiten bestehen zur Laufzeit bei Ausführung der ASSIGN-Anweisung. Hier können Sie zunächst den gewünschten Typ des Feldsymbols vorgeben:

```
ASSIGN field TO <field_symbol> TYPE type.
```

Bei dieser Form der Anweisung findet zunächst die Zuweisung zum Feldsymbol und erst danach dessen Typisierung statt. Die Typprüfungen sind an dieser Stelle daher nicht so streng wie bei der statischen Vorgabe des Typs. Von erheblicher Bedeutung sind allerdings die Längen der beteiligten Felder. Außerdem muss sich der Inhalt des Bezugsfelds in einen Wert des gewünschten Datentyps umwandeln lassen. Der Zieltyp wird, im Gegensatz zur Definition, als Zeichenkette oder als Feld übergeben. Korrekte Formen der Anweisungen sind somit:

```
DATA:
  field(3) TYPE c,
  fs_type VALUE 'C'.

FIELD-SYMBOLS:
  <fs>.

ASSIGN field TO <fs> TYPE 'N'.
ASSIGN field TO <fs> TYPE fs_type.
```

Mit den bisher genannten Varianten legen Sie den Typ des Feldsymbols fest. Sie können aber auch für das Bezugsfeld die Interpretation gemäß einem vorgegebenen Datentyp erzwingen, falls der Typ des Feldsymbols feststeht. Dazu dient der CASTING-Zusatz. Die Grundform lautet:

```
ASSIGN field TO <field_symbol> CASTING.
```

Dabei wird das Quellfeld unabhängig von seinem tatsächlichen Typ als Datenfeld vom Typ des Feldsymbols betrachtet. Voraussetzung ist natürlich, dass das Quellfeld gemäß des angegebenen Typs interpretiert werden kann. Das Feldsymbol muss in diesem Fall bereits als typisiertes Feldsymbol deklariert worden sein.

Die folgenden Varianten verbinden die Interpretation des Bezugsfeldes und die Zuweisung eines neuen Datentyps zum Feldsymbol miteinander. Dabei ist auch eine dynamische Zuweisung des neuen Typs des Feldsymbols möglich.

```
ASSIGN field TO <field_symbol> CASTING TYPE type.
ASSIGN field TO <field_symbol> CASTING TYPE (typename).
ASSIGN field TO <field_symbol> CASTING LIKE typefield.
```

Zur Demonstration der Wirkung der Typisierung soll hier die Modifikation eines bereits vorgestellten Beispiels dienen:

```
REPORT yz432120.
DATA:
  d TYPE D,
  s(8).
FIELD-SYMBOLS
  <fd> TYPE D.

s = '19991225'.
ASSIGN s TO <fd> CASTING.
d = <fd> + 7.
WRITE /(10) d.
d = '19991225'.
d = d + 7.
WRITE /(10) d.
```

Das Feldsymbol hat den Datentyp D. Da bei der Zuweisung des Quellfelds dessen Interpretation als Datum erzwungen wird, erhält das Feldsymbol einen korrekten Wert als Inhalt. Der nachfolgende Ausdruck arbeitet daher mit einem Datum, wodurch sich das gewünschte Ergebnis bei beiden Berechnungen einstellt.

An dieser Stelle noch ein Programm zur Demonstration der Unterschiede zwischen den diversen Typisierungsmöglichkeiten:

```
 1: REPORT yz432130.
 2: DATA:
 3:   fc(8) VALUE '20020117'.
 4:
 5: FIELD-SYMBOLS:
 6:   <d> TYPE D,
 7:   <f>.
 8:
 9: ASSIGN fc TO <f>.    "ok
10: WRITE: /(10) <f>.
11:
12: *ASSIGN fc TO <d>. "nicht ok, da typ von fc und <d>
    unterschiedlich
13: ASSIGN fc TO <d> CASTING.
14: WRITE: /(10) <d>.
```

```
15:
16: ASSIGN fc TO <f> TYPE 'D'.
17: WRITE: /(10) <f>.
```

Zunächst wird ein Zeichenkettenfeld `fc` angelegt und mit einem gültigen Datumswert belegt. Anschließend erfolgt die Deklaration von zwei Feldsymbolen, von denen eines den Datentyp D besitzt, das andere hingegen ohne Typ angelegt wird.

In Zeile 9 wird die Zeichenkette dem untypisierten Feldsymbol zugewiesen. Dabei übernimmt das Feldsymbol den Datentyp des Quellfelds. Die nachfolgende Ausgabeanweisung schreibt somit die unveränderte Zeichenkette auf den Monitor.

Eine Zuweisung des Zeichenkettenfelds zum getypten Feldsymbol `<d>` (Zeile 12) ist nicht möglich, da die Datentypen beider Elemente nicht identisch sind. Diese im Beispiel auskommentierte Anweisung verursacht daher einen Syntaxfehler. Um das Zeichenkettenfeld über ein Feldsymbol als Datumswert auszugeben, haben Sie nun zwei Möglichkeiten. Die Anweisung in Zeile 13 zeigt, wie die korrekte Zuweisung zu einem Feldsymbol des Typs D erfolgen muss. Durch den `CASTING`-Zusatz wird das Quellfeld als Datumsfeld interpretiert und dem getypten Feldsymbol zugewiesen. Dadurch kann die nachfolgende `WRITE`-Anweisung einen korrekten Datumswert ausgeben.

Eine andere Möglichkeit besteht darin, die Zuweisung zu einem ungetypten Feldsymbol durchzuführen, dabei aber einen vom Quellfeld abweichenden Typ des Feldsymbols zu erzwingen. Zeile 16 zeigt die entsprechende Anweisung. Dabei wird erst die Zuweisung durchgeführt, danach der Typ für das Feldsymbol festgelegt. Daher tritt auch kein Problem bei der Typprüfung auf. Die nachfolgende Ausgabe erzeugt auch bei dieser Variante einen korrekten Datumswert.

Dynamische Feldzuweisung

Bisher wurde das Bezugsfeld direkt angegeben. Aber auch indirekte bzw. dynamische Zuweisungen sind möglich. Zunächst kann der Name des Bezugsfelds einem anderen Feld entnommen werden:

```
ASSIGN (field) TO <field_symbol>.
```

Der Name des Verweisfeldes ist in runde Klammern einzuschließen, wobei keine Leerzeichen zwischen Klammer und Feldname stehen dürfen. Eine andere Variante ermöglicht es, ein Teilfeld einer Struktur (Feldleiste) per Index oder über den Namen anzusprechen:

```
ASSIGN COMPONENT [idx | name] OF STRUCTURE struc TO <fs>.
```

In Verbindung mit der Typisierung eines Feldsymbols zur Laufzeit können Sie den Inhalt beliebiger Datenbanktabellen typgerecht aufbereiten und ausgeben. Die Kommandos des folgenden Beispiels, die Ihnen noch nicht bekannt sind, werden in den nachfolgenden Abschnitten beschrieben.

```
REPORT yz432140.
DATA:
  tabname TYPE tabname VALUE 'E070',
  record(2048).
FIELD-SYMBOLS:
  <rec>,
  <field>.

SELECT * FROM (tabname) INTO record UP TO 100 ROWS.
  ASSIGN record TO <rec> CASTING TYPE (tabname).
  DO.
    ASSIGN COMPONENT sy-index OF STRUCTURE <rec> TO <field>.
    IF sy-subrc <> 0.
      EXIT.
    ENDIF.
    WRITE: /(3) sy-index, <field>.
  ENDDO.
ENDSELECT.
```

Die `SELECT`-Anweisung liest Daten aus einer Datenbanktabelle. In der hier verwendeten Form wird der Name der zu lesenden Tabelle dynamisch im Feld `tabname` übergeben. Der hier benutzte Tabellenname eignet sich besonders, da neben einfachen Zeichenketten auch Datums- und Zeitangaben enthalten sind. Sie haben so die Möglichkeit, sich von der korrekten Ausgabeformatierung zu überzeugen. Sie können durch Modifikation des Wertes für `tabname` auch andere Tabellen lesen. Benutzen Sie dazu Tabellennamen wie `TADIR`, `TOOO` oder `USR01`, da in diesen Tabellen immer Daten enthalten sind.

Weitere Optionen in der `SELECT`-Anweisung sorgen dafür, dass jeder gelesene Datensatz im unstrukturierten Feld `record` abgelegt wird und dass maximal 100 Datensätze gelesen werden.

Die erste `ASSIGN`-Anweisung legt ein Feldsymbol auf das Feld `record`, wobei das Feldsymbol typisiert wird. Als Typ dient wiederum der Tabellenname, der indirekt im Feld `tabname` übergeben wird. Dies ist möglich, da ja auch Dictionary-Elemente als Typbezeichner benutzt werden können. Innerhalb der nun folgenden Schleife finden Sie eine zweite `ASSIGN`-Anweisung, mit der ein zweites Feldsymbol nacheinander auf die einzelnen Felder innerhalb des Datensatzes gelegt wird. Bei dieser Form der Zuweisung übernimmt das Feldsymbol den Datentyp des Bezugsfelds. Die nachfolgende `WRITE`-Anweisung kann somit eine typgerechte Ausgabekonvertierung durchführen.

Löschen

Die mit ASSIGN durchgeführte Zuweisung kann mit dem Kommando

```
UNASSIGN <field_symbol>.
```

wieder rückgängig gemacht werden. Das Feldsymbol verweist danach nicht mehr auf gültige Daten. Der logische Ausdruck

```
... <field_symbol> IS ASSIGNED
```

liefert ein Leerzeichen als Symbol für „Falsch“ zurück.

3.2.6 Feldeigenschaften ermitteln

Datenfelder besitzen Eigenschaften, die zur Laufzeit ermittelt und ausgewertet werden können. Innerhalb von Bedingungen können Sie ausgewählte Zustände von Datenfeldern mit dem Operator IS ermitteln. Die Varianten

```
... field IS SPACE
```

bzw.

```
... field IS INITIAL
```

prüfen, ob der Inhalt eines Feldes das Leerzeichen ist oder ob das Feld mit seinem Initialwert belegt ist. Mit der Anweisung

```
... <field_symbol> IS ASSIGNED ...
```

können Sie testen, ob dem Feldsymbol field_symbol ein Feld zugewiesen wurde. Innerhalb von Funktionsbausteinen können Sie die Bedingung

```
... parameter IS REQUESTED ...
```

testen. Der Vergleich ist dann erfüllt, wenn dem formalen Parameter beim Aufruf des Funktionsbausteins ein aktueller Parameter zugewiesen wurde.

Neben der Auswertung von Zuständen können Sie mit dem Kommando DESCRIBE eine Reihe weiterer Eigenschaften ermitteln. Das Kommando wird in der Form

```
DESCRIBE field option result.
```

aufgerufen. Dabei ist *field* der Name des auszuwertenden Feldes. Die möglichen Bezeichner für *option* entnehmen Sie bitte Tabelle 3.12. Das jeweilige Ergebnis wird im Feld *result* abgelegt, dessen Datentyp dem zu erwartenden Ergebnis entsprechen muss. Einige der Werte sind nur für Felder mit Dictionary-Bezug verfügbar.

Option	Beschreibung
`LENGTH [ IN CHARACTER MODE | IN BYTE MODE]`	Definierte Länge
`TYPE`	Datentyp
`TYPE type COMPONENTS count`	Type: Kennzeichen für komplexen Datentyp Count: Zahl der Komponenten
`OUTPUT-LENGTH`	Ausgabelänge
`DECIMALS`	Zahl der Dezimalstellen
`EDIT MASK`	Ausgabemaske
`HELP-ID`	ID des Hilfe-Textes

Tabelle 3.12
Optionen zum Kommando DESCRIBE

Für den Datentyp liefert das Kommando nicht nur die Bezeichner der 8 vordefinierten Datentypen zurück sondern ggf. auch einen der in Tabelle 3.13 aufgeführten Werte.

Alternativer Datentyp	Beschreibung
b	1 Byte Integer ohne Vorzeichen
g	Datentyp: STRING
h	Interne Tabelle
l (kleines L)	Datenreferenz
r	Objektreferenz
s	2-Byte-Integer mit Vorzeichen
u	Struktur ohne interne Tabelle
v	Struktur, die mindestens eine interne Tabelle enthält
Y	Datentyp: XSTRING

Tabelle 3.13
Zusätzliche Kennbuchstaben für Datentypen

3.2.7 Interne Tabellen

Neben den vom Data Dictionary verwalteten Tabellen existiert eine zweite Tabellenform, die interne Tabelle. Diese existiert nicht auf der Datenbank, sondern nur im Hauptspeicher des Computers. Demzufolge ist sie auch nicht für die dauerhafte Speicherung von Informationen gedacht, sondern dient innerhalb einer Anwendung zur temporären Datenspeicherung und -bearbeitung. Von wenigen Ausnahmen abgesehen müssen Sie interne Tabellen unter ABAP immer dann benutzen, wenn Sie mehrere Datensätze identischer Struktur in einem gemeinsamen Datenobjekt zusammenfassen möchten. Der aus anderen Programmiersprachen bekannte Begriff des Arrays existiert unter ABAP nicht. Als Ausgleich dazu bietet ABAP für die internen Tabellen zusätzlich zu den tabellenorientierten Zugriffsmöglichkeiten auch einige Kommandos an, mit denen Sie auf interne Tabellen per Index, also ähnlich wie auf ein Array zugreifen können.

Interne Tabellen werden aus den verschiedensten Gründen eingesetzt. Sehr häufig puffern sie innerhalb einer Anwendung das Ergebnis einer `SELECT`-Anweisung und verbessern so das Laufzeitverhalten. Interne Tabellen dienen andererseits auch zum Datenaustausch mit Unterprogrammen oder Funktionsbausteinen. Des Weiteren erlauben sie es, temporäre Datenbestände nach verschiedenen Kriterien zu sortieren und zu verdichten.

Beim Datenaustausch zwischen interner Tabelle und Anwendung ist eine Besonderheit zu beachten. Der Inhalt der Tabelle selbst ist dem Programm weitgehend verborgen, der Zugriff kann im Allgemeinen nur über eine Feldleiste, die Kopfzeile oder Arbeitsbereich genannt wird, erfolgen. Spezielle Kommandos lesen einen Datensatz der internen Tabelle und stellen ihn im Arbeitsbereich bereit, andere wiederum fügen den Inhalt des Arbeitsbereiches in die interne Tabelle ein. Für interne Tabellen ist der Arbeitsbereich die einzige Schnittstelle zwischen Datenbestand und Anwendung. Je nach Tabellentyp kann der Arbeitsbereich ein beliebiges Feld sein, das ausreichend Platz zur Aufnahme des Datensatzes bieten muss oder aber eine Feldleiste, die denselben Namen besitzt wie die interne Tabelle. Im letzteren Fall spricht man von einer Kopfzeile.

Das Konzept der internen Tabellen wurde mit Version 4.0 erheblich erweitert. Es existieren nun unterschiedliche Arten interner Tabellen, die sich durch die Behandlung eines Schlüssels für die Datensätze unterscheiden. In diesem Zusammenhang werden umfangreiche Erweiterungen der Syntax wirksam. Die drei neuen Gruppen sind:

- `Standard-Tabelle (Standard Table)`
- `Sortierte Tabelle (Sorted Table)`
- `Hash-Tabelle (Hashed Table)`

Die Tabelle der Art STANDARD TABLE ähnelt in vielen Merkmalen den aus älteren Versionen der R/3-Software bekannten Grundform. Genau genommen sind die herkömmlichen internen Tabellen eine Teilmenge der Tabellenart STANDARD TABLE. Interne Tabellen, die mit Kommandos erzeugt wurden, die bereits vor Version 4.0 verfügbar waren, haben somit automatisch die Tabellenart STANDARD TABLE. Die neuen Varianten der Deklaration erlauben aber, zusätzliche Eigenschaften für diese Tabellenart zu setzen. Daher kann die neue Tabellenart die bisherige Grundform problemlos ersetzen. Wenn im Folgenden von Tabellen der Art STANDARD TABLE oder kurz von Standard-Tabellen gesprochen wird, gilt das sowohl für Tabellen, denen diese Tabellenart explizit zugewiesen wurde als auch für die herkömmlichen internen Tabellen.

Zugriffe auf Standard-Tabellen, egal ob lesend oder schreibend, erfolgen sequenziell oder über die Datensatznummer. Es existiert keine automatische, auf einem Tabellenschlüssel beruhende Indexverwaltung. Demzufolge können sich auch beliebig viele Datensätze mit identischem Inhalt in der Tabelle befinden. Die Datensätze werden durch die Tabelle nicht sortiert, eine definierte Reihenfolge der Datensätze muss explizit hergestellt werden. Die Suche nach einem bestimmten Datensatz erfolgt, von Ausnahmen abgesehen, durch zeitaufwändiges sequenzielles Lesen des Datenbestandes.

Die nächste Variante interner Tabellen sind die sortierten Tabellen. Sie erhalten die Bezeichnung SORTED TABLE. Das R/3-System hält den Inhalt dieser Tabellen beim Einfügen von neuen Datensätzen stets in einem sortierten Zustand. In einer LOOP-Schleife über die Tabelle erhalten Sie die Datensätze daher automatisch in sortierter Reihenfolge. Bei Suchfunktionen wird eine binäre Suche durchgeführt. Auf die Datensätze dieser Tabellenart können Sie auch mit Indexangaben sowohl lesend als auch, unter Berücksichtigung der Sortierreihenfolge, schreibend zugreifen. Das sortierende Einfügen erfordert vom Laufzeitsystem zusätzlichen Aufwand, der sich in einem erhöhten Zeitaufwand niederschlägt. Sortierte Tabellen werden vor allem dann eingesetzt, wenn der Datenbestand in einer Schleife in geordneter Reihenfolge aufbereitet werden soll.

Als letzte Variante kommen Tabellen zum Einsatz, deren Schlüssel das R/3-System nach dem Hash-Verfahren verwaltet. Sie erhalten die Bezeichnung HASHED TABLE. Auf Grund des Verwaltungsprinzips sind Zugriffe über eine Datensatznummer nicht möglich. Die Datensätze in dieser Tabelle sind nicht sortiert. Neue Datensätze werden immer am Ende der Tabelle angefügt. Innerhalb einer LOOP-Schleife über den Datenbestand erhalten Sie die Datensätze daher in der Reihenfolge, in der sie in die Tabelle eingefügt wurden. Allerdings wird gemäß der Schlüsseldefinition ein Index verwaltet, der den Zugriff auf Datensätze der internen Tabelle über diesen Schlüssel ermöglicht (Kommando READ TABLE ... WITH TABLE KEY ...). Der Vorteil derartiger Tabellen besteht darin, dass die Zeit zum Finden eines Datensatzes unabhängig vom Platz des Datensatzes in der Tabelle ist. Außerdem wird die Tabelle beim Einfügen eines neuen Datensatzes nicht umsortiert, so dass der Zeitaufwand beim Einfügen geringer ist als bei

sortierten Tabellen. Hash-Tabellen eignen sich daher besonders für Tabellen, in denen häufig nach einem Datensatz gesucht werden muss.

Zu den bisher erwähnten Tabellenarten können Sie mit dem Kommando DATA reale Datenobjekte erstellen. Dabei können Sie die Eigenschaften der Tabelle direkt in der DATA-Anweisung oder indirekt über einen Tabellentyp beschreiben. Dabei müssen Sie jedoch beachten, dass interne Tabellen, die mit Hilfe eines Tabellentyps oder durch die neuen Syntax-Varianten deklariert werden, nur dann über eine Kopfzeile verfügen, wenn diese bei der Deklaration explizit erzeugt wird.

Tabellen der Art SORTED TABLE und HASHED TABLE können an Unterprogramme nur noch als USING/CHANGING-Parameter und nicht mehr als TABLES-Parameter übergeben werden. Dies erfordert eine neue Form der Typisierung der Unterprogramm-Parameter. Nach Möglichkeit sollte für interne Tabellen daher immer ein benutzerdefinierter Datentyp zur Verfügung stehen.

Neben den drei beschriebenen Tabellenarten existieren zwei weitere Tabellenbezeichnungen, die nur zur Typisierung von Unterprogramm-Parametern dienen. Tabellentypen, die auf diesen Tabellenarten beruhen, können nicht zur Deklaration interner Tabellen in einer DATA-Anweisung benutzt werden. Es handelt sich um INDEX TABLE als Oberbegriff für Standard- und sortierte Tabellen sowie um ANY TABLE als Kennzeichen für alle drei Tabellengruppen.

An dieser Stelle sei nochmals ausdrücklich darauf hingewiesen, dass Zeilentypen nicht nur einfache Felder sondern auch komplexe Datenobjekte enthalten können. Ein Feld einer internen Tabelle kann daher Feldleisten oder sogar interne Tabellen enthalten.

Definition

Ab Release 4.6 können Sie interne Tabellen im Data Dictionary definieren. Diese Definitionen sind dann ebenso wie Dictionary-Tabellen oder Datenelemente als Datentypen verfügbar. Details dazu finden Sie im Kapitel 5, das sich ausschließlich mit dem Data Dictionary befasst. Darüber hinaus können Sie interne Tabellentypen natürlich auch innerhalb Ihrer Anwendung definieren. Ein Beispiel für die Definition eines Tabellentyps im Dictionary folgt am Ende dieses Abschnitts. Für das Verständnis dieses Beispiels sind die nachfolgend dargestellten Grundlagen allerdings Voraussetzung. Zur Definition eines Tabellentyps direkt im Programm dient die Anweisung:

```
TYPES itabtype
  [ TYPE | LIKE ]
  [ STANDARD TABLE |
    SORTED   TABLE |
    HASHED   TABLE |
    INDEX    TABLE |
    ANY      TABLE ]
  OF
```

```
[ linetype | lineobject ]
[ WITH [ UNIQUE | NON-UNIQUE ]
       [ KEY fieldlist   |
         KEY TABLE LINE |
         DEFAULT KEY    | ] ]
[ INITIAL SIZE n ].
```

Unmittelbar nach TYPES geben Sie den Namen des zu erzeugenden Tabellentyps an. Anschließend legen Sie die Tabellenart fest. Dabei können Sie alle 5 erwähnten Tabellenarten benutzten. Datenobjekte können Sie aber nur mit Tabellentypen deklarieren, die eine der drei Arten STANDARD TABLE, SORTED TABLE oder HASHED TABLE besitzen. Tabellentypen mit den beiden anderen Tabellenarten können nur zur Typisierung von Unterprogramm-Parametern eingesetzt werden.

Nach der Tabellenart müssen Sie die Struktur der internen Tabelle festlegen. Dazu geben Sie entweder einen Zeilentyp oder ein zeilenartiges Datenobjekt an. Diese Angabe muss mit der LIKE- bzw. Type-Anweisung korrespondieren. Wenn Sie zu Beginn der Definition durch

```
TYPES ... TYPE
```

festgelegt haben, dass Sie die Struktur der Tabelle durch einen Zeilentyp beschreiben wollen, müssen Sie nach OF einen Zeilentyp angeben. Wenn Sie hingegen mit

```
TYPES ... LIKE
```

den Bezug auf einen existierenden Datensatz herstellen möchten, so müssen Sie nach OF den Namen einer Feldleiste oder einer Dictionary-Struktur angeben. Sie können an dieser Stelle auch einen unstrukturierten Datentyp verwenden. In diesem Fall wird eine interne Tabelle definiert, die nur eine Spalte mit den Eigenschaften des unstrukturierten Elements besitzt. Derartige Tabellen werden z.B. benötigt, um der SELECT-Anweisung dynamische Selektionskriterien zu übergeben. In einer so erzeugten Tabelle sind keine Spaltennamen verfügbar. Zugriffe erfolgen immer nur über den Namen der Tabelle. Ein Beispiel für Deklaration und Verwendung einer derartigen Tabelle finden Sie weiter unten.

Anschließend legen Sie fest, ob die interne Tabelle einen Schlüssel erhalten soll oder nicht und ob dieser Schlüssel eindeutig ist. Diese Angabe ist nicht für alle Tabellenarten sinnvoll. Beim Versuch, einer Standard-Tabelle einen eindeutigen Schlüssel zuzuweisen, erhalten Sie bei der Syntaxprüfung einen Fehler. Für Standard-Tabellen ist ein nicht eindeutiger Schlüssel als Vorgabe eingestellt, so dass Sie für diese Tabellen auf die Festlegung der Schlüsselart verzichten können. Dies ist ebenfalls bei sortierten Tabellen möglich, dadurch entsteht aber ein generischer Tabellentyp, der nur zur Typisierung von Unterprogramm-Parametern, nicht aber zur Deklaration von Datenobjekten benutzt werden kann. Für die vollwertige Definition eines Datentyps für sortierte Tabellen ist die Angabe einer der beiden Schlüsselarten zwingend erforderlich. Auch für Hash-Tabellen

muss eine Schlüsselart angegeben werden, hier ist aber nur ein eindeutiger Schlüssel zugelassen. Der Aufbau des Schlüssels wird durch einen der nachfolgenden Parameter

- `KEY fieldlist` oder
- `KEY TABLE LINE` oder
- `DEFAULT KEY`

beschrieben. Mit der ersten Variante können Sie die Schlüsselfelder explizit benennen. Die zweite Schlüsseldefinition verwendet die gesamte Tabellenzeile als Schlüssel. Die herkömmliche Variante der Schlüsselbildung (alle elementaren Nicht-Zahlenfelder) wählen Sie mit der dritten Anweisung.

Die letzte Variante bedarf einer Erläuterung. Vor Schaffung der neuen Tabellenarten verfügten interne Tabellen nicht über einen echten Schlüssel, mit dem ein Datensatz identifiziert werden konnte. Um trotzdem einen Datensatz in einer internen Tabelle suchen zu können, waren per Definition alle einfachen Felder, die keine Zahlenfelder waren, (Typ F, I und P), Schlüsselfelder der Tabelle. Dieser Schlüssel wurde vor allem von einer Variante des Kommandos `READ` benutzt.

Mit dem abschließenden Parameter `INITIAL SIZE` legen Sie die Größe eines Pufferbereichs im Hauptspeicher fest. Bei einem Wert von 0 übernimmt das System die Berechnung der Puffergröße. Im Normalfall ist diese Einstellung ausreichend.

Fehlt die Deklaration des Schlüssels oder wird einer der Tabellentypen `INDEX TABLE` bzw. `ANY TABLE` benutzt, handelt es sich um einen generischen Typ. Derartige Datentypen können Sie nur zur Typisierung von Unterprogramm-Parametern verwenden. Eine Verwendung von generischen Datentypen innerhalb der `DATA`-Anweisung verursacht einen Syntaxfehler.

Als abkürzende Schreibweise zur Definition eines Typs für eine Standard-Tabelle dient die Anweisung

```
TYPES itabtype [TYPE linetype | LIKE lineobject ] OCCURS n.
```

Diese Anweisung entspricht der aus älteren R/3-Versionen verwendeten Anweisung. Dabei ist die Wirkung des Parameters `OCCURS` identisch mit der von `INITIAL SIZE`.

Beachten Sie bitte, dass Sie sich bei der Definition eines Tabellentyps immer auf eine existierende Struktur oder einen bereits vorhandenen Zeilentyp stützen müssen. Sie können die Struktur einer internen Tabelle nicht direkt innerhalb der Definition der internen Tabelle (z.B. mit `BEGIN OF` und `END OF`) festlegen. Dieser Zeilentyp ist separat zu definieren, so wie im Abschnitt über Feldleisten bereits beschrieben.

Nachfolgend finden Sie einige Beispiele zur Definition interner Tabellen. Die hier vorgestellten Anweisungen stellen den Beginn eines Programms dar, das im Verlauf dieses Abschnittes durch weitere Anweisungen ergänzt wird. Das folgende Listing enthält nur Definitionen, so dass das Programm in diesem Stadium noch nicht ausgeführt werden kann.

```
REPORT yz432150.
*..definition
TABLES: tadir.
TYPES:
  t_line(72),

  BEGIN OF tr_tadir,
    pgmid(4),
    object   LIKE tadir-object,
    obj_name TYPE sobj_name,
  END OF tr_tadir,

  ti_standard
    TYPE STANDARD TABLE OF tr_tadir
    INITIAL SIZE 0,

  ti_sorted
    TYPE SORTED TABLE   OF tr_tadir
    WITH UNIQUE DEFAULT KEY
    INITIAL SIZE 0,

  ti_hashed
    TYPE HASHED TABLE   OF tr_tadir
    WITH UNIQUE KEY pgmid object obj_name
    INITIAL SIZE 0,

  ti_flat
    TYPE STANDARD TABLE OF t_line
    INITIAL SIZE 0,

  ti_old
    TYPE tr_tadir OCCURS 0.
```

Bei den ersten beiden Datentypen handelt es sich nicht um Datentypen für interne Tabellen, sondern um einen elementaren Typ und eine Feldleiste. Diese beiden Elemente werden als Hilfsmittel zur Definition der internen Tabellen benötigt. Der erste Tabellentyp ist `ti_standard`. Er wird für eine Standardtabelle angelegt, welche die Struktur von `ti_tadir` besitzen soll. Angaben zum Schlüssel erfolgen nicht.

Etwas aufwändiger ist die Definition von `ti_sorted`. Hier wird durch eine zusätzliche Option festgelegt, dass diese Tabelle einen eindeutigen Schlüssel besit-

zen soll, dessen Aufbau der Standard-Vorgabe (alle elementaren Nicht-Zahlenfelder) entspricht.

Die Definition eines Schlüssels ist auch für den dritten Tabellentyp `ti_hash` erforderlich. Hier wird das Schlüsselfeld explizit benannt, um auch diese Variante des Kommandos praktisch vorzuführen.

Der Typ `ti_flat` beschreibt eine Standard-Tabelle, die nur aus einer Spalte besteht. Bezüglich des Schlüssels können zusätzliche Angaben entfallen. Die letzte Definition für `ti_old` verwendet eine bereits unter Version 3.x existierende Variante der `TYPES`-Anweisung für interne Tabellen.

Deklaration

Auf Grund des verallgemeinerten Typkonzeptes ähnelt die Deklaration von internen Tabellen der Deklaration anderer Datenobjekte. Die Grundform lautet:

```
DATA itab TYPE itabtype [ WITH HEADER LINE ].
```

Durch den Zusatz `WITH HEADER LINE` erzeugen Sie eine interne Tabelle mit Kopfzeile. Fehlt dieser Zusatz, muss der Datenaustausch zwischen Anwendung und interner Tabelle über einen separaten Arbeitsbereich erfolgen.

Auch bei der Deklaration von internen Tabellen können Sie ggf. auf eine separate Definition eines Tabellentyps verzichten. Sie müssen dann die für die Definition erforderlichen Optionen in der Deklaration notieren. So entsteht die Anweisung

```
DATA itabtype
  [ TYPE | LIKE ]
  [ STANDARD TABLE |
    SORTED   TABLE |
    HASHED   TABLE ]
  OF
  [ linetype | lineobject ]
  [ WITH [ UNIQUE | NON-UNIQUE ]
         [ KEY fieldlist |
           KEY TABLE LINE |
           DEFAULT KEY    | ] ]
  [INITIAL SIZE n]
  [WITH HEADER LINE].
```

In dieser Anweisung dürfen keine Angaben enthalten sein, mit denen ein generischer Typ definiert werden würde. Das bedeutet, dass die Tabellenarten `INDEX TABLE` und `ANY TABLE` nicht zugelassen sind. Außerdem ist für die Tabellenart `SORTED TABLE` sowie `HASHED TABLE` die Angabe einer korrekten Schlüsselart (`UNIQUE` oder `NON-UNIQUE`) obligatorisch. Bis auf `WITH HEADER LINE`, mit dem wieder eine Kopfzeile erzwungen wird, entsprechen die übrigen Parameter denen der `TYPES`-Anweisung.

Für die Deklaration einer Tabelle der Art STANDARD TABLE mit dem Standard-Schlüssel existiert eine Kurzform. Sie lautet:

```
DATA itab [ TYPE TABLE OF linetype | LIKE TABLE OF lineobj ].
```

Hier fehlt eine Angabe für die Speicherbelegung (Parameter INITIAL SIZE). Er wird vom System automatisch mit 0 belegt, da der Zusatz TABLE OF darauf hinweist, dass eine interne Tabelle und keine Feldleiste erzeugt werden soll. Relativ ähnlich ist die Anweisung

```
DATA itab [ TYPE linetype | LIKE lineobj ] OCCURS n [ WITH
HEADER LINE ].
```

Auch hier wird eine Tabelle des Typs STANDARD TABLE erzeugt. Dieses Kommando steht bereits ab der Version 3.x zur Verfügung.

Für Standard-Tabellen ist noch eine weitere Komprimierung der Deklaration möglich. So können Sie die Struktur der Tabelle ohne Bezug auf einen Zeilentyp direkt in der Deklaration definieren. Sie sind somit nicht auf einen explizit vorhandenen Datentyp angewiesen. In diesem Fall lautet die Syntax der Deklarationsanweisung wie folgt:

```
DATA:
  BEGIN OF itab OCCURS n,
    declarations,
  END OF itab.
```

Sie entspricht bis auf den OCCURS-Parameter der Deklaration einer Feldleiste. Das bedeutet, dass auch hier die bei Feldleisten beschriebenen Besonderheiten im Zusammenhang mit dem Kommando INCLUDE STRUCTURE auftreten können. Interne Tabellen, die auf diese Weise angelegt werden, haben immer eine Kopfzeile.

Nachfolgend einige Beispiele zur Deklaration, in denen die zuvor angelegten Tabellentypen Verwendung finden.

```
*..declaration
DATA:
  r_tadir TYPE tr_tadir,
  i_standard TYPE tr_tadir OCCURS 0 WITH HEADER LINE,
  i_sorted TYPE SORTED TABLE OF tr_tadir
    WITH UNIQUE KEY pgmid object obj_name
    INITIAL SIZE 0,
  i_hashed TYPE ti_hashed,
  i_condition TYPE ti_flat WITH HEADER LINE.
```

Die erste Anweisung deklariert eine Feldleiste, die später als Arbeitsbereich für die verschiedenen internen Tabellen dient. Die zweite Anweisung erzeugt eine Standard-Tabelle. Dazu findet die schon unter Version 3.x verfügbare Variante der Deklaration Anwendung. Ersichtlich ist dies vor allem durch die Verwen-

dung des OCCURS-Parameters. Der zuvor definierte Tabellentyp wird hier nicht eingesetzt sondern nur der Zeilentyp tr_tadir. Ebenfalls ohne Bezug auf den existierenden Tabellentyp erfolgt die Deklaration der sortierten Tabelle i_sorted. Abweichend vom Tabellentyp ti_sorted wird hier der Schlüssel durch Aufzählung der Schlüsselfelder vorgegeben. Da der Zusatz with header line hier fehlt, besitzt diese Tabelle keine Kopfzeile.

Während die beiden vorangegangenen Deklarationen relativ viel Schreibarbeit erfordern, vereinfacht sich die Deklaration einer internen Tabelle durch Verwendung des Tabellentyps erheblich. Die Deklaration der Hash-Tabelle ist nicht aufwändiger als die eines elementaren Datenfelds. Auch hier entsteht eine Tabelle ohne Kopfzeile, da der entsprechende Zusatz fehlt. Er kann aber auch bei dieser Form der Deklaration benutzt werden, wie die letzte Anweisung für die Tabelle i_condition demonstriert.

Zugriffe auf interne Tabellen

Die Kommandos zur Arbeit mit internen Tabellen können in zwei Gruppen eingeteilt werden. Dies sind

- Kommandos zum Modifizieren des Tabelleninhalts
- Kommandos zum Lesen von Datensätzen

Jede dieser Kommandogruppen enthält diverse Kommandos, die nicht immer auf alle Tabellenarten angewendet werden können. Des Weiteren kann unterschieden werden in Kommandos, die generisch arbeiten und somit für alle Tabellenarten verwendet werden können und solche, die nur für indexbasierte Tabellen (Standard-Tabellen und sortierte Tabellen) benutzt werden. Die generischen Formen der Kommandos sind oft durch das Schlüsselwort TABLE innerhalb der Anweisung zu erkennen.

Anfügen von Datensätzen

Neue Datensätze werden mit dem Kommando APPEND an das Ende einer internen Tabelle angefügt. Dieses Kommando ist nicht für Hash-Tabellen zugelassen. Falls Sie einen Datensatz mit APPEND an eine Tabelle der Tabellenart SORTED TABLE anfügen, muss sich die Tabelle auch nach dem Anfügen noch im sortierten Zustand befinden, sonst wird ein Laufzeitfehler ausgelöst. Dieses Kommando wird daher fast ausschließlich für interne Tabellen der Tabellenart STANDARD TABLE verwendet. Die Syntax des Kommandos lautet:

```
APPEND [ record TO | INITIAL LINE TO ] itab.
```

Die kürzestmögliche Form des Kommandos ist somit

```
APPEND itab.
```

In diesem Fall wird der einzufügende Datensatz der Kopfzeile der internen Tabelle itab entnommen und an diese angefügt. Falls Sie mit Tabellen ohne Kopf-

zeile arbeiten oder aber die Daten aus einem anderen Datensatz als der Kopfzeile entnehmen wollen, müssen Sie dazu die Option TO verwenden:

```
APPEND record TO itab.
```

Durch die Option INITIAL LINE TO fügen Sie einen leeren Datensatz an die Tabelle an. Beispiele zum APPEND-Kommando finden Sie am Ende des nachfolgenden Abschnitts über das INSERT-Kommando.

Einfügen von Datensätzen

Das Einfügen neuer Datensätze mit dem Kommando INSERT ist etwas komplizierter als das bloße Anfügen am Ende der Tabelle. Grund dafür ist, dass die Position des einzufügenden Datensatzes durch verschiedene Methoden bestimmt werden muss. Die allgemeingültige, generisch arbeitende Version des Kommandos lautet

```
INSERT [ record INTO | INITIAL LINE INTO ] TABLE itab.
```

Diese Variante ist für alle Tabellenarten verwendbar. Bezüglich der Bereitstellung der Daten in der Kopfzeile oder einem separaten Datensatz gelten die Bemerkungen beim Kommando APPEND. Bei sortierten Tabellen sorgt dieses Kommando dafür, dass der neue Datensatz entsprechend des definierten Schlüssels eingefügt wird. Bei Hash-Tabellen wird der Datensatz am Ende angefügt, allerdings wird die interne Schlüsseltabelle aktualisiert. Sollte der Schlüssel als eindeutig (UNIQUE) definiert sein, verursachen Duplikate einen Returncode von 4 im Feld SY-SUBRC.

Für Tabellen ohne automatische Sortierung, also Tabellen der Tabellenart STANDARD TABLE, arbeitet das generische INSERT TABLE-Kommando wie das APPEND-Kommando und fügt den Datensatz am Ende der Tabelle an. Falls dieses Verhalten nicht erwünscht ist, kann durch eine nicht generische Form des INSERT-Kommandos ein Datensatz an einer genau definierten Position in die Tabelle eingefügt werden:

```
INSERT [ record INTO | INITIAL LINE INTO ] itab [ INDEX index ].
```

Falls bei sortierten Tabellen durch das Einfügen die Sortierreihenfolge verletzt wird, entsteht ein Laufzeitfehler. Beachten Sie hier bitte, dass sich diese Anweisung durch das fehlende Schlüsselwortes TABLE innerhalb der Anweisung von der vorangegangenen Variante des Kommandos unterscheidet. Da dieses Kommando nicht generisch arbeitet, kann es nicht für Hash-Tabellen verwendet werden.

Dieses Kommando ist auch ohne Index-Angabe verwendbar, allerdings nur innerhalb einer LOOP-Schleife über die interne Tabelle, in die eingefügt werden soll. In diesem Fall wird der neue Datensatz an der aktuellen Position des Datensatzzeigers eingefügt. Außerhalb von LOOP-Schleifen verursacht das Fehlen eines Index einen Laufzeitfehler. Dieser Fehler wird durch die Syntaxprüfung nicht erkannt.

Für beide Formen des INSERT-Kommandos existiert eine Variante, mit der Sie mehrere Datensätze aus einer Tabelle in eine andere Kopieren können. Dazu wird der Quellbereich durch zwei optionale Index-Angaben eingeschränkt. Fehlen die Index-Angaben, wird von der ersten bis zur letzten Zeile gelesen.

Die allgemein verwendbare Form des Kommandos lautet:

```
INSERT LINES OF itab_source [ FROM index1 ] [ TO index2 ]
  INTO TABLE itab_target.
```

Auch hier gilt wieder, dass für Standard-Tabellen eigentlich ein APPEND stattfindet. Für alle Tabellen außer Hash-Tabellen können Sie daher mit einer zweiten Form des Kommandos festlegen, an welcher Position der Zieltabelle die neuen Datensätze eingefügt werden.

```
INSERT LINES OF itab_source [ FROM index1 ] [ TO index2 ]
  INTO itab_target [ INDEX index3 ].
```

Nachfolgend einige Beispiele zum An- und Einfügen von Datensätzen in interne Tabellen.

```
*..append
i_standard-pgmid    = 'R3TR'.
i_standard-object   = 'PROG'.
i_standard-obj_name = 'YZ332020'.
APPEND i_standard.
r_tadir = i_standard.
APPEND r_tadir TO i_sorted.

i_standard-obj_name = 'YZ332030'.
INSERT i_standard INDEX 1.

r_tadir = i_standard.
INSERT r_tadir INTO TABLE i_sorted.
INSERT r_tadir INTO TABLE i_hashed.

r_tadir-obj_name = 'YZ332010'.
INSERT r_tadir INTO TABLE i_standard.
INSERT r_tadir INTO TABLE i_sorted.
INSERT r_tadir INTO TABLE i_hashed.
```

Die Tabelle i_standard verfügt über eine Kopfzeile. Die ersten 3 Anweisungen füllen diese Kopfzeile mit Werten. Die Anweisung in der vierten Zeile fügt die Kopfzeile an die gleichnamige interne Tabelle an. Die beiden anderen Tabellen (i_sorted und i_hashed) verfügen nicht über eine Kopfzeile. Der Datenaustausch mit diesen Tabellen erfordert den Einsatz eines separaten Feldes als Arbeitsbereich. Dieser Arbeitsbereich wird in der fünften Zeile mit den Werten aus der Kopfzeile der Tabelle i_standard gefüllt und anschließend an die sortierte Tabelle angefügt. Da diese Tabelle leer ist, können auch keine Probleme bezüg-

lich der Sortierreihenfolge entstehen. Die nachfolgenden Anweisungen demonstrieren das Einfügen in eine Standard-Tabelle über einen expliziten Index sowie den Einsatz der generischen Variante des INSERT-Kommandos.

Modifizieren von Datensätzen

Das Modifizieren existierender Datensätze einer internen Tabelle erfolgt mit dem Kommando MODIFY. Auch hier existiert eine generische Variante, die für alle Tabellenarten verwendet werden kann und eine nicht generische, die nicht für Hash-Tabellen zugelassen ist. Die Syntax der allgemein verwendbaren Anweisung lautet:

```
MODIFY TABLE itab [ FROM record ] [ TRANSPORTING field_1 ...
field_n ].
```

Die Daten, die vom Kommando benötigt werden, stehen entweder in der Kopfzeile der Tabelle oder aber in einem separaten Arbeitsbereich record. Das Kommando extrahiert aus dem Datensatz zunächst den Schlüssel und sucht in der internen Tabelle den ersten Datensatz, der zu diesem Schlüssel passt. Wie der Schlüssel aufgebaut ist, also welche Felder ausgewertet werden müssen, wurde bereits bei der Definition der Tabelle festgelegt. Der Suchvorgang hängt von der Art der Tabelle ab. In Standard-Tabellen wird beispielweise sequenziell gesucht, was bei wiederholten Zugriffen auf große Tabellen zu erheblichen Performance-Problemen führen kann.

Wird ein passender Eintrag gefunden, werden dessen Daten durch die aus der Kopfzeile oder dem Arbeitsbereich ersetzt. Bei Bedarf können Sie durch die Option TRANSPORTING die Zahl der zu übertragenden Felder einschränken. Die Feldliste kann dabei auch dynamische Angaben enthalten. Dazu müssen Sie ein einfaches Datenfeld notieren, das den Namen des zu übertragenden Tabellenfeldes enthält. Der Name des Verweisfelds ist in der Feldliste in runde Klammern einzuschließen. Durch dieses Kommando kann der Schlüssel eines Datensatzes natürlich nicht geändert werden.

Mitunter sollen Einträge in mehreren Datensätzen geändert werden, wobei die Modifikation von einer Bedingung abhängig sein soll. Für diese Aufgabe steht eine Variante des MODIFY-Kommandos zur Verfügung:

```
MODIFY itab [ FROM record ] TRANSPORTING field_1 ... field_n
  WHERE condition.
```

Dieses Kommando ist für alle Tabellenarten verwendbar. Innerhalb der WHERE-Bedingung muss auf der linken Seite jedes einzelnen Vergleichsausdrucks ein Feld der internen Tabelle stehen.

Neben den oben beschriebenen Varianten des MODIFY-Kommandos existiert wiederum eine nicht generische Variante, die einen indizierten Zugriff auf Datensätze ermöglicht. Wie beim INSERT-Kommando ist dieses Kommando nicht für Hash-Tabellen einsetzbar:

```
MODIFY itab [ FROM record ] [ INDEX index ]
  [ TRANSPORTING field_1 ... field_n ].
```

Die Bereitstellung der Daten und die Indizierung des zu ändernden Datensatzes entsprechen dem INSERT-Kommando. Zusätzlich ist auch hier die Angabe der zu modifizierenden Felder möglich. Wie beim INSERT-Kommando kann bei der nicht generischen Variante der MODIFY-Anweisung auf die Angabe eines Index verzichtet werden, wenn das MODIFY-Kommando innerhalb einer LOOP-Schleife über die interne Tabelle steht. In diesem Fall wird der aktuelle Datensatz geändert.

Hier je ein Beispiel für den indizierten Zugriff und die Modifikation innerhalb einer LOOP-Schleife.

```
*..insert
i_standard-pgmid = 'R3TR'.
i_standard-object = 'PROG'.
i_standard-obj_name = 'modified'.
MODIFY i_standard INDEX 2.

LOOP AT i_standard WHERE obj_name NS 'modified'.
  CONCATENATE '<<' i_standard-obj_name '>>'
    INTO i_standard-obj_name
    SEPARATED BY SPACE.
  MODIFY i_standard.
ENDLOOP.
```

Im ersten Teil des Beispiels wird die Kopfzeile der Standard-Tabelle mit neuen Werten gefüllt. Anschließend wird der zweite Datensatz der Tabelle mit den neuen Werten überschrieben. Die nachfolgende LOOP-Schleife liest alle Datensätze außer dem gerade modifizierten. Sie wird zu diesem Zweck mit einer WHERE-Klausel ergänzt. Innerhalb der Schleife wird das Feld obj_name mit einem neuen Inhalt gefüllt. Anschließend schreibt die Anweisung MODIFY, hier ohne Index-Angabe, den geänderten Datensatz wieder zurück in die interne Tabelle.

Löschen von Datensätzen

Datensätze in internen Tabellen werden mit dem Kommando DELETE gelöscht. Auch hier existieren generische Varianten, die mit allen Tabellenarten zusammenarbeiten sowie andere Varianten, die nicht für Hash-Tabellen möglich sind. Zunächst wieder die generischen Varianten:

```
DELETE TABLE itab [ FROM record ].
```

Dieses Kommando entspricht völlig dem bereits bekannten Prinzip. Der Kopfzeile oder dem separaten Arbeitsbereich werden die Schlüsselfelder entnommen. Anschließend wird der Datensatz gesucht und gelöscht. Bei dieser Form des Kommandos wird stets der in der Tabellendefinition vorgegebene Schlüssel benutzt, dessen Aufbau muss somit beim Einsatz des Kommandos nicht berücksichtigt werden. Möglicherweise kann der zu löschende Datensatz aber auch

über andere Felder identifiziert werden. Für diesen Fall existiert ein zweites generisches Kommando, das die freizügige Definition des Schlüssels unabhängig von der Schlüsseldefinition im Tabellentyp ermöglicht:

```
DELETE TABLE itab
  WITH TABLE KEY { [ table_field | (pointer_field)] = pattern }.
```

Auf der linken Seite jedes Teilschlüssels muss ein Feld der internen Tabelle stehen. Dieses Feld kann direkt notiert oder dynamisch übergeben werden.

Ein weiteres Lösch-Kommando, dass bedingt für alle Tabellenarten einsetzbar ist, lautet:

```
DELETE itab WHERE condition [ FROM index1 ] [ TO index2 ].
```

Mit diesem Kommando werden alle Datensätze gelöscht, die der angegebenen Bedingung genügen. Die Vergleichsbedingungen entsprechen denen des `MODIFY-WHERE`-Kommandos. Für alle index-artigen Tabellen (also alle außer den Hash-Tabellen) kann über zwei zusätzliche Index-Angaben der zu löschende Bereich weiter eingeschränkt werden. Für Hash-Tabellen darf keiner der beiden Indizes benutzt werden.

Alle weiteren Lösch-Kommandos arbeiten nicht generisch und sind somit nicht für Hash-Tabellen einsetzbar. Das gezielte Löschen eines einzelnen Datensatzes ist über die Angabe eines Index möglich:

```
DELETE itab [ INDEX index ].
```

Der Index kann in einer Schleife über die bewusste Tabelle entfallen. In diesem Fall wird der aktuelle Datensatz gelöscht. Einen größeren Bereich können Sie durch das Kommando

```
DELETE itab [ FROM index1 ] [ TO index2 ].
```

löschen. Dabei muss mindestens eine der Index-Schranken angegeben werden. Fehlt der Wert für `FROM`, wird vom ersten Datensatz an gelöscht. Eine fehlende `TO`-Angabe dehnt den Löschbereich bis zum letzten Datensatz der Tabelle aus.

Speziell für Tabellen ohne eindeutigen Schlüssel wurde die folgende Anweisung geschaffen:

```
DELETE ADJACENT DUPLICATES FROM itab
  [ COMPARING [ fieldlist | ALL FIELDS ] ].
```

Sie erlaubt es, doppelte Einträge aus einer Tabelle zu entfernen. Dabei muss die Tabelle entsprechend dem anzuwendenden Schlüssel sortiert sein. Mittels der Option `COMPARING` können Sie vom Standard-Schlüssel abweichen. Verwendet wird dieser Parameter entweder mit einer Feldliste oder aber dem Bezeichner `ALL FIELDS`, mit dem Sie die Überprüfung aller Felder und nicht nur der eventuell definierten Schlüsselfelder erzwingen.

Das folgende Beispiel benutzt die DELETE TABLE-Anweisung, um einen Datensatz aus einer internen Tabelle zu löschen.

```
*..delete
DELETE TABLE i_sorted
  WITH TABLE KEY pgmid = 'R3TR'
                 object = 'PROG'
                 obj_name = 'YZ332020'.
```

Sequenzielles Lesen

Zum Lesen der Datensätze einer internen Tabelle stehen prinzipiell zwei Methoden zur Verfügung. Sie können die Tabelle in einer Schleife sequenziell durcharbeiten oder Sie können einen Einzelzugriff auf einen Datensatz ausführen.

Zum sequenziellen Lesen steht das Kommando LOOP zur Verfügung.

```
LOOP AT itab.
...
ENDLOOP.
```

Die Befehle der LOOP-Schleife werden für jeden Datensatz einmal durchlaufen. Die Daten werden dabei jeweils im Kopfsatz bereitgestellt. Mittels einiger Zusätze können Sie den Bereich der zu durchlaufenen Datensätze einschränken. Mittels FROM und TO geben Sie dazu einen Indexbereich an, mit WHERE können Sie eine Auswahl gemäß einer oder mehrerer Suchbegriffe vorgeben. Das LOOP-Kommando lautet dann:

```
LOOP AT itab FROM first TO last.
...
ENDLOOP.
```

oder

```
LOOP AT itab
WHERE condition.
...
ENDLOOP.
```

Falls Sie mit READ bzw. LOOP interne Tabellen bearbeiten, die keine eigene Kopfzeile besitzen, muss der Zugriff über einen ausdrücklich benannten Arbeitsbereich erfolgen. Dieser wird bei beiden Kommandos durch den Zusatz

```
... INTO workarea
```

angegeben.

Ein weiterer Zusatz steigert die Performance dadurch, dass Kopiervorgänge unterdrückt werden. Stattdessen erfolgt die Zuweisung eines Feldsymbols direkt zum Datensatz der internen Tabelle. Änderungen über das Feldsymbol finden somit sofort in der internen Tabelle statt, ein MODIFY ist nicht mehr erforderlich.

```
LOOP AT itab ASSIGNING <field_symbol>.
```

Da bei Tabellen der Arten SORTED TABLE und HASHED TABLE Modifikationen der Schlüsselfelder die Sortierreihenfolge der Tabelle stören könnten, sind sie verboten. Das Laufzeitsystem überwacht derartige Zugriffe und löst bei Verstößen einen Laufzeitfehler aus.

Nachfolgend einige zusätzliche Beispiele für die verschiedenen Varianten des LOOP-Kommandos.

```
*..loop
FIELD-SYMBOLS:
  <f_tadir> TYPE tr_tadir.

WRITE: / '  Standard-Tabelle'.
LOOP AT i_standard.
  WRITE: / i_standard-obj_name.
ENDLOOP.

WRITE: /,/ '  Sortierte Tabelle'.
LOOP AT i_sorted  ASSIGNING <f_tadir>.
  WRITE: / <f_tadir>-obj_name.
ENDLOOP.

WRITE: /,/ '  Hash-Tabelle'.
LOOP AT i_hashed INTO r_tadir.
  WRITE: / r_tadir-obj_name.
ENDLOOP.
```

Suchen von Datensätzen

Zur Suche eines einzelnen Datensatzes dient das Kommando READ. Es bietet zwei prinzipielle Varianten zur Verwendung an. Zunächst kann READ einen Datensatz anhand eines oder mehrerer Suchbegriffe in der internen Tabelle suchen. Außerdem ist der Zugriff auf einen Datensatz über eine Index-Angabe möglich. Wie bei den anderen Kommandos auch existieren für READ generische und nicht generische Varianten. Allerdings sind alle nicht generisch arbeitenden Kommandos von SAP ausdrücklich als veraltet gekennzeichnet. Im Interesse der Kompatibilität sollte sie daher nicht mehr benutzt werden. Die neuen generischen Varianten übernehmen die Aufgabe der veralteten Kommandos vollständig.

Alle Varianten dieses Kommandos füllen das Systemfeld SY-TABIX mit der Datensatznummer des entsprechenden Satzes, sofern einer gefunden wurde. Das Feld SY-SUBRC ist in diesem Fall mit dem Wert 0 belegt. Der Wert in SY-TABIX kann später in einem MODIFY-Kommando benutzt werden, um den Datensatz nach einer Änderung wieder in die interne Tabelle zurückzuschreiben.

Zur Suche eines Datensatzes ist dem Kommando READ der gültige Suchbegriff zu übergeben. Am einfachsten ist dies mit

```
READ TABLE itab FROM record.
```

Der Suchbegriff wird dem angegebenen Datensatz entnommen. Welche Felder berücksichtigt werden, hängt von der Definition des Schlüssels der jeweiligen Tabelle ab. Die Schlüsselfelder können auch einzeln mit Werten versorgt werden:

```
READ TABLE itab
  WITH TABLE KEY { [ table_field | (pointer_field)] = pattern }.
```

Dabei können die Namen der Schlüsselfelder der internen Tabelle auch dynamisch übergeben werden. Der entsprechende Feldname steht dann in einem Feld, dessen Name in der Anweisung in runde Klammern eingeschlossen zu notieren ist. Wichtig bei dieser Variante des Kommandos ist, dass auch hier der Zugriff über den definierten Schlüssel der Tabelle erfolgt. Dies ist bei der folgenden Variante des Kommandos nicht der Fall:

```
READ TABLE itab
  WITH KEY { [ table_field | (pointer_field)] = pattern }
  [ BINARY SEARCH ].
```

Hier können Sie beliebige Felder zur Suche benutzten. Beachten Sie bitte, dass der einzige Unterschied zur vorangegangenen Anweisung im Fehlen des Wortes TABLE zwischen WITH und KEY besteht. Die Suche in der Tabelle findet bei diesem Kommando normalerweise sequenziell statt. Falls eine Standard-Tabelle gemäß der Suchfelder sortiert ist, kann durch den Zusatz BINARY SEARCH eine schnelle binäre Suche erzwungen werden. Auch ohne diesen Zusatz wird in Tabellen der Tabellenart SORTED TABLE binär gesucht, wenn die angegebenen Suchfelder dem Schlüssel oder zumindest dem Beginn des Schlüssels entsprechen.

Der direkte Zugriff auf Datensätze über einen Index erfolgt mit

```
READ TABLE itab INDEX index.
```

Alle der genannten Kommandos können mit einigen Zusätzen aufgerufen werden. Zunächst kann der gelesene Datensatz mit

```
... INTO record
```

in einen separaten Arbeitsbereich geschrieben werden. Bei Tabellen ohne Kopfzeile ist dies sowieso unabdingbar. Ebenso wie beim sequenziellen Lesen können Sie auch ein Feldsymbol auf den gesuchten Datensatz setzen lassen:

```
... ASSIGNING <field_symbol>
```

Da dieses Feldsymbol direkt in den Datenbereich der internen Tabelle zeigt, wirken sich Modifikationen über das Feldsymbol sofort und ohne Notwendigkeit eines MODIFY auf den Inhalt der internen Tabelle aus. Die Schlüsselfelder von Ta-

bellen der Art SORTED TABLE oder HASHED TABLE dürfen natürlich nicht verändert werden.

Mitunter soll nur getestet werden, ob ein Satz mit einem bestimmten Schlüssel in einer internen Tabelle existiert oder nicht. In diesem Fall ist es unnötig, Daten aus der internen Tabelle in den Arbeitsbereich zu transportieren. Der Verzicht auf den Datentransport erspart etwas Abarbeitungszeit. Der Zusatz

```
... TRANSPORTING NO FIELDS
```

verhindert die Kopie von Daten in den Arbeitsbereich. Falls Sie nur ausgewählte Informationen aus der internen Tabelle benötigen, können Sie mit

```
... TRANSPORTING fieldlist
```

die zu transportierenden Felder angeben. Die Feldnamen können dabei auch dynamisch übergeben werden.

Das folgende Beispiel sucht einen Datensatz in einer internen Tabelle und gibt ihn aus, falls er gefunden wurde.

```
*..search
WRITE: /,/ '  Suche'.
READ TABLE i_sorted INTO r_tadir
  WITH TABLE KEY pgmid = 'R3TR'
                 object = 'PROG'
                 obj_name = 'YZ332010'.
IF sy-subrc = 0.
  WRITE: / r_tadir-pgmid, r_tadir-object, r_tadir-obj_name.
ENDIF.
```

Sortieren und Zusammenfassen

Neben den bisher beschriebenen Kommandos existieren noch einige weitere eigenständige Kommandos bzw. Varianten zu den bereits beschriebenen Kommandos, die Auswirkungen auf die Reihenfolge der Sätze in der internen Tabelle haben. Soll nach einem anderen Feld oder einer Liste von Feldern sortiert werden, muss das Kommando SORT zum Einsatz kommen. Die Angabe des Namens einer internen Tabelle ist obligatorisch. Mittels des Zusatzes BY können die Felder benannt werden, nach denen sortiert werden soll. Die Reihenfolge bestimmen die Parameter ASCENDING oder DESCENDING. Wird auf die Verwendung einer Feldliste mittels BY verzichtet, bildet das SORT-Kommando den Schlüssel aus allen alphanummerischen Feldern des Datensatzes. Bei der Sortierung nach dem Inhalt von Textfeldern kann der Zusatz AS TEXT Einfluss auf die Sortierung spezieller Zeichen (z.B. Umlaute) nehmen. In diesem Fall findet die Sortierung gemäß einer vorher eingestellten Textumgebung statt.

Beim Sortieren kann es vorkommen, dass mehrere Datensätze identische Sortierkriterien aufweisen. Das kann dann der Fall sein, wenn nacheinander nach mehreren Feldern sortiert werden soll. Die Grundform des Sortierkommandos kann nicht sicherstellen, dass sich diese Datensätze nach der Sortierung noch in genau derselben Reihenfolge befinden wie vorher. Falls die Reihenfolge erhalten bleiben soll, müssen Sie dies mit dem Zusatz `STABLE` explizit erzwingen.

Die komplette Syntax des `SORT`-Kommandos lautet somit:

```
SORT itab
  [ BY { field [ ASCENDING | DESCENDING ] [ AS TEXT ]  } ]
  [ STABLE ].
```

Hier ein Anwendungsbeispiel für das Sortierkommando:

```
*..sort
sort i_standard stable by pgmid object obj_name.
WRITE: /,/ '  Sortierte Standard-Tabelle'.
LOOP AT i_standard.
  WRITE: / i_standard-obj_name.
ENDLOOP.
```

Das nachträgliche Sortieren einer internen Tabelle ist relativ zeitaufwändig und wird daher nur selten durchgeführt. Unter R/3-Versionen ab 4.0 kann von vornherein eine sortierende Tabelle (`SORTED TABLE`) benutzt werden. In älteren Versionen ist es günstiger, die gewünschte Reihenfolge der Datensätze bereits beim Füllen einer interne Tabelle herzustellen. Dies gelingt z. B. durch die binäre Suche eines Datensatzes und Einfügen per Index, wie das nachfolgende Beispiel zeigt.

```
REPORT yz432160.
TABLES tadir.
DATA:
  itadir LIKE tadir OCCURS 50,
  ftadir LIKE tadir.

SELECT * FROM tadir
WHERE author = sy-uname.

* Index für neuen Datensatz ermitteln
* Durch READ soll nur SY-TABIX gesetzt werden,
* daher kein Transport von Feldinhalten notwendig
  READ TABLE itadir
    WITH KEY obj_name = tadir-obj_name
             object   = tadir-object
             pgmid    = tadir-pgmid
    BINARY SEARCH
    TRANSPORTING NO FIELDS.
```

```
* TADIR-Eintrag in interne Tabelle einfügen
  IF sy-subrc <> 0.
    INSERT tadir INTO itadir INDEX sy-tabix.
  ENDIF.
ENDSELECT.

* Testweises Auflisten des Tabelleninhaltes
LOOP AT itadir INTO ftadir.
  WRITE: / ftadir-obj_name, ftadir-object.
ENDLOOP.
```

In diesem Beispiel wird wiederum die Tabelle TADIR gelesen. Das SELECT-Kommando ermittelt alle vom aktuellen Anwender erzeugten Objekte. Für jedes Objekt wird in der internen Tabelle itadir nach einem identischen Objekt gesucht. Diese Suche wird immer erfolglos enden, da die drei zur Suche benutzten Felder Schlüsselfelder der Tabelle TADIR sind und somit immer nur ein Datensatz mit diesem Schlüssel existieren kann. Das Kommando READ stellt aber neben dem eigentlichen Suchergebnis (SY-SUBRC) im Feld SY-TABIX auch noch den Index des nächstgrößeren Datensatzes zur Verfügung. Dieser Wert kann nun benutzt werden, um den neuen Datensatz entsprechend der gewünschten Sortierreihenfolge in die interne Tabelle einzufügen.

Das APPEND-Kommando kennt einen Zusatz SORTED BY. Es sorgt aber nicht nur für eine definierte Reihenfolge beim Einfügen neuer Sätze, sondern weist zwei Besonderheiten auf:

Die interne Tabelle wird gemäß dem benannten Feld in absteigender (!) Reihenfolge sortiert.

Es wird nur die im OCCURS-Parameter angegebene Zahl von Sätzen in der Tabelle gehalten. In diesem speziellen Fall ist der OCCURS-Parameter also eine echte Größenangabe.

Diese Anweisung ist natürlich nur sinnvoll, wenn alle Datensätze auf diese Weise in die interne Tabelle geschrieben werden.

Etwas spezieller und dementsprechend selten verwendet ist eine Anweisung, die automatisch mehrere Datensätze zusammenfasst. Es handelt sich um die Anweisung COLLECT. Diese Anweisung überprüft, ob bereits ein Satz existiert, der in allen alphanummerischen Feldern denselben Inhalt aufweist wie der einzufügende. Ist dies der Fall, werden die nummerischen Felder der beiden Sätze summiert und der bereits existierende Satz dementsprechend aktualisiert. Existiert kein derartiger Satz, wird der neue Satz angefügt. Dazu ein einfaches Beispiel:

```
REPORT yz432170.
TABLES tadir.
DATA:
  BEGIN OF it_object OCCURS 0,
    object TYPE trobjtype,
    count TYPE I,
  END OF it_object.

SELECT * FROM tadir WHERE author = sy-uname.
  it_object-object = tadir-object.
  it_object-count = 1.
  COLLECT it_object.
ENDSELECT.

LOOP AT it_object.
  WRITE: / it_object-object, it_object-count.
ENDLOOP.
```

In einer `SELECT`-Schleife über die Tabelle `TADIR` werden all die Verwaltungseinträge gelesen, die sich auf Objekte des aktuellen Anwenders beziehen. Der Name der Objektgruppe wird in die Kopfzeile der internen Tabelle kopiert. Außerdem wird das Zählfeld auf den Wert 1 gesetzt. Die nachfolgende `COLLECT`-Anweisung fügt nun entweder einen neuen Datensatz an die interne Tabelle `IT_OBJECT` an oder aber addiert zu einem bereits vorhandenen Datensatz im Feld `COUNT` den Wert 1 hinzu. Nach Abschluss der Selektion steht im Feld `COUNT` jedes Datensatzes die Anzahl der entsprechenden Objekte.

Besonderheiten

Zum Kopieren von internen Tabellen steht eine besondere Schreibweise des Zuweisungsoperators zur Verfügung. Sie können mit

```
itab_body_target = itab_body_source.
```

den Inhalt einer internen Tabelle in eine andere kopieren. Allerdings reicht es nicht aus, nur den Tabellennamen zu notieren. Für interne Tabellen mit Kopfzeile steht in Zuweisungen der Tabellenname für die Kopfzeile und nicht den Datenbereich der Tabelle. Dieser ist durch zwei eckige Klammern direkt hinter dem Tabellennamen zu kennzeichnen. Für Tabellen ohne Kopfzeile ist die Angabe des Tabellennamens hingegen ausreichend.

Beispiele zu internen Tabellen

Die folgenden Beispiele demonstrieren einige weitere Details zur Anwendung der internen Tabellen. Zunächst ist es möglich, interne Tabellen anzulegen, die keine innere Struktur aufweisen. Diese Art von Tabelle entspricht einem eindimensionalen Vektor. Solche Tabellen werden oft bei ABAP-Kommandos mit dy-

namischen Parametern benutzt. Erzeugt werden interne Tabellen ohne Struktur, indem bei der Definition als Zeilentyp ein einfacher unstrukturierter Datentyp benutzt wird. Im folgenden Beispiel wird eine interne Tabelle definiert und später deklariert, die aus Zeichenfeldern der Länge 72 besteht.

```
REPORT yz432180.
TYPES:
  t_line(72),
  ti_flat
    TYPE STANDARD TABLE OF t_line
    INITIAL SIZE 0.

DATA:
  tadir TYPE tadir,
  i_condition TYPE ti_flat,
  line TYPE t_line.

line = 'OBJ_NAME LIKE ''YZ%'' OR'.
APPEND line TO i_condition.
line = 'AUTHOR = SY-UNAME'.
APPEND line TO i_condition.

SELECT * FROM tadir
  WHERE (i_condition).
  WRITE: / tadir-pgmid, tadir-object, tadir-obj_name.
ENDSELECT.
```

Da die Tabelle nicht strukturiert ist, kann der Zugriff auf die einzelnen Zeilen auch nur über den Namen der Tabelle (hier `i_condition`) erfolgen.

Das zweite Beispiel soll die Übergabe von internen Tabellen an Unterprogramme erläutern. Im Programm übernimmt das Unterprogramm `FILL_ITAB` das Füllen der internen Tabelle mit Testdaten. Diese ist deshalb als Parameter zu übergeben.

Die Option `TABLES` im `FORM`- bzw. `PERFORM`-Kommando kann nur die herkömmlichen internen Tabellen behandeln, nicht aber die neuen Tabellentypen. Diese neuen Tabellen müssen wie einfache Parameter über die Optionen `USING` oder `CHANGING` an Unterprogramme übergeben werden. Dazu muss eine Typisierung mit `TYPE` oder `LIKE` erfolgen. Bei dieser Form der Übergabe geht eine eventuell vorhandene Kopfzeile der internen Tabelle verloren. Innerhalb des Unterprogramms können Sie deshalb nur dann auf einzelne Felder der internen Tabelle zugreifen, wenn Sie dazu einen entsprechenden Arbeitsbereich deklarieren. Es folgt das Listing der Anwendung.

```
REPORT  yz432190.
TYPES:
  BEGIN OF tr_rec,
    number(6) TYPE n,
    price TYPE p DECIMALS 2,
  END OF tr_rec,

  ti_sort
    TYPE SORTED TABLE OF tr_rec
    WITH UNIQUE KEY number
    INITIAL SIZE 5.

DATA:
  i_rec TYPE ti_sort WITH HEADER LINE.

PERFORM fill1 USING i_rec[].
PERFORM fill2 USING i_rec[].
PERFORM fill3 USING i_rec[].

READ TABLE i_rec WITH TABLE KEY number = '000112'.
WRITE: / i_rec-number, i_rec-price.

WRITE: /.
LOOP AT i_rec.
  WRITE: / i_rec-number, i_rec-price.
ENDLOOP.


FORM fill1 USING pi LIKE i_rec[].
  DATA l LIKE LINE OF pi.
  l-number = '000112'.
  l-price = '3.95'.
  INSERT l INTO TABLE pi.
ENDFORM.

FORM fill2 USING pi TYPE SORTED TABLE.
  DATA l TYPE tr_rec.
  l-number = '300111'.
  l-price = '432.00'.

  INSERT l INTO TABLE pi.
ENDFORM.

FORM fill3 USING pi TYPE ti_sort.
  DATA l TYPE tr_rec.
  l-number = '000111'.
  l-price = '123.45'.
  INSERT l INTO TABLE pi.
ENDFORM.
```

Zu Beginn des Programms erfolgt wiederum die Definition eines Datensatztyps und einer Typanweisung für eine interne Tabelle. Vom Tabellentyp wird anschließend eine interne Tabelle angelegt, diesmal aber mit Kopfzeile. Ein separater Arbeitsbereich kann daher innerhalb des Hauptprogramms entfallen.

Die soeben erzeugte Tabelle wird als Parameter an drei Unterprogramme übergeben. Da die Tabelle `ti_rec` eine Kopfzeile besitzt und der `USING`-Zusatz zur Übergabe der Tabelle dient, müssen Sie durch nachgestellte eckige Klammern nach dem Tabellennamen kennzeichnen, dass sie die interne Tabelle und nicht die Kopfzeile als Parameter benutzen möchten. Dies ist nicht notwendig, wenn die interne Tabelle keine Kopfzeile besitzt.

Die drei Unterprogramme `FILL1` bis `FILL3` demonstrieren unterschiedliche Varianten der Typisierung. Im ersten Unterprogramm wird der Parameter `ps` durch eine Ähnlichkeitsbeziehung von der Tabelle `i_rec` abgeleitet. Auch hier ist wieder durch die beiden eckigen Klammern zu kennzeichnen, dass die Tabelle `i_rec` und nicht die Kopfzeile `i_rec` gemeint ist. Da sie mit `USING` nur ein Datenobjekt übergeben können, geht die Kopfzeile von `i_rec` verloren, nur der Rumpf der Tabelle bleibt erhalten. Sie können aber innerhalb des Unterprogramms durch einen Bezug auf eine Zeile des Parameters `ps` dessen Struktur benutzen, um einen Arbeitsbereich zu deklarieren.

Im zweiten Unterprogramm erfolgt die Typisierung durch einen generischen Tabellentyp. Diese Form der Typangabe legt nur fest, dass der übergebene Parameter eine Tabelle der Tabellenart `SORTED TABLE` ist. Sie können hier Tabellen mit beliebiger Struktur übergeben. Innerhalb des Unterprogramms wird wiederum ein Arbeitsbereich deklariert. Es besteht aber, abgesehen von der automatischen Überwachung der Länge von Tabellenzeile und Arbeitsbereich, keine Möglichkeit, die Kompatibilität von Tabelle und Arbeitsbereich zu prüfen.

Das dritte Unterprogramm beschreibt den Parameter durch Bezug auf die Typdeklaration `ti_sort`. Da hier schon im Datentyp festgelegt ist, dass es sich um eine Tabelle handelt, sind natürlich keine eckigen Klammern mehr notwendig.

Der übrige Teil des Programms ähnelt den anderen. Das generische `READ`-Kommando kann auch mit dieser Tabellenart umgehen. Da Standard-Tabellen weder eine Sortierung noch einen eindeutigen Schlüssel aufweisen, ist das `INSERT`-Kommando in diesem Beispiel korrekt. Anhand der Ausgabe in der Schleife erkennen Sie, dass die Datensätze in der Tabelle unsortiert vorliegen.

3.2.8 Steueranweisungen

Bisher wurden nur relativ einfache Kommandos zur Datenbearbeitung erwähnt. Diese werden sequenziell abgearbeitet, sie ermöglichen es nicht, den Programmablauf zu beeinflussen. ABAP besitzt natürlich auch einige Anweisungen, mit denen sich Programmverzweigungen oder zyklische Wiederholungen realisieren lassen. Das zu Grunde liegende Prinzip dürfte hinlänglich bekannt sein, so dass eine ausgedehnte Betrachtung unnötig erscheint.

IF

Eine IF-Anweisung verzweigt entsprechend dem Auswertungsergebnis einer komplexen Bedingung in einen Ja-(IF-) oder einen optionalen Nein-(ELSE-) Zweig. Für die Beschreibung der Bedingungen wurde ein eigener Abschnitt reserviert, da sie in einigen anderen Anweisungen ebenfalls benutzt werden. Die beiden alternativen Programmabschnitte werden durch die Anweisungen IF und ELSE bzw. ELSE und ENDIF gebildet.

Der ELSE-Zweig einer IF-Anweisung kann mit einer weiteren IF-Anweisung zu einem ELSEIF-Kommando verbunden werden. Beispiele:

```
IF a < b.
  WRITE / 'A ist kleiner B'.
ELSEIF a > b.
  WRITE / 'A ist größer B'.
ELSE.
  WRITE / 'A gleich B'.
ENDIF.
```

WHILE

Mittels der Anweisung WHILE werden Schleifen gebildet, die wiederholt durchlaufen werden, solange eine Bedingung erfüllt ist. Die vollständige Syntax der WHILE-Anweisung lautet:

```
WHILE condition.
  statements
ENDWHILE.
```

Das folgende Listing zeigt ein konkretes Beispiel für die WHILE-Anweisung:

```
WHILE n < 10.
  WRITE: / n.
  n = n + 1.
ENDWHILE.
```

CASE

Mittels der CASE-Anweisung wird der Inhalt eines Feldes mit mehreren Mustern verglichen. Jedem Muster werden Anweisungen zugeordnet, die bei Übereinstimmung von Feld und Muster ausgeführt werden. Ein optionaler Alternativzweig wird ausgeführt, wenn der Feldinhalt zu keinem Muster passt. Die Syntax lautet:

```
CASE field.
  WHEN pattern_1.
    statement_block_1.
  WHEN pattern_ 2.
```

```
    statement_ block_2.
  WHEN pattern _x.
    statement_block_x.
  WHEN OTHERS.
    alternativ statement_block.
ENDCASE.
```

In der CASE-Anweisung darf nur ein WHEN OTHERS-Zweig vorhanden sein. Falls ein Anweisungsblock für mehrere Werte von *field* ausgeführt werden soll, können Sie die Vergleichsmuster durch eine OR-Operator miteinander verbinden:

```
CASE field.
  WHEN pattern_1 { OR pattern_x }.
    statements.
```

Das nachfolgende Beispiel entstammt einer realen Anwendung.

```
CASE okcode.
  WHEN 'OPC+'.
    result = value1 + value2.
    op     = '+'.
    LEAVE TO SCREEN 200.
  WHEN 'OPC-'.
    result = value1 - value2.
    op     = '-'.
    LEAVE TO SCREEN 200.
  WHEN 'OPC*'.
    result = value1 * value2.
    op     = '*'.
    LEAVE TO SCREEN 200.
  WHEN 'OPC/'.
    result = value1 / value2.
    op     = '/'.
    LEAVE TO SCREEN 200.
  WHEN '/NEX'.
    LEAVE PROGRAM.
  WHEN OTHERS.
    MESSAGAE E137.
ENDCASE.
```

DO

Die DO-Anweisung existiert in zwei Varianten. Die erste Form bildet eine Endlosschleife, die explizit mit einer EXIT-, STOP- oder REJECT-Anweisung verlassen werden muss. Am häufigsten wird die EXIT-Anweisung benutzt.

```
DO.
  statements
ENDDO.
```

In der zweiten Form kann die DO-Schleife als einfacher Ersatz für eine FOR-Schleife dienen, die unter ABAP nicht existiert. Ein Parameter gibt an, wie oft die Schleife durchlaufen werden soll:

```
DO n TIMES.
  statements
ENDDO.
```

Innerhalb der DO-Schleife kann die Nummer des Schleifendurchlaufs der Systemvariablen SY-INDEX entnommen werden.

```
DO 6 TIMES.
  WRITE: / sy-index.
ENDDO.
```

EXIT, CONTINUE und CHECK

Es wurde bereits erwähnt, dass DO-Schleifen mit EXIT verlassen werden müssen. Innerhalb von Schleifen (auch bei LOOP, WHILE, SELECT) bewirkt dieses Kommando die Beendigung der Schleife. Wird dieses Kommando hingegen außerhalb von Schleifen, aber innerhalb von Modularisierungseinheiten (FORM, MODULE, FUNCTION, AT) aufgerufen, erfolgt der Aussprung aus dieser Einheit.

In Reports existieren oft keine untergeordneten Modularisierungseinheiten. In diesem Fall bewirkt ein EXIT das Beenden der Reporterstellung und das Ausgeben der Liste.

Der Einsatz der Anweisung CONTINUE ist nur innerhalb von Schleifen sinnvoll. Diese Anweisung bewirkt lediglich einen Abbruch des aktuellen Schleifendurchlaufs und den Sprung zum Schleifenanfang. Dort wird die Schleifenbedingung wieder ausgewertet und die Schleife mit dem nächsten Durchlauf fortgesetzt. Die Anweisung

```
CHECK condition.
```

stellt eine Abkürzung für die Anweisungsfolge

```
IF NOT condition.
  CONTINUE.
ENDIF.
```

dar. Diese Anweisung beendet also einen Schleifendurchlauf und beginnt den nächsten. Sie wird sehr häufig in SELECT-Schleifen innerhalb von Reports benutzt, um so genannte Selektionen auszuwerten.

```
DO 10 TIMES.
  ADD 1 TO i.
  WRITE: / i.
  CHECK i < 6.
  WRITE: '<'.
ENDDO.
```

Bedingungen

In den Kommandos zur Ablaufsteuerung müssen Sie Bedingungen , auch logische Ausdrücke genannt, angeben. Diese Ausdrücke lassen sich in verschiedene Gruppen einteilen. Einige der Ausdrücke sind nur für spezielle Datentypen zugelassen.

Die einfachste Variante zum Bilden von Bedingungen besteht im Vergleich eines Feldes gegen einen Wert oder ein anderes Feld. Mit den dazu verwendeten Operatoren (siehe Tabelle 3.14) können nicht nur nummerische Werte verglichen werden, sondern Felder aller Datentypen. Auch der Vergleich von Werten unterschiedlichen Typs ist möglich, wobei natürlich Typumwandlungen ablaufen, deren Wirkung beim Formulieren des Ausdrucks bedacht werden muss!

Operator	Alternative Schreibweise	Beschreibung
>	GT	Größer
>= =>	GE	Größer oder gleich
=	EQ	Gleich
<= =<	LE	Kleiner oder gleich
<	LT	Kleiner
<>, ><	NE	Ungleich

Tabelle 3.14
Vergleichsoperatoren

Die Grundform der Programmiersprache ABAP ist tolerant gegenüber der Schreibweise einiger Vergleichsoperatoren. So können Sie beispielsweise sowohl >= als auch => verwenden. Innerhalb der objektorientierten Variante ABAP-OO sind allerdings nur noch die Varianten >=, <= und <> zugelassen, die alternative Schreibweise wird als Syntaxfehler zurückgewiesen.

Die verschiedenen Datentypen werden bei Vergleichen individuell behandelt. Der Vergleich von Zeichenketten erfolgt anhand der lexikographischen Reihen-

folge. Die beiden Zeichenketten werden zeichenweise miteinander verglichen, das erste unterschiedliche Zeichen entscheidet über den Ausgang des Vergleiches. Bei Datumsfeldern ist das jüngere Datum das größere, während bei Zeitangaben die spätere Zeitangabe der größere Wert ist.

Für Zeichenketten stehen einige zusätzliche Vergleichsoperatoren zur Verfügung. Tabelle 3.15 zeigt diese Operatoren. Sie werden in der Form

```
string1 operator string2
```

verwendet. Diese Reihenfolge der Operanden wird in der Tabelle 3.15 zur Beschreibung der Funktion vorausgesetzt.

Operator	Beschreibung
CO	Contains only String1 enthält nur Zeichen aus String2.
CN	contains not only String1 enthält nicht nur Zeichen aus String2.
CA	Contains any String1 enthält mindestens 1 Zeichen aus String2.
NA	not any String1 enthält kein Zeichen aus String2.
CS	contains string String1 enthält String2.
NS	no string S1 enthält S2 nicht.
CP	contains pattern S1 entspricht Muster in S2.
NP	not pattern S1 entspricht nicht Muster in S2.

Tabelle 3.15
Operatoren für Stringvergleiche

Zu den Operatoren CP und NP ist eine Bemerkung hinsichtlich des Musters erforderlich. Im Gegensatz zu anderen Anweisungen unter ABAP sind hier das Zeichen "*" für ein beliebiges Muster und das Zeichen "+" für ein beliebiges Zeichen zu verwenden.

Einige andere, teilweise schon im Zusammenhang mit der WHERE-Klausel beschriebene Möglichkeiten zeigt die Tabelle 3.16.

Funktion	Beschreibung
`[NOT] BETWEEN ... AND`	Bereichsangabe.
`IS [NOT] INITIAL`	Feld gleich Initialwert.
`IS [NOT] ASSIGNED`	Feldsymbol wurde zugewiesen.
`IS [NOT] REQUESTED`	Der formale Parameter eines Funktionsbausteins wird im rufenden Programm mit einem Wert versorgt.
`[NOT] IN`	Enthalten in Selektionstabelle.
`IS [NOT] SUPPLIED`	Parameter eines Funktionsbausteins mit Wert versorgt.
`IS [NOT] BOUND`	Objektreferenz mit gültigem Wert gefüllt.

Tabelle 3.16
Weitere Vergleiche

Darüber hinaus besteht für hexadezimale Felder der Länge 1 die Möglichkeit eines bitweisen Vergleichs. Diese Vergleiche unterscheiden sich teilweise von den herkömmlichen Bitvergleichen anderer Programmiersprachen. Der Einsatz der Operatoren erfolgt in der Form

```
f1 operator f2
```

wobei `f1` das zu überprüfende Feld und `f2` die Bitmaske ist. Tabelle 3.17 zeigt die drei möglichen Bit-Operatoren.

Operator	Beschreibung
O (One)	Wahr, wenn f1 an den Positionen mit 1 gefüllt ist, in denen in f2 ebenfalls eine 1 steht.
Z (Zero)	Wahr, wenn f1 an den Positionen mit 0 gefüllt ist, in denen in f2 eine 0 steht.
M (Mixed)	Wahr, wenn f1 in mindestens einer der in f2 gekennzeichneten Positionen mit 1 und in mindestens einer Position mit 0 gefüllt ist.

Tabelle 3.17
Bit-Operatoren

Neu in der Version 6 sind Vergleichsoperatoren für Bytefolgen, also Felder des Typs X oder XSTRING. Diese Vergleiche ähneln den Stringvergleichen. Tabelle 3.18 zeigt die entsprechenden Möglichkeiten.

Operator	Ausdruck f1 <op> f2 ist wahr, wenn:
BYTE-CO	f1 nur Bytes aus der Bytemenge f2 enthält.
BYTE-CN	f1 nicht nur Bytes aus f2 enthält.
BYTE-CA	f1 mindestens ein Byte aus F2 enthält.
BYTE-NA	f1 kein einziges Byte aus f2 enthält.
BYTE-CS	f1 die Bytefolge f2 enthält.
BYTE-NS	f1 die Bytefolge f2 nicht enthält.

Tabelle 3.18
Operatoren für Bytefolgen

Das folgende Programm zeigt einige der Einsatzmöglichkeiten der IF-Anweisung.

```
REPORT yz432200.
DATA:
  s1(10) VALUE 'ABCDEFGHIJ',
  s2(3)  VALUE 'XFH',
  s3(5)  VALUE '*DEF*'.

IF s1 CA s2.
  WRITE: / s1, 'contains any', s2, sy-fdpos.
ELSE.
  WRITE: / s1, 'contains not any', s2.
ENDIF.

IF s1 CP s3.
  WRITE: / s1, 'contains pattern', s3, sy-fdpos.
ELSE.
  WRITE: / s1, 'contains not', s3.
ENDIF.

IF s1 CO s2.
  WRITE: / s1, 'CO', s2, sy-fdpos.
ELSE.
  WRITE: / s1, 'contains not only', s2.
ENDIF.
```

3.2.9 Inhalt von Dictionary-Tabellen bearbeiten

Die eigentliche Aufgabe von Reports ist die Auswertung von Tabellen, die nun endlich im Mittelpunkt des Interesses stehen sollen. Die bisher vermittelten Kenntnisse bilden die unbedingt erforderliche Grundlage für die Arbeit mit Datenbanktabellen.

Prinzipieller Zugriff auf Daten

Mit der Version 6.x ergeben sich einige Änderungen im Umgang mit Datenbanktabellen. Der Zugriff auf die Daten erfolgt, ähnlich wie bei internen Tabellen, über einen Arbeitsbereich. Dabei kann unterschieden werden zwischen einem Arbeitsbereich, der denselben Namen trägt wie die Tabelle und einem Arbeitsbereich, der einen beliebigen Namen tragen darf. Im Falle der ersten Variante wird die Verbindung zwischen interner Tabelle und Arbeitsbereich durch das System automatisch unter Ausnutzung der Namensgleichheit hergestellt. Bei der zweiten Variante muss der Arbeitsbereich in den jeweiligen Kommandos explizit angegeben werden. Für Datenbanktabellen kann ebenfalls ein Arbeitsbereich verwendet werden, der denselben Namen trägt wie die Datenbanktabelle. Dieser Arbeitsbereich kann ab der Version 6.x aber auf zwei unterschiedliche Arten erzeugt werden. Seit jeher steht das Kommando `TABLES` zur Verfügung, das eine Datenbanktabelle in einem Programm bekannt macht und gleichzeitig den Arbeitsbereich erzeugt. Alternativ dazu kann nun auch das Kommando `DATA` mit Typbezug auf eine Datenbanktabelle benutzt werden, um einen Arbeitsbereich zu erzeugen. Beide Varianten werden nachfolgend genauer beschrieben. Es ist nicht notwendig, die in einer Anwendung benutzten Tabellen explizit bekannt zu machen. Sie können problemlos benutzt werden, sofern ein passender Arbeitsbereich (egal mit welchem Namen) zur Verfügung steht.

Die TABLES-Anweisung

Wie alle anderen Datenobjekte müssen auch Tabellen vor ihrer Benutzung in Programmen deklariert werden. Dies erfolgt mit der Anweisung

```
TABLES table.
```

oder in der Doppelpunktvariante mit

```
TABLES: table_1, table_2, ..., table_n.
```

Voraussetzung für die Deklaration mit `TABLES` ist, dass die Tabelle aktiv im Data Dictionary enthalten ist. Ansonsten führt bereits der vor Abarbeitung eines Programms erfolgende Syntaxcheck zu einem Fehler. Neben Tabellen im engeren Sinn kann das Lesen von Daten auch über Views erfolgen. Die Pflege von Daten über Views ist aber nur mit Einschränkungen möglich.

Im Zusammenhang mit ABAP-OO ist die TABLES-Anweisung nicht zulässig, sie wird als Syntaxfehler zurückgewiesen.

Die TABLES-Anweisung arbeitet global. Das bedeutet, dass der Tabellenarbeitsbereich auch dann global verfügbar ist, wenn die TABLES-Anweisung in einem Unterprogramm oder einer anderen Modularisierungseinheit notiert wird.

Das System speichert intern einige Verwaltungsinformationen, beispielsweise die Position und den Inhalt des aktuellen Datensatzes, mit Bezug auf den Namen der Tabelle ab. Diese Form der Verwaltung macht es zunächst unmöglich, auf mehrere Datensätze einer Tabelle gleichzeitig zuzugreifen. Da derartige Zugriffe in der Praxis aber sehr häufig notwendig sind, kann für jede Datenbanktabelle ein zweiter Arbeitsbereich mit separaten Verwaltungsinformationen erzeugt werden. Dazu wird für jede Tabelle ein Pseudo-Name eingeführt, der aus dem Tabellennamen mit einem führenden Stern besteht. Dieser zweite Arbeitsbereich kann völlig unabhängig zur ersten Form der Deklaration benutzt werden. Ein Beispiel für eine derartige Deklaration ist

```
TABLES: TADIR, *TADIR.
```

Wie eingangs erläutert, ist die TABLES-Anweisung inzwischen nicht die einzige Anweisung, mit deren Hilfe der Zugriff auf Datenbanktabellen möglich ist. Der Einsatz ist nur noch erforderlich, wenn der direkte Bezug von Tabellenfeldern zu Dynpro-Feldern hergestellt werden soll. In allen anderen Fällen kann entweder die DATA- oder die NODES-Anweisung benutzt werden.

Deklaration des Arbeitsbereichs mit DATA

Eine Variante der DATA-Anweisung ermöglicht es, einen Arbeitsbereich für Dictionary-Tabellen anzulegen. Falls dieser Arbeitsbereich denselben Namen trägt wie die Dictionary-Tabelle, stellt das System die Verbindung zwischen Tabelle und Arbeitsbereich automatisch her. Wenn die Namen voneinander abweichen, müssen Sie den Arbeitsbereich in den diversen Kommandos explizit angeben.

Die Syntax ist sehr einfach:

```
DATA workarea TYPE dictionary_table.
```

Da die TABLES-Anweisung im Zusammenhang mit ABAP-OO nicht mehr benutzt werden kann, sollten Neuentwicklungen einen Tabellenarbeitsbereich immer mit DATA definieren.

Zugriff auf Tabellen

Nach der Deklaration einer Tabelle kann deren Inhalt mit verschiedenen Kommandos bearbeitet werden. Dabei muss der Inhalt eines Datensatzes von der Tabelle zu programminternen Feldern oder in umgekehrter Richtung von Feldern in die Tabelle übertragen werden. Das Dateikonzept herkömmlicher Program-

miersprachen benutzt dazu explizite Lese- und Schreibanweisungen. Dabei müssen sowohl Dateideskriptoren als auch Variablen zur Aufnahme der Werte deklariert und in den Anweisungen benutzt werden. Dateien sind zu öffnen und zu schließen, bei Lese- und Schreibaktionen sind Quelle und Ziel anzugeben.

Unter ABAP gestaltet sich der Zugriff auf Tabellen wesentlich einfacher. Die Deklaration mit `TABLES` macht nicht nur die Tabelle im Programm bekannt. Vielmehr wird zusätzlich eine Feldleiste mit dem Namen der Tabelle angelegt. Sie besitzt automatisch die Struktur, die im Data Dictionary für die jeweilige Tabelle festgelegt wurde. Auf sie kann mit den bereits beschriebenen Anweisungen für Datenfelder und Feldleisten zugegriffen werden. Eine derartige Feldleiste nimmt immer nur einen Datensatz aus der Tabelle auf. Im SAP-Sprachgebrauch wird eine solche, einer Tabelle zugeordnete Feldleiste auch *Kopfzeile*, *Arbeitsbereich* oder *Tabellenarbeitsbereich* genannt.

Durch die Deklaration einer Tabelle mit `TABLES` existieren also zwei unterschiedliche, aber gleichnamige Objekte. Diese Tatsache gerät mitunter in Vergessenheit, da die relativ einfachen ABAP-Anweisungen auch bei intuitiver Anwendung fast immer das gewünschte Ergebnis erzielen. Allerdings gibt es Anweisungen, mit welchen die Datenbanktabelle (und zwar nur diese) bearbeitet wird, andere wiederum beeinflussen nur die Kopfzeile, ohne Änderungen auf der Datenbanktabelle zu bewirken. Der Inhalt des Arbeitsbereiches wird automatisch aktualisiert, wenn ein Satz der Datenbanktabelle gelesen wird. Änderungen im Arbeitsbereich, z.B. per Zuweisung, werden aber nicht automatisch in die Datenbanktabelle übertragen. Dies muss durch entsprechende Anweisungen erfolgen.

Datenbankabfragen mit SELECT

Die unter R/3 verwendeten Datenbanksysteme arbeiten SQL-basiert. Für den Zugriff auf die Datenbestände stellt ABAP einige Anweisungen bereit, die unter dem Oberbegriff *Open-SQL* eingeordnet werden. Diese Kommandos orientieren sich an den zumeist gleichnamigen Standard-SQL-Kommandos, wobei aber zum Teil erhebliche Unterschiede bezüglich der implementierten Funktionalität bestehen. Der Funktionsumfang der Open-SQL-Kommandos hängt in hohem Maße vom Releasestand des SAP-Systems ab. Die volle Leistungsfähigkeit des SQL-Standards wird nicht erreicht, da die gewünschte Systemunabhängigkeit die Beschränkung auf den kleinsten gemeinsamen Nenner aller unter R/3 verwendbaren Datenbanksysteme erzwingt.

Das Lesen von Datensätzen läuft auf die Ausführung einer Datenbankabfrage hinaus. Diese wird durch die `SELECT`-Anweisung ausgeführt. Bis zum Release 2.2 benutzt die `SELECT`-Anweisung einige Klauseln aus dem Standard-SQL-Befehlssatz, ohne aber auch nur annähernd deren Funktionalität zu bieten. Ab Release 3.0 werden diese SQL-Klauseln wesentlich besser unterstützt. Das aktuelle Release brachte nochmals eine Erweiterung der Funktionalität. Die `SELECT`-Anweisung besteht aus mehreren – teilweise optionalen – Klauseln. Die Syntax

dieser Anweisung ist sehr komplex, so dass es unmöglich ist, das gesamte Kommando in einer einzigen Syntaxbeschreibung übersichtlich darzustellen. Als Ausgangspunkt soll zunächst folgende vereinfachte Übersicht dienen:

```
SELECT what
  [INTO target]
   FROM source
  [WHERE condition]
  [GROUP BY fieldlist
    [HAVING condition]]
  [ORDER BY fieldlist]
[ENDSELECT].
```

Eine syntaktisch korrekte SELECT-Anweisung besteht mindestens aus der SELECT- und der FROM-Klausel. Die SELECT-Klausel definiert die zu lesenden Werte genauer. Sie legt beispielsweise die Menge der zu lesenden Felder fest. Dabei können die Ergebnisfelder auch Resultat von Funktionen sein, die auf die gelesenen Einzelwerte angewendet werden.

In der FROM-Klausel geben Sie an, welchen Tabellen gelesen werden soll. Während bis zur Version 3.x nur eine einzige Tabelle ausgewertet werden konnte, besteht ab Version 4.x die Möglichkeit, durch Joins mehrere Tabellen miteinander zu verknüpfen.

Im praktischen Einsatz wird noch die WHERE-Klausel erforderlich, um Suchbedingungen anzugeben. Ohne WHERE-Klausel ist die SELECT-Anweisung zwar syntaktisch korrekt, liefert dann aber alle Datensätze der Tabelle zurück.

Die INTO-Klausel wird eingesetzt, um die Ergebnismenge in anderen Feldern als der Kopfzeile der Tabelle abzulegen. In einigen Fällen ist sie optional, in anderen zwingend erforderlich. Das hängt vom Aufbau der SELECT-Klausel und dem Aufbau der anderen Klauseln ab.

Mit der Klausel ORDER BY können Sie eine Sortierung der Ergebnismenge erzwingen.

Durch die GROUP BY-Klausel wird es möglich, Gruppen von Datensätzen mit gleichen Eigenschaften zu einer Zeile der Ergebnismenge zu verdichten. Beim Einsatz von GROUP BY kann die HAVING-Klausel Bedingungen enthalten, die bei der Gruppenbildung berücksichtigt werden.

Die SELECT-Anweisung kann sowohl eine einfache als auch eine Schleifenanweisung sein. Dies hängt von der Art der Bearbeitung der selektierten Datensätze ab, die Sie durch verschiedene Optionen zu diversen Klauseln näher bestimmen können. Wenn durch verschiedene Optionen (z.B. SELECT SINGLE oder COUNT(*)) die SELECT-Anweisung nur einen einzigen Datensatz zurückliefern kann oder die gesamte Ergebnismenge mit INTO in eine interne Tabelle umgeleitet wird, müssen Sie die SELECT-Anweisung als einfache Anweisung ein-

setzen, in allen anderen Fällen als Schleifenanweisung, die durch ENDSELECT abzuschließen ist.

Die einzelnen Klauseln der SELECT-Anweisung werden in den folgenden Unterabschnitten detailliert beschrieben. Zunächst jedoch ein einfaches Beispiel. Beachten Sie bitte, dass die Tabelle TADIR eine äußerst wichtige Systemtabelle ist, die von Ihnen zwar gelesen, keinesfalls aber verändert werden darf. Diese Tabelle ist das „Inhaltsverzeichnis" der Entwicklungsumgebung. Sie enthält z.B. für jedes Programm einen Eintrag. Genau diese Einträge (Programme des aktuellen Benutzers) sollen im ersten Beispiel ermittelt werden.

```
PROGRAM yz432210.
DATA:
  tadir TYPE tadir.

SELECT * FROM tadir
  WHERE author = sy-uname
  AND object = 'PROG'.
  WRITE: / tadir-pgmid,
           tadir-object,
           tadir-obj_name,
           tadir-author.
ENDSELECT.
```

Im obigen Beispiel werden zwei mit AND verknüpfte Vergleichsausdrücke benutzt. Beide sind eigentlich selbst erklärend, geprüft wird einmal auf die Übereinstimmung des Eigentümers des Programms mit dem aktuell angemeldeten Anwender (Systemvariable SY-UNAME), zum anderen werden über das Feld OBJECT alle Programme ausgefiltert. Beachten Sie hier bitte die Verwendung der Anweisung DATA. Sie deklariert den Arbeitsbereich für die Tabelle TADIR. In der SELECT-Anweisung wird dieser Arbeitsbereich automatisch benutzt, um Daten aus der Tabelle in das Programm zu übernehmen.

Wie alle anderen Kommandos der Open-SQL-Gruppe setzt SELECT die Systemvariable SY-SUBRC auf einen definierten Wert. Sie signalisiert damit, ob das Kommando erfolgreich war oder missglückte. Tabelle 3.19 zeigt mögliche Werte und deren Bedeutung.

SY-SUBRC	Bedeutung
0	Suche erfolgreich, mindestens ein Satz gefunden.
4	Kein Satz gefunden.
8	Nur bei SELECT SINGLE. Der Suchbegriff war mehrdeutig, ein beliebiger Satz der Ergebnismenge wird zurückgeliefert.

Tabelle 3.19
Werte für SY-SUBRC nach SELCET

Von der SELECT-Anweisung werden über SY-SUBRC weitere Systemvariablen gesetzt. Sehr häufig wird in Programmen SY-DBCNT ausgewertet. Dieses Feld enthält die Zahl der gefundenen Datensätze.

Die SELECT-Klausel

Die SELECT-Klausel wird durch das Schlüsselwort SELECT eingeleitet. Sie legt durch eine Feldliste fest, welche Tabellenfelder in der zurückgelieferten Datenmenge enthalten sein sollen. Diese Feldliste besteht aus der Auflistung einzelner Felder oder einem Stern als Symbol für alle Tabellenfelder. Falls der Stern benutzt wird, können die selektierten Datensätze in der Kopfzeile der Tabelle abgelegt werden. Mit der Klausel INTO können Sie die Ergebnismenge in andere Ziele (Feldleiste, interne Tabelle) umleiten. Die INTO-Klausel ist auf jeden Fall erforderlich, wenn Sie in der SELCET-Klausel eine Feldauswahl angeben oder Aggregatausdrücke benutzen. Programm YZ432220 zeigt ein Beispiel für die Anwendung einer Feldauswahl. Es erfüllt die gleiche Aufgabe wie Programm YZ432210.

```
PROGRAM yz432220.
DATA:
  pid TYPE tadir-pgmid,
  obj TYPE tadir-object,
  nam TYPE tadir-obj_name,
  aut TYPE tadir-author.

SELECT pgmid object obj_name author
  INTO (pid, obj, nam, aut)
  FROM tadir
  WHERE author = sy-uname
    AND object = 'PROG'.

  WRITE: / pid, obj, nam, aut.
ENDSELECT.
```

Dieses Programm zeigt den Unterschied bezüglich der Verwendung von Dictionary-Tabellen im Vergleich zu älteren R/3-Versionen besonders deutlich. Es reicht in der Version 6.x aus, die 4 Felder zu deklarieren, die in der INTO-Klausel benutzt werden. In älteren Versionen hätte zusätzlich noch die Tabelle TADIR mittels der TABLES-Anweisung bekannt gemacht werden müssen, damit auf sie zugegriffen werden kann, auch wenn der Arbeitsbereich TADIR nicht benutzt worden wäre.

Einige weitere Optionen der SELECT-Klausel erzwingen eine Verarbeitung der selektierten Sätze direkt durch die SELECT-Anweisung. Dabei handelt es sich üblicherweise um eine Verdichtung von mehreren Datensätzen nach unterschiedlichen Kriterien.

Eine dieser Verarbeitungsmöglichkeiten besteht im Zusammenfassen identischer Ausgabezeilen durch den Zusatz DISTINCT. Identische Ausgabezeilen können auftreten, wenn durch die SELECT-Anweisung nicht alle Felder der Tabelle gelesen, sondern mittels einer Feldliste einzelne Felder ausgewählt werden. Fehlen dabei ein oder mehrere Schlüsselfelder der Tabelle, können durchaus Ausgabezeilen mit identischem Inhalt erscheinen. Das folgende Programm listet alle vom aktuellen Anwender erzeugten Objekttypen, nicht aber die einzelnen Objekte auf. Da DISTINCT nur im Zusammenhang mit einer Feldliste in der SELECT-Klausel sinnvoll eingesetzt werden kann, wird zwangsläufig auch der Einsatz der INTO-Klausel notwendig, um die Inhalte der selektierten Tabellenfelder an programminterne Felder zu übergeben.

```
REPORT yz432230.
DATA:
  obj TYPE tadir-object.

SELECT DISTINCT object
  INTO obj
  FROM tadir
  WHERE author = sy-uname.

  WRITE: / obj.
ENDSELECT.
```

Eine andere Möglichkeit zur Auswertung der selektierten Datensätze besteht im Einsatz von Aggregatfunktionen. Eine Aggregatfunktion wertet den Inhalt eines (meist nummerischen) Tabellenfeldes in allen selektierten Datensätzen aus und stellt als Ergebnis einen einzigen Wert bereit. Tabelle 3.20 zeigt die verfügbaren Aggregatfunktionen.

Aggregatfunktion	Beschreibung
AVG (*Feld*)	Durchschnitt.
COUNT(DISTINCT *Feld*)	Anzahl der verschiedenen Werte.
COUNT (*)	Anzahl der selektierten Datensätze.
MAX (*Feld*)	Größter Wert.
MIN (*Feld*)	Kleinster Wert.
SUM (*Feld*)	Summe der Feldinhalte.

Tabelle 3.20
Aggregatfunktionen

Mit dem folgenden Programm wird die Zahl der Objekte, die dem aktuellen Anwender gehören, mit Hilfe einer Aggregatfunktion ermittelt.

```
REPORT yz432240.
DATA:
  anz TYPE i.

SELECT COUNT( * )
  INTO anz
  FROM tadir
  WHERE author = sy-uname.

WRITE: / anz.
```

Aggregatfunktionen stellen eine spezielle Feldauswahl dar und erfordern damit ebenfalls den Einsatz der `INTO`-Klausel, um das Ergebnis der Funktion an ein programminternes Datenfeld zu übergeben. In der Praxis werden Aggregatfunktionen meist in Zusammenhang mit der `GROUP BY`-Klausel eingesetzt.

Da Tabellen im SAP-System einen eindeutigen Schlüssel besitzen, kann durch entsprechende Formulierung der `WHERE`-Klausel gezielt ein einzelner Satz angesprochen werden. In diesem Fall ist eine `SELECT-ENDSELECT`-Schleife unnötig. Mit dem zusätzlichen Schlüsselwort `SINGLE` in der `SELECT`-Anweisung wird diese veranlasst, nur einen einzelnen Satz zu suchen und zurückzuliefern.

```
SELECT SINGLE * ...
```

Die `WHERE`-Klausel muss in diesem Fall den vollständigen Schlüssel der Tabelle testen. In den einzelnen Vergleichen ist nur die Prüfung auf Gleichheit zugelassen, somit kann (bei eindeutigem Schlüssel) nur ein Satz gefunden werden. Bei dieser Variante der `SELECT`-Klausel kann mit dem Zusatz

```
... FOR UPDATE
```

eine temporäre Datensatzsperre ausgelöst werden, die den selektierten Datensatz gegen Überschreiben durch Dritte schützt.

Die FROM-Klausel

Die auszuwertende Tabelle ist in der obligatorischen `FROM`-Klausel anzugeben. Dabei kann der Name als Zeichenkette oder in einem Datenfeld übergeben werden. Im letzteren Fall müssen Sie den Feldnamen in runde Klammern einschließen:

```
... FROM dbtab [AS alias]
... FROM (dbtabname)
```

Einen Aliasnamen für Tabellen vergeben Sie nur dann, wenn Sie Joins benutzen und dort Feldnamen eindeutig bezeichnen müssen. In solchen Fällen erspart ein kurzer Aliasname Schreibarbeit und verbessert die Übersicht. Den Joins ist we-

gen der Komplexität des Themas ein eigener Unterabschnitt gewidmet. An dieser Stelle sollen nur noch einige Optionen zur FROM-Klausel beschrieben werden.

Das gesamte R/3-System ist als Client-Server-Anwendung realisiert. Dies bedeutet in der Praxis, dass die Verbindung zur Datenbank über mehrere hierarchisch verknüpfte Rechner erfolgen kann. Tabellendaten werden möglicherweise im Speicher so genannter Anwendungsserver gepuffert, falls dies für die jeweilige Tabelle zugelassen ist. Es erfolgt also nicht in jedem Fall der direkte Zugriff auf die Datenbank. Dies kann bei Datenbeständen, die häufig geändert werden, oder bei besonders kritischen Abfragen zu Differenzen zwischen dem Ergebnis der SELECT-Anweisung und dem realen Datenbestand führen. Mittels des Zusatzes

```
... BYPASSING BUFFER
```

zur FROM-Klausel kann das direkte Lesen von der Datenbank unter Umgehung aller Puffer erzwungen werden. Da der direkte Datenbankzugriff natürlich zu Lasten der Performance geht, sollte der Einsatz dieser Anweisung nur in begründeten Fällen erfolgen.

In Sonderfällen ist es nicht erforderlich, wirklich alle dem Selektionskriterium genügenden Datensätze zurückzuliefern. In solchen Fällen kann mit dem Zusatz

```
... UP TO n ROWS
```

die maximale Zahl der zu liefernden Datensätze bestimmt werden. Mit dem Zusatz

```
... CLIENT SPECIFIED
```

kann das automatische Mandantenhandling des Systems abgeschaltet werden. Dieser Zusatz ist auch für andere Open-SQL-Kommandos möglich. Bei Einsatz dieser Option muss das Mandantenfeld in mandantenabhängigen Tabellen ebenso wie andere Schlüsselfelder durch die Anwendung mit einem gültigen Wert versorgt werden. Sie ermöglicht es, mandantenabhängige Tabellen mandantenübergreifend zu bearbeiten! Derartige Tabellenzugriffe sind nur in Sonderfällen von Interesse und haben in gängigen Anwendungsprogrammen nichts verloren.

Die INTO-Klausel

Die INTO-Klausel ist immer dann erforderlich, wenn die Ergebnismenge nicht in der Kopfzeile der Tabelle abgelegt werden soll oder kann. Sofern die SELECT-Klausel den gesamten Datensatz liefert (Feldliste *), ist die INTO-Klausel optional. Je nach Umfang der Ergebnismenge kann nach INTO eine Feldleiste oder eine interne Tabelle angegeben werden, um die selektierten Datensätze aufzunehmen.

```
... INTO record
```

Diese Form der INTO-Klausel schreibt den gelesenen Datensatz in einen alternativen Arbeitsbereich record und nicht in die Kopfzeile der Datenbanktabelle. Die eigentliche Funktion der SELECT-Anweisung ist davon nicht betroffen, insbesondere wird eine eventuell vorhandene ENDSELECT-Anweisung nicht überflüssig. Die Struktur von record muss natürlich der Struktur der Datenbanktabelle entsprechen, damit die gelesenen Daten korrekt ausgewertet werden können. Falls die Struktur von record von der Datenbanktabelle abweicht, bietet

```
... INTO CORRESPONDING FIELDS OF record
```

die Möglichkeit, nur Felder mit identischen Namen aus der Datenbanktabelle in den Datensatz zu übertragen. In der Praxis wird das erforderlich, um Daten aus mehreren abhängigen Tabellen zu lesen und in einem Datensatz zusammenzufassen. Dazu ein Beispiel:

```
REPORT yz432250.
DATA:
  BEGIN OF f_tabinfo,
    tabname TYPE tabname,
    ddtext  TYPE as4text,
  END OF f_tabinfo.

SELECT SINGLE *
  FROM dd02l
  INTO CORRESPONDING FIELDS OF f_tabinfo
  WHERE tabname = 'TADIR'.

SELECT SINGLE *
  FROM dd02t
  INTO CORRESPONDING FIELDS OF f_tabinfo
  WHERE tabname    = f_tabinfo-tabname
    AND ddlanguage = sy-langu.

WRITE: / f_tabinfo.
```

Auf ähnliche Weise können Sie mit INTO die Ergebnismenge auch in eine interne Tabelle umleiten. Auch dabei besteht die Möglichkeit, entweder eine interne Tabelle zu benutzen, deren Struktur der Datenbanktabelle entspricht oder aber nur namensgleiche Felder zuzuweisen:

```
... INTO TABLE itab
... INTO CORRESPONDING FIELDS OF TABLE itab
```

Die als Ziel angegebene interne Tabelle wird dabei jedes Mal komplett überschrieben, der alte Inhalt geht somit verloren. Dies lässt sich durch den Ersatz von INTO durch APPENDING vermeiden:

```
... APPENDING TABLE itab
... APPENDING CORRESPONDING FIELDS OF TABLE itab
```

Obwohl das Schlüsselwort INTO in dieser Form der Anweisung nicht mehr zu finden ist, handelt es sich trotzdem um eine INTO-Klausel im Sinne der Open-SQL-Syntax. Die Wirkung entspricht der INTO TABLE-Anweisung mit dem Unterschied, dass der bereits vorhandene Inhalt der internen Tabelle erhalten bleibt und die neu gelesenen Datensätze angefügt werden.

Falls in der SELECT-Klausel eine aus einzelnen Feldern bestehende Feldliste steht oder Aggregatausdrücke benutzt werden, muss mittels der INTO-Klausel für jedes in der SELECT-Klausel aufgeführte Feld oder jeden Aggregatausdruck ein Datenfeld benannt werden, welches das Ergebnis aufnimmt. Falls die Feldliste der INTO-Klausel mehr als ein Feld enthält, ist sie in runde Klammern einzuschließen. Die Feldnamen sind in diesem Fall durch Kommata zu trennen. Beispiele für die Anwendung wurden bereits gegeben. Die allgemeine Form lautet

```
... INTO field
... INTO (field_1, ..., field_n)
```

Die Klausel ORDER BY

Die Klausel ORDER BY erlaubt die Sortierung der Ausgabemenge. Es können sowohl der Zusatz PRIMARY KEY zur aufsteigenden Sortierung gemäß der Schlüsselfelder oder aber eine Feldliste verwendet werden. Die Feldliste kann als Direktwert notiert oder aber in einer internen Tabelle übergeben werden:

```
ORDER BY [PRIMARY KEY | fieldlist | (itab)]
```

Die Klausel GROUP BY

Mit der Klausel GROUP BY können Sie in einer SELECT-Anweisung Datensätze zu Gruppen zusammenfassen. Eine Gruppe besteht aus Datensätzen, die in ausgewählten Feldern identische Werte aufweisen. Für jede Gruppe erzeugt die SELECT-Anweisung nur einen Ausgabesatz. Falls in der SELECT-Klausel Aggregatfunktionen eingesetzt werden, so werden deren Ergebnisse jeweils für eine Gruppe ermittelt und ausgegeben.

Die für die Gruppenbildung zuständigen Felder werden in der Feldliste der GROUP BY-Klausel aufgeführt. Auch diese Klausel erfordert die Verwendung einer Feldliste in der SELECT-Klausel, in der alle zur Gruppenbildung benutzten Felder aufgeführt werden müssen. Außerdem muss die Feldliste Felder zur Aufnahme der Ergebnisse eventuell vorhandener Aggregatfunktionen besitzen. Es ist nicht sinnvoll, mit der SELECT-Klausel Felder von der Datenbank zu lesen, die nicht zur Gruppenbildung herangezogen werden, da diese wegen der Zusammenfassung von Datensätzen keinen gültigen Wert aufweisen wird. Die Feldliste der SELECT- und der GROUP BY-Klausel müssen daher, abgesehen von den Aggregatfunktionen, identisch sein. Diese Aggregatfunktionen dürfen sich

allerdings nur auf Felder beziehen, die nicht zur Unterscheidung der Datensatzgruppen benutzt werden. Es folgt ein Beispiel für den Einsatz der GROUP BY-Klausel:

```
REPORT yz432260.
DATA:
  obj LIKE tadir-object,
  anz TYPE I.

SELECT OBJECT COUNT( DISTINCT obj_name )
  INTO (obj, anz)
  FROM tadir
  WHERE author = sy-uname
  GROUP BY object .

   WRITE: / obj, anz.
ENDSELECT.
```

Dieses kurze Programm ermittelt alle vom aktuellen Anwender angelegten Entwicklungsobjekte (Programme, Transaktionen, Dynpros etc.), listet diese aber nicht einzeln auf, sondern summiert je Gruppe die Anzahl der Elemente je Objekttyp auf und schreibt lediglich die Anzahl auf den Bildschirm.

Die HAVING-Klausel

Mit der HAVING-Klausel ist es möglich, zusätzliche Bedingungen zur Auswertung der von GROUP BY erzeugten Datensatzgruppen zu formulieren. Diese Bedingungen beziehen sich meist auf Informationen, die bezüglich einer Datensatzgruppe ermittelt werden wie z.B. die Anzahl der Datensätze oder die Werte von Aggregatfunktionen. Zur Demonstration das leicht veränderte GROUP BY-Beispiel:

```
REPORT yz432270.
DATA:
  obj LIKE tadir-object,
  anz TYPE I.

SELECT OBJECT COUNT( DISTINCT obj_name )
  INTO (obj, anz)
  FROM tadir
  WHERE author = sy-uname
  GROUP BY object
  HAVING count( * ) > 1 .

    WRITE: / obj, anz.
ENDSELECT.
```

In diesem Beispiel sorgt die HAVING-Klausel dafür, dass nur die Objekt-Gruppen aufgelistet werden, zu denen mehr als ein Objekt existiert. Dabei bezieht sich der Ausdruck count(*) aber nicht auf die Gesamtzahl der durch SELECT ermittelten Sätze sondern separat auf jede durch GROUP BY erzeugte Gruppe.

Die WHERE-Klausel

Die WHERE-Klausel der SELECT-Anweisung kann einen oder mehrere Vergleiche enthalten, die bei Bedarf mit den Operatoren AND und OR verknüpft werden. In jedem Vergleich wird ein Feld der Tabelle auf Erfüllung oder Nichterfüllung einer Bedingung geprüft, wobei unterschiedliche Arten von Vergleichen (numerische, String- bzw. Musterprüfungen, Prüfung gegen eine Selektionstabelle) möglich sind. Ähnliche Vergleiche werden auch in einigen Steueranweisungen (z.B. IF, WHILE oder CHECK) benutzt, wobei aber Unterschiede zwischen den möglichen Ausdrücken der WHERE-Klausel und den Ausdrücken anderen Anweisungen bestehen.

Die wichtigste Klausel der Open-SQL-Kommandos ist die WHERE-Klausel, da sie die Auswahl der Datenbanksätze bestimmt. Sie verarbeitet einige unterschiedliche Ausdrücke, die in diesem Abschnitt beschrieben werden. Die WHERE-Klausel legt die Selektionsbedingung fest. Sie kann unterschiedliche Teilausdrücke auswerten, die durch Klammerung und die Operatoren AND und OR gruppiert werden können. Außerdem ist mit dem Operator NOT die Negation eines Teilausdrucks möglich. Folgende Auswahlkriterien stehen zur Verfügung:

- Vergleich eines Datenbankfelds mit einem Wert oder einem Feld.
- Prüfen eines Datenbankfelds gegen eine Menge von Werten oder Feldern.
- Prüfen eines Datenbankfelds gegen ein Muster.
- Prüfen eines Datenbankfelds gegen einen Bereich (untere und obere Grenze).
- Prüfen eines Datenbankfelds gegen eine speziell aufgebaute interne Tabelle (Selektion).
- Prüfen gegen Felder einer beliebigen internen Tabelle (nur für SELECT-Anweisung).
- Dynamische Deklaration der Bedingung (nur für SELECT-Anweisung).

Beim direkten Vergleich von Feld gegen Feld (oder Wert) stehen die gängigen Vergleichsoperatoren zur Verfügung. Diese können entweder als Symbol oder als zweistelliger Operator angegeben werden. Tabelle 3.21 zeigt die verfügbaren Operatoren.

Operator	Symbol	Bedeutung
EQ	=	gleich
NE	<>	ungleich
LT	<	kleiner
LE	<=	kleiner oder gleich
GT	>	größer
GE	>=	größer oder gleich

Tabelle 3.21
Vergleichsoperatoren

Die folgenden Beispiele für Ausdrücke in `WHERE`-Klauseln beziehen sich auf die Tabelle `TADIR`. Sie können daher durch Modifikation des Programms `YZ432210` getestet werden.

```
REPORT yz432280.
DATA:
  obj  TYPE tadir-object,
  name TYPE tadir-obj_name.

SELECT object obj_name
  INTO (obj, name)
  FROM tadir
  WHERE
    author = sy-uname and
    obj_name LIKE 'V\_T%' ESCAPE '\'.

  WRITE: / obj, name.
ENDSELECT.
```

Dabei sollte der Test auf den Anwendernamen in der `WHERE`-Klausel verbleiben, da sonst unter Umständen tausende Ergebniszeilen in der Liste erscheinen können. Denken Sie bitte daran, in allen Beispielen, in denen Programmnamen im Suchbegriff benutzt werden, die beiden Zeichen `Z4` durch Ihre Initialen zu ersetzen, damit wirklich nach den von Ihnen erzeugten Programmen gesucht wird.

Ein Selektionsausdruck, der alle Objekte mit einer Versionsnummer kleiner 3 auflistet, könnte lauten:

```
WHERE ... versid < 3
```

Soll nicht auf einen einzelnen Wert geprüft werden, sondern auf Übereinstimmung mit einem Element aus einer größeren Menge, kommt die Anweisung IN zum Einsatz. Die Mengenangabe steht dabei in runden Klammern.

```
WHERE ... object IN ('PROG', 'TABL', 'DOMA')
```

Die Prüfung gegen ein Muster erfolgt mit dem Befehl LIKE. Im Muster können zwei Zeichen als Wildcards (Musterzeichen, Joker) fungieren. Der Unterstrich „_" steht dabei als Muster für ein beliebiges Zeichen, während das Prozentzeichen „%" als Muster für eine beliebige Zeichenkette (einschließlich einer leeren) eingesetzt wird. Diese Prüfung ist allerdings nur für alphanummerische Werte möglich. Auch dazu eine Anweisung:

```
WHERE ... obj_name LIKE 'YZ4%'
```

Mitunter soll eines der beiden Zeichen % oder _ gesucht werden, wozu dessen Sonderbedeutung im Muster auszuschalten ist. Mit der ESCAPE-Anweisung kann ein beliebiges Zeichen definiert werden, das im Muster die Wirkung eines der beiden Musterzeichen aufhebt. Durch die freie Auswahl können auch schwierige Situationen gemeistert werden. Beim Test des folgenden Beispiels sollte die WHERE-Klausel ausnahmsweise nur einen einzigen Ausdruck enthalten. Das Suchkriterium ist restriktiv genug, um die Länge der Ergebnisliste in sinnvollen Grenzen zu halten. Ein zusätzlicher Test gegen den Anwendernamen würde die Zahl der gefundenen Datensätze auf 0 reduzieren, da Sie mit hoher Wahrscheinlichkeit noch kein Objekt angelegt haben, das auf das folgende Suchmuster passt:

```
WHERE obj_name LIKE 'V\_T%' ESCAPE '\'
```

Soll gegen einen aus oberer und unterer Schranke bestehenden Bereich geprüft werden, kommt die Anweisung BETWEEN zum Einsatz. Untere und obere Grenze werden durch AND voneinander getrennt. Bei den Grenzwerten kann es sich sowohl um Strings als auch um nummerische Werte handeln.

```
WHERE obj_name BETWEEN 'YZ432030' AND 'YZ432100'.
```

Sollen mehrere Datensätze selektiert werden, die sich wegen völlig unterschiedlicher Schlüssel oder einer dynamischen Auswahl der Selektionsmuster der Verwendung der anderen Selektionskriterien entziehen, wird eine so genannte Selektionstabelle eingesetzt. Diese hat einen speziellen Aufbau und kann im so genannten Selektionsbildschirm eines Reports vom Anwender interaktiv mit Daten gefüllt werden. Genaueres zur Verwendung von Selektionstabellen folgt in einem späteren Abschnitt. Geprüft wird gegen eine solche Selektionstabelle einfach durch Eintragen des Namens der Selektion nach der IN-Anweisung:

```
WHERE NAME IN selection.
```

Selektionstabellen werden nicht durch die Datenbank selbst verarbeitet, sondern durch die SAP-Steuerlogik in Einzelanweisungen zerlegt. Die Selektionstabelle darf daher nicht zu umfangreich werden, da die Datenbanksysteme die Größe von SQL-Statements beschränken. Als verwandte Möglichkeit steht daher ausschließlich für die `SELECT`-Anweisung die Prüfung gegen eine beliebig strukturierte interne Tabelle mit der Anweisung `FOR ALL ENTRIES` zur Verfügung. Dabei wird jeder Datensatz der Datenbanktabelle gegen alle Einträge einer internen Tabelle geprüft. Zusätzlich ist noch anzugeben, welche Felder verglichen werden sollen. Fehlt diese Angabe, prüft die `WHERE`-Klausel alle Felder mit identischem Namen. Ein Ausschnitt aus einer `WHERE`-Klausel könnte also folgendermaßen aussehen:

```
FOR ALL ENTRIES IN icondition WHERE name = icondition-name.
```

Dieser Befehl steht erst ab Release 3.0 zur Verfügung.

Wie aus den Beispielen ersichtlich, muss in der `WHERE`-Klausel nur der Feldname angegeben werden, nicht aber der Tabellenname, da dieser ja bereits aus der `FROM`-Klausel ersichtlich ist. Es ist aber durchaus möglich, den Wert, gegen den ein Tabellenfeld geprüft werden soll, in der Kopfzeile der Tabelle abzulegen. Es ergibt sich dadurch folgende etwas undurchsichtige Anweisung:

```
tadir-obj_name = 'YZ4%'.
SELECT * FROM tadir
WHERE obj_name LIKE tadir-obj_name.
```

Diese Anweisung ist nur verständlich, wenn man bedenkt, dass Tabelle und Arbeitsbereich zwei verschiedene Objekte sind. In der Kopfzeile wird zunächst ein Wert bereitgestellt. Die `SELECT`-Anweisung wird durch die interne Steuerlogik des R/3-Systems bearbeitet und erst dann zum Datenbankserver geschickt. Dieser erhält eine Anweisung ähnlich der Folgenden:

```
SELECT * FROM tadir
WHERE obj_name LIKE 'YZ4%'.
```

Es erfolgt also nicht, wie die ursprüngliche Anweisung auf den ersten Blick vermuten lässt, eine Überprüfung des jeweils in der Tabelle gelesenen Satzes gegen das Feld `obj_name` der Tabelle `tadir` (also gegen sich selbst), sondern gegen einen Wert, der in der Feldleiste `TADIR` abgelegt wurde.

Parameterübergabe in internen Tabellen

Die Parameter der verschiedenen Klauseln müssen nicht unbedingt hart im Programm notiert werden. Es ist möglich, allen Klauseln außer `INTO` die erforderlichen Parameter in einer internen Tabelle zu übergeben. Diese muss einen speziellen Aufbau haben (nur ein Feld, Typ C, Länge 72 Zeichen). In den verschiedenen Klauseln notieren Sie nur noch den Namen der jeweiligen internen Tabelle, eingeschlossen in runde Klammern:

```
... SELECT [SINGLE [FOR UPDATE] | DISTINCT] (itab_s)
... FROM (itab_f)
... WHERE (itab_w)
... GROUP BY (itab_g)
... HAVING (itab_h)
... ORDER BY (itab_o)
```

Die Parameterübergabe in internen Tabellen ermöglicht es, die SELECT-Anweisung weitgehend zur Laufzeit zu formulieren. Dies erleichtert die Programmierung von generischen Anwendungen, die sich zur Laufzeit auf unterschiedliche Datensatzstrukturen oder Abfragen einstellen können.

```
REPORT yz432290.

TYPES:
  t_line(72).
DATA:
  i_where TYPE t_line OCCURS 0 WITH HEADER LINE,
  i_to    LIKE tadir  OCCURS 0 WITH HEADER LINE,
  tabname LIKE dd02l-tabname.

i_where = 'OBJ_NAME LIKE ''YZ%'' '.
APPEND i_where.

tabname = 'TADIR'.

SELECT * FROM (tabname)
  UP TO 100 ROWS
  INTO TABLE i_to
  WHERE (i_where).

LOOP AT i_to.
  WRITE: / i_to-obj_name.
ENDLOOP.
```

Alias-Bezeichnungen

In den bisherigen Beispielen wurde in einer SELECT-Anweisung immer nur auf eine einzige Tabelle zugegriffen. Somit waren alle Feldangaben in der SELECT-Anweisung automatisch eindeutig. Die neue Klausel JOIN ermöglicht es nun, in einer SELECT-Anweisung mehr als eine Tabelle anzusprechen. Falls in zwei Tabellen Felder mit identischem Namen enthalten sind, muss durch Voranstellen des Tabellennamens vor den Feldbezeichner Eindeutigkeit erreicht werden. Um die dabei anfallende Schreibarbeit zu verringern, können in der FROM-Klausel Alias-Bezeichnungen für die Tabellen vergeben werden. In den übrigen Klauseln sind die Alias-Bezeichnungen alternativ zu den Tabellennamen verwendbar. Beispiele dazu finden Sie im Abschnitt zur JOIN-Klausel.

Joins

Ein Entwurfsziel beim Einsatz von relationalen Datenstrukturen besteht in der weitgehenden Reduzierung von Redundanzen. Dabei werden logisch zusammengehörende Daten in verschiedenen Tabellen abgespeichert, die Verknüpfung wird durch eindeutige Schlüssel übernommen. Innerhalb einer Anwendung ist es daher oft notwendig, Datensätze aus mehreren Tabellen miteinander zu kombinieren. Beispielsweise erfordert das Ausstellen einer Rechnung neben dem Lesen der Rechnungsposten auch die Ermittlung der Stammdaten des Rechnungsempfängers. Bisher mussten im R/3-System für vergleichbare Aufgaben diverse Hilfsmittel oder Kunstgriffe wie logische Datenbanken mit verschachtelten SELECT-Anweisungen oder temporäre Zwischenspeicher (interne Tabellen, Extrakt-Datenbestände) benutzt werden. Der SQL-Standard bietet zur Lösung derartiger Probleme die JOIN-Klausel an. Die in einer JOIN-Klausel enthaltenen Bedingungen legen fest, wie Datensätze aus zwei oder mehr Tabellen miteinander verknüpft werden sollen.

Der SQL-Standard unterscheidet dabei zwischen einem INNER JOIN und einem OUTER JOIN. Bei einem INNER JOIN ermittelt das Datenbanksystem zunächst alle Datensatzpaare, die der JOIN-Bedingung genügen. Dann werden auf diese Datenmenge die übrigen Klauseln des SQL-Statements, insbesondere die SELECT-Klausel, angewendet. In der Ergebnismenge erscheinen daher nur Datensätze, die sowohl den Bedingungen der SELECT-Klausel als auch der JOIN-Klausel genügen. Bedingt durch die UND-Verknüpfung der Bedingungen in beiden Klauseln hat es keinen Einfluss auf die Ergebnismenge, in welcher Klausel ein Selektionskriterium steht.

Bei einem OUTER JOIN werden allen Datensätzen einer primären Tabelle zunächst die Datensätze einer sekundären Tabelle zugeordnet, welche die JOIN-Bedingung erfüllen. Alle anderen Datensätzen der primären Tabelle werden mit einem leeren Datensatz verknüpft. Die bei diesem Zwischenschritt gefundene Datenmenge enthält also alle Datensätze der primären Tabelle. Auf diese Datenmenge wird dann wieder die SELECT-Klausel angewendet. Im Gegensatz zum INNER JOIN ist die Ergebnismenge eines OUTER JOIN davon abhängig, in welcher Klausel eine Bedingung notiert wird. Der Gedanke liegt nahe, auf diese Weise alle Datensätze der primären Tabelle zu ermitteln, für die der JOIN-Ausdruck nicht erfüllt ist, indem in der WHERE-Klausel ein Feld der sekundären Tabelle auf den Wert NULL getestet wird. Leider ist dies nicht möglich, da in der eigentlichen SELECT-Klausel nur Felder der primären Tabelle enthalten sein dürfen.

Die JOIN-Klausel wird in der FROM-Klausel notiert. Für einen INNER JOIN lautet die Syntax der Anweisung

```
... FROM primary_table [INNER] JOIN secondary_table ON
{condition}
```

Ein OUTER JOIN wird nicht durch das Schlüsselwort OUTER, sondern durch LEFT identifiziert. Die Syntax lautet daher

```
... FROM primary_table LEFT [OUTER] JOIN secondary_table ON
{condition}
```

Die von R/3 unterstützten Datenbanksysteme unterscheiden sich hinsichtlich der Kompatibilität zum SQL-Standard. Daher sind beim Einsatz der JOIN-Klausel einige Einschränkungen zu beachten:

- Joins können nicht verschachtelt werden, rechts vom Schlüsselwort JOIN darf daher nur ein Tabellen- oder View-Name stehen, aber kein weiterer JOIN-Ausdruck.
- Die Bedingungen eines Joins können nur durch AND verknüpft werden.
- Jede JOIN-Bedingung muss ein Feld aus der sekundären Tabelle enthalten.

Für einen OUTER JOIN gelten zusätzlich folgende Einschränkungen:

- In der WHERE-Klausel dürfen keine Felder der sekundären Tabelle enthalten sein.
- In den Bedingungen der JOIN-Klausel ist nur EQ (=) zugelassen.
- Die JOIN-Bedingung muss mindestens ein Feld aus der primären Tabelle enthalten.

Beispiele

Die folgenden Beispiele demonstrieren die JOIN-Klausel anhand einiger einfacher Selektionen. Sie greifen dazu auf Tabellen des Korrektur- und Transportwesens zu. Ebenso wie bei der des öfteren benutzten Tabelle TADIR sollten Sie nur lesend auf diese Tabellen zugreifen.

In der Tabelle E070 legt das Korrektur- und Transportwesen die Kopfzeilen aller Aufgaben und Aufträge ab. In der Tabelle E071 stehen die zu den Aufgaben oder Aufträgen gehörenden Elemente. Durch Auswertung dieser Tabellen können Sie unter anderem ermitteln, welcher Programmierer wann welches Objekt bearbeitet hat.

Das erste Beispielprogramm soll die Namen aller Programmierer ausgeben, die Objekte geändert haben, sowie die von ihnen geänderten Objekte auflisten. Da die Zahl der Einträge sehr groß sein kann, schränkt ein Parameter die Auswahl auf ausgewählte Programmierer ein. Ohne das Kommando JOIN sind zwei ineinander verschachtelte SELECT-Anweisungen erforderlich. Die erste würde in der Tabelle E070 lesen und alle Aufträge – sortiert nach dem Namen des Programmierers – ermitteln. Eine zweite SELECT-Anweisung würde aus der Tabelle E071 alle Objekte zu den jeweiligen Aufträgen lesen. Dabei weist diese einfache Variante einen Mangel auf. Wenn ein Anwender ein und dasselbe Objekt mehrfach bearbeitet, taucht es in der Ergebnismenge auf mehrfach auf. Die korrekte

Lösung der Aufgabe würde also auch noch die Verwendung einer internen Tabelle erfordern, um doppelte Einträge eliminieren zu können.

Das JOIN-Kommando ermöglicht eine wesentlich einfachere Programmierung. Innerhalb der JOIN-Klausel werden die beiden Tabellen E070 und E071 miteinander verknüpft. Da die Verknüpfung über ein Feld erfolgt, das in beiden Tabellen vorhanden ist, muss für die beiden Tabellen jeweils ein Alias angelegt werden, hier A und B. In der Verknüpfungsbedingung sorgen diese beiden Alias dann für eindeutige Feldbezeichner. Es ist nicht möglich, die Eindeutigkeit der Feldbezeichnungen durch Voranstellen des Tabellennamens zu erreichen.

Aus der Ergebnismenge interessieren laut Aufgabenstellung nur die Felder AS4USER, PGMID, OBJECT und OBJ_NAME. Diese werden in der Feldliste der SELECT-Anweisung aufgeführt. Da diese Felder jeweils nur in einer Tabelle enthalten sind, ist kein Alias erforderlich. Die weiteren Klauseln der SELECT-Anweisung sind bereits aus früheren Versionen bekannt, was eine weitere Erläuterung erübrigt.

```
REPORT yz432300.
DATA:
  e070 TYPE e070,
  e071 TYPE e071.

PARAMETERS p_user LIKE e070-as4user.
TRANSLATE p_user USING '*%?_'.

SELECT DISTINCT as4user pgmid object obj_name
  INTO (e070-as4user, e071-pgmid, e071-object, e071-obj_name)
  FROM e070 as a LEFT JOIN e071 as b ON a~trkorr = b~trkorr
  WHERE as4user LIKE p_user
  ORDER BY as4user pgmid object obj_name.

    WRITE: / e070-as4user, e071-pgmid, e071-object,
             e071-obj_name(30).
ENDSELECT.
```

Wie bereits erwähnt, darf der rechte Teil einer JOIN-Klausel keine weitere derartige Klausel enthalten, wohl aber der linke Abschnitt. Dies eröffnet die Möglichkeit, mehr als zwei Tabellen miteinander zu verknüpfen, wie das nächste Beispiel demonstriert. Es baut auf dem vorangegangenen Beispiel auf. Zusätzlich ermittelt die SELECT-Anweisung die zum jeweiligen Objekt gehörende Entwicklungsklasse:

```
REPORT yz432310.
DATA:
  e070  TYPE e070,
  e071  TYPE e071,
  tadir TYPE tadir.
```

```
PARAMETERS p_user LIKE e070-as4user.
PARAMETERS p_devc LIKE tadir-devclass.
TRANSLATE p_user USING '*%?_'.
TRANSLATE p_devc USING '*%?_'.

SELECT DISTINCT as4user b~pgmid b~object b~obj_name devclass
  INTO (e070-as4user, e071-pgmid, e071-object,
        e071-obj_name, tadir-devclass)
  FROM e070 AS a JOIN e071 AS b ON a~trkorr = b~trkorr
  JOIN tadir AS c ON
    b~pgmid = c~pgmid AND
    b~object = c~object AND
    b~obj_name = c~obj_name
  WHERE as4user LIKE p_user
    AND devclass LIKE p_devc
  ORDER BY as4user b~pgmid b~object b~obj_name.
  WRITE: / e070-as4user, e071-pgmid, e071-object,
           e071-obj_name(30),tadir-devclass.
ENDSELECT.
```

Das dritte Beispiel demonstriert einen Outer Join. Eine `SELECT`-Anweisung soll alle im aktuellen Mandanten existierenden Anwender und die möglicherweise von ihnen angelegten Aufgaben oder Aufträge des Korrektur- und Transportwesens ermitteln. Die Lösung ähnelt dem ersten Beispiel, wobei durch die Anweisung `LEFT JOIN` auch alle Anwender ohne Einträge im Korrekturwesen in der Ausgabemenge erscheinen.

```
REPORT yz432320.
DATA:
  usr    TYPE usr01-bname,
  trkorr TYPE e070-trkorr.

PARAMETERS p_user LIKE e070-as4user.
TRANSLATE p_user USING '*%?_'.

SELECT DISTINCT a~bname b~trkorr
  INTO (usr, trkorr)
  FROM usr01 AS a LEFT JOIN e070 AS b ON a~bname = b~as4user
  WHERE bname LIKE p_user
  ORDER BY a~bname.
  WRITE: / usr, trkorr.
ENDSELECT.
```

Subquerys

Ab Version 6.x können in der SELECT-Anweisung, genauer in der WHERE-Klausel, auch Subquerys enthalten sein. Eine Subquery liefert einen oder mehrere Werte einer Tabellenspalte. Diese Werte können innerhalb einer WHERE-Klausel anstelle eines einzelnen konstanten Suchmusters benutzt werden.

Eine Subquery beruht auf der SELECT-Klausel. Bedingung für eine Subquery ist, dass ihre Feldliste nur aus einem einzigen Feld besteht. Das folgende Listing stellt eine SELECT-Anweisung mit einer Subquery vor.

```
SELECT author count( * ) FROM tadir
  INTO (a, i)
  WHERE object = 'PROG'
  GROUP BY author
  HAVING count( * ) >=
     ( SELECT count( * ) FROM tadir
         WHERE author = 'MEIER' and object = 'PROG' ).

  WRITE: / a, i.


ENDSELECT.
```

Für das Verständnis ist es am einfachsten, die Subquery zunächst durch einen konstanten Wert, in diesem Fall eine Zahl, zu ersetzen. Die Anweisung erstellt zunächst eine Liste mit allen Programmierern, die Objekte vom Typ PROG, also Programme, erstellt haben. Dazu dient die GROUP BY-Klausel in Verbindung mit der WHERE-Klausel. Für jeden Programmautor wird außerdem die Anzahl der Programme ermittelt. Dafür ist die COUNT-Anweisung innerhalb der äußeren SELECT-Anweisung zuständig. Die zusätzlich angefügte HAVING-Klausel ist verantwortlich dafür, dass nur Entwickler mit einer bestimmten Mindestzahl von Programmen berücksichtigt werden. Dieser Vergleichswert könnte hart programmiert werden. Durch eine Subquery kann er aber auch zur Laufzeit dynamisch bereitgestellt werden. In diesem Beispiel wird durch die Subquery die Anzahl der Programme des Entwicklers MEIER ermittelt und an die äußere SELECT-Anweisung als Vergleichswert übergeben.

Die Subquery ist in diesem Beispiel so formuliert, dass maximal ein einziger Wert bereitgestellt wird. Sollte die Subquery möglicherweise mehrere Werte zurückliefern, so muss durch zusätzliche Anweisungen festgelegt werden, wie diese zu berücksichtigen sind. Soll die Vergleichsbedingung für alle von der Subquery gelieferten Ergebnisse erfüllt sein, so ist das Schlüsselwort ALL vor der Subquery zu notieren. Reicht es aus, wenn die Bedingung durch mindestens einen Wert erfüllt wird, benutzen Sie das Schlüsselwort ANY.

Die folgende Modifikation des Subquery-Beispiels ermittelt für alle Entwickler, deren Namen mit M beginnen, die Anzahl der erstellten Programme. Die äußere SELECT-Anweisung bleibt unverändert. Da mit der Option ALL geprüft wird, werden nur die Entwickler aufgelistet, die mehr Programme erstellt haben als jeder der in der Subquery gefundenen Programmierer.

```
SELECT author count( * ) FROM tadir
  INTO (a, i)
  WHERE object = 'PROG'
  GROUP BY author
  HAVING count( * ) >=
     ALL ( SELECT count( distinct obj_name ) FROM tadir
             WHERE author like 'M%' AND
                   object = 'PROG'
             GROUP BY author ).


  WRITE: / a, i.


ENDSELECT.
```

Ändern, Anfügen und Löschen von Datensätzen

Bevor Datensätze in einer Tabelle gelesen werden können, müssen sie natürlich zunächst in diese geschrieben werden. Weitere Anweisungen sind für das Modifizieren oder Löschen von Datensätzen notwendig. Neue Datensätze werden mit dem Kommando

```
INSERT table.
```

an eine Tabelle angefügt. Dabei wird der Inhalt des angegebenen Arbeitsbereichs in die Tabelle geschrieben. Das Feld SY-SUBRC gibt anschließend wieder Auskunft über den Erfolg des Kommandos. Ein Wert von 0 bedeutet, dass der neue Satz erfolgreich angefügt wurde, Werte größer 0 deuten auf einen Fehler hin. Ein Fehler tritt bei INSERT beispielsweise dann auf, wenn bereits ein Datensatz mit identischem Schlüssel existiert. Mit INSERT ist es also nicht möglich, den Inhalt bereits existierender Datensätze zu ändern. Dazu dient das Kommando

```
UPDATE table.
```

Es überschreibt einen existierenden Datensatz vollständig mit dem Inhalt des Arbeitsbereiches. Im Gegensatz zu INSERT muss der zu überschreibende Datensatz bereits existieren. Auch diese Anweisung beeinflusst den Inhalt des Feldes SY-SUBRC. Der Schlüssel zur Identifikation des Datensatzes wird wiederum dem Arbeitsbereich entnommen.

Ähnlich der gleichnamigen SQL-Anweisung UPDATE existiert auch für das ABAP-Kommando eine Variante, mit der ein Feld oder mehrere Felder von Datensätzen geändert werden können, die einem eingegebenen Muster entsprechen. In diesem Fall lautet die Syntax:

```
UPDATE table
SET expression
WHERE condition.
```

In dieser Form werden in allen Datensätzen, die der Bedingung genügen, die angegebenen Zuweisungen ausgeführt. Nicht immer ist es möglich, zwischen Einfügen und Ändern zu unterscheiden. In diesem Fall kann

```
MODIFY table.
```

verwendet werden. Dieses Kommando erkennt selbständig, ob in die Tabelle ein neuer Datensatz eingefügt oder ein bereits vorhandener geändert werden soll. Dieser Komfort geht allerdings zu Lasten der Performance.

Alle bisher genannten Kommandos außer dem Massen-Update arbeiten mit der Kopfzeile der Tabelle zusammen, der sie insbesondere den Schlüssel für den zu bearbeitenden Datensatz entnehmen. Dies ist auch beim Löschen von Datensätzen möglich. Ein einfaches

```
DELETE table.
```

reicht aus, um den Datensatz, der durch den Schlüssel im Arbeitsbereich identifiziert wird, zu löschen. Diese Form des Löschkommandos ist beispielsweise hilfreich, wenn innerhalb einer SELECT-Schleife einige der gelesenen Sätze gelöscht werden sollen. Aber ähnlich wie bei der Massenänderung mit UPDATE kann auch eine Massenlöschung stattfinden, wenn eine Auswahl der zu löschenden Datensätze über eine WHERE-Klausel erfolgt:

```
DELETE FROM table
WHERE condition.
```

Die Bedingungen der WHERE-Klausel entsprechen dabei, ebenso wie beim Kommando UPDATE, denen der SELECT-Anweisung.

Die erwähnten Kommandos besitzen einige weitere Varianten und Optionen, deren Syntax Sie bitte der Kurzreferenz entnehmen. Diese Optionen dienen vor allem der Abschaltung des automatischen Mandantenhandlings (CLIENT SPECIFIED) und der Entnahme der zu modifizierenden Felder aus einer Feldleiste oder einer internen Tabelle.

Nach Ausführung der Kommandos zur Datenbankmodifikation sind die Änderungen auf der Datenbank noch nicht endgültig festgeschrieben. Sie können mit

```
ROLLBACK WORK.
```

jederzeit wieder rückgängig gemacht werden. Dies kann z.B. erforderlich werden, wenn beim Schreiben in mehrere logisch voneinander abhängige Tabellen eines der Schreibkommandos mit einem Fehler endet. Um die Konsistenz der Daten zu wahren, müssen in diesem Fall die Änderungen in allen Tabellen widerrufen werden. Als Gegenstück zu `ROLLBACK WORK` muss natürlich auch ein Kommando zum Bestätigen der Änderungen existieren. Es lautet

```
COMMIT WORK.
```

Nach Ausführung dieses Kommandos sind alle Modifikationen endgültig festgeschrieben. In diesem Zusammenhang muss auf eine Besonderheit des SAP-Systems hingewiesen werden. Bei jedem Bildwechsel, also jedem Beenden eines Dynpros, führt die interne Steuerlogik selbständig ein `COMMIT WORK` aus. Neben dem Festschreiben der Modifikationen löscht dieses Kommando beispielsweise den internen Cursor (Datensatzzeiger) einer `SELECT`-Anweisung. Zwischen den Anweisungen `SELECT` und `ENDSELECT` darf deshalb kein Dynpro-Wechsel stattfinden. Es ist daher nicht ohne weiteres möglich, innerhalb einer solchen Schleife die gelesenen Datensätze nacheinander in einem Dynpro zu bearbeiten.

Die beschriebenen Kommandos zum Aktualisieren von Datenbeständen werden in ABAP-Anwendungen sehr oft in spezielle Programmteile verlagert, deren einzige Aufgabe das Schreiben von Informationen in die Datenbank, also die Verbuchung ist. Da derartige Verbuchungsfunktionen vor allem in Dialogprogrammen eingesetzt werden, finden Sie weitergehende Ausführungen zur Verbuchung im Abschnitt 3.2.17.

Das folgende kleine Programm füllt eine Tabelle mit einigen Datensätzen, die als Testdaten für einige weitere Programme dieses Buches benötigt werden. Führen Sie dieses Programm nicht mehr aus, wenn Sie im Rahmen des Beispiels in Kapitel 7 bereits Daten erfasst haben! Zu Beginn dieses Programms wird der eventuell schon vorhandene Inhalt der Tabelle gelöscht!

Um dieses Programm ausführen zu können, muss natürlich erst die Tabelle erzeugt werden. Sie gehört nicht zum SAP-Standard. Im Abschnitt über die Pflege der Dictionary-Elemente wird beschrieben, wie Datenelemente, Domänen und Tabellen angelegt werden.

```
REPORT yz432330.
DATA yz4stock TYPE yz4stock.

DELETE FROM yz4stock WHERE wkz LIKE '%'.

yz4stock-wkz = '123456'.
yz4stock-name = 'SAP'.
INSERT yz4stock from yz4stock.

yz4stock-wkz = '654321'.
yz4stock-name = 'XYZ-AG'.
```

```
INSERT yz4stock from yz4stock.

yz4stock-wkz = '888888'.
yz4stock-name = 'NAGEL & CO'.
INSERT yz4stock from yz4stock.
COMMIT WORK.

WRITE: / 'Inhalt nach Anfügen'.
SELECT * FROM yz4stock.
  WRITE: / yz4stock-wkz, yz4stock-name, yz4stock-branch.
ENDSELECT.

UPDATE yz4stock
SET branch = '1'
WHERE name = 'SAP' OR name = 'XYZ-AG'.

WRITE: /, /, 'Inhalt nach Update'.
SELECT * FROM yz4stock.
  WRITE: / yz4stock-wkz, yz4stock-name, yz4stock-branch.
ENDSELECT.

DELETE FROM yz4stock
WHERE branch <> '1'.

WRITE: /, /, 'Inhalt nach Löschen'.
SELECT * FROM yz4stock.
  WRITE: / yz4stock-wkz, yz4stock-name, yz4stock-branch.
ENDSELECT.
```

3.2.10 Datencontext

Im SAP-System existieren einige Tabellen mit zentraler Bedeutung. Nahezu alle Anwendungen eines betriebswirtschaftlichen Moduls greifen auf diese Tabellen zu. Oft werden anwendungs- und anwenderübergreifend nur wenige unterschiedliche Datensätze gelesen. Als Beispiele seien die Angaben zu Buchungskreisen oder zu Währungen genannt. Andere Datensätze, z.B. Katalogdaten, besitzen einen relativ statischen Charakter. Sie ändern sich nur in großen Zeitabständen. In beiden Fällen liest jede Anwendung die Daten neu und verursacht so eine erhebliche Belastung des Systems.

Zur Vermeidung der unnötigen Datenselektion stellt SAP das Konzept des *DatenContextes* bereit. Ein Datencontext ist ein Puffer, der auf einem Applikationsserver angelegt wird und dessen Inhalt allen Anwendungen und allen Anwendern, die auf diesem Applikationsserver arbeiten, zur Verfügung steht. Im Datencontext werden Datensätze aus einer oder mehreren logisch zusammengehörenden Tabellen gespeichert. Beim erstmaligen Zugriff auf einen Datensatz

wird dieser aus der Datenbanktabelle gelesen und im Context abgelegt. Dort steht er dann allen Anwendern und Anwendungen zur Verfügung. Möchte eine andere Anwendung denselben Datensatz lesen, so wird dieser nicht nochmals von der Datenbank geholt, sondern aus dem Context bereitgestellt. Die Performance ist dabei nur unbedeutend schlechter als die einer MOVE-CORRESPONDING-Anweisung.

In der Version 6.x können Sie neben Tabellen auch Funktionsbausteine als Datenquellen angeben. Da sich an der prinzipiellen Funktion des Context dadurch aber nichts ändert, soll diese Möglichkeit nicht näher beschrieben werden.

Im Programm ist ein Context ähnlich wie ein Datentyp zu deklarieren. Das übernimmt die Anweisung

```
CONTEXTES contextname.
```

Mit Hilfe der Doppelpunktvariante des Kommandos können Sie auch mehrere Contexte gleichzeitig deklarieren.

Bei der Deklaration eines Contextes legt das System automatisch mehrere Datentypenan. Einer der Datentypen erhält die Bezeichnung CONTEXT_contextname. Diesen Datentyp müssen Sie benutzen, um mit der Anweisung DATA Instanzen eines Contextes zu deklarieren. Die weiteren Datentypen beschreiben die Felder des Contextes. Sie bekommen Bezeichnungen der Form CONTEXT_T_contextname-contextfield.

Der Zugriff auf einen Context ist nur über eine mit DATA erzeugte Instanz möglich. Sie können mehrere Instanzen eines Contextes anlegen, um parallel auf verschiedene Datensätze zugreifen zu können.

```
DATA context TYPE contexttype.
```

Einer Instanz eines Contextes müssen nach der Deklaration die zur Selektion erforderlichen Schlüsselwerte übergeben werden. Dazu dient das Kommando

```
SUPPLY { parameter = value } TO CONTEXT context.
```

Im weiteren Verlauf der Anwendung werden die entsprechenden Datenfelder dann mit der Anweisung DEMAND gelesen.

```
DEMAND { parameter = value } FROM CONTEXT context.
```

Erst beim Ausführen der DEMAND-Anweisung wird im Context nach den entsprechenden Daten gesucht. Bei Bedarf erfolgt dabei eine Datenbankselektion. Bei der Verarbeitung eines Contextes können Fehlermeldungen entstehen. Es ist möglich, diese Meldungen mit dem Kommandozusatz

```
... MESSAGES INTO itab
```

in eine interne Tabelle umzuleiten. Diese Tabelle muss die Struktur von SYMSG besitzen.

Ein Beispiel

Das nachfolgende Beispiel soll Definition und Einsatz eines Contextes zeigen, der die Entwicklungsklasse eines konkret vorgegebenen Objekts ermittelt und den für diese Entwicklungsklasse zuständigen Bearbeiter zurückliefert.

Um in einem Programm einen Datencontext benutzen zu können, ist er zunächst mit der Transaktion `SE33` zu definieren. Bei dieser Definition werden die beteiligten Tabellen, ihre Beziehungen untereinander sowie die interessierenden Felder festgelegt. Bei diesen Feldern handelt es sich um die Schlüsselfelder, mit deren Hilfe ein Datensatz selektiert wird sowie die zu lesenden Datenfelder, die den eigentlichen Inhalt des Datensatzes ausmachen.

Im Startbild der Transaktion `SE33` (siehe Abbildung 3.28) ist lediglich der Name des Contextes (`YZ_TADIR`) einzutragen und die gewünschte Funktion per Drucktaste zu wählen.

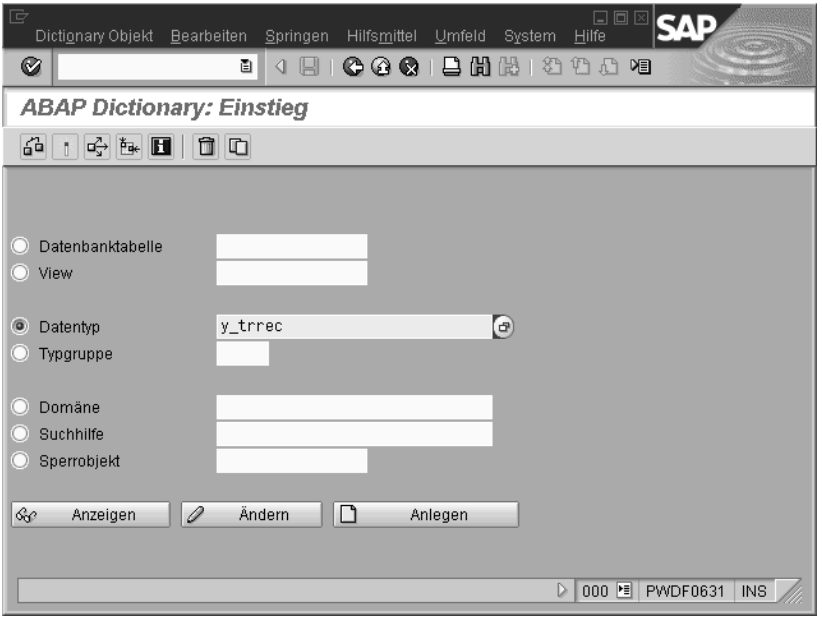

Abbildung 3.28 **© SAP AG**
Startbild der Context-Pflege

Beim Anlegen eines neuen Contextes pflegen Sie die Beschreibung und ein Kürzel für die Anwendungsgruppe in einem auf das Startbild folgenden Dynpro (Abbildung 3.29).

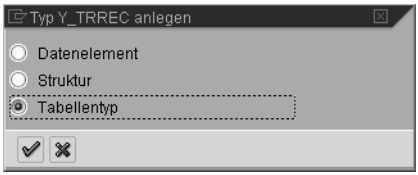

Abbildung 3.29 **© SAP AG**
Attribute eines Contextes pflegen

Von dort aus gelangen Sie mit der Menüfunktion SPRINGEN | FELDER UND MODULE oder der Funktionstaste F6 zum eigentlichen Bearbeitungsbild (Abbildung 3.30) für den Context. Falls Sie einen bereits existierenden Context bearbeiten, gelangen Sie vom Startbild direkt zum Bearbeitungsbild.

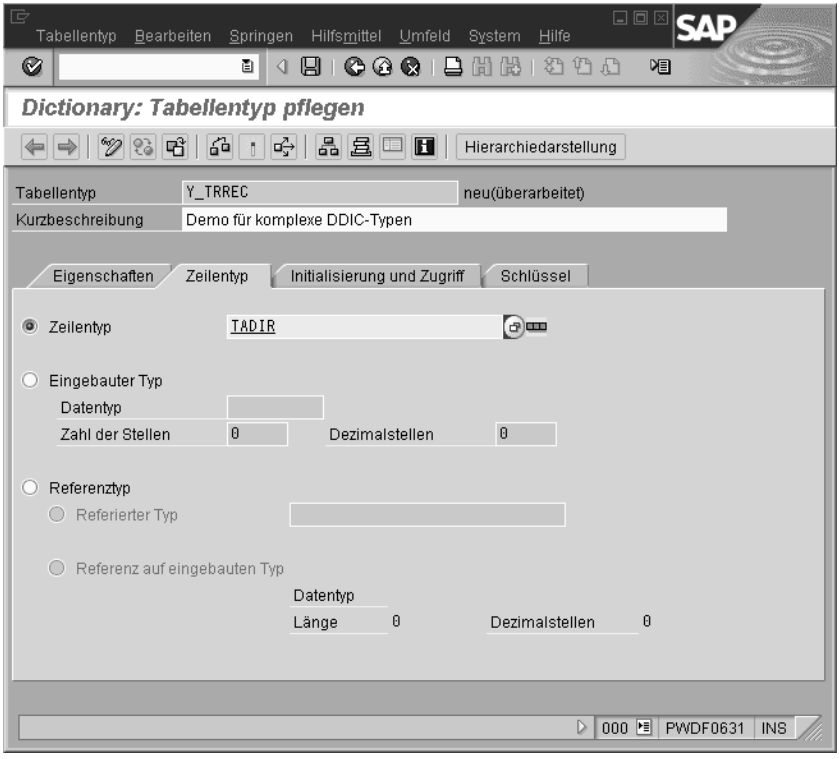

Abbildung 3.30 **© SAP AG**
Eigenschaften eines Contextes pflegen

Das Pflege-Dynpro für den Context besteht aus mehreren Table Views. Deren Größe wird vom R/3-System an die Größe des aktuellen Fensters angepasst. Allerdings reicht auch auf großen Monitoren die zur Verfügung stehende Fläche

nicht für ein wirklich komfortables Arbeiten aus. Sie können daher jeweils einen Table View durch einen Mausklick auf das Lupensymbol an der linken oberen Ecke des Table Views auf die Größe des Fensters vergrößern.

Zunächst müssen im Table View *Module* (siehe Abbildung 3.31) alle beteiligten Tabellen (`TADIR` und `TDEVC`) eingetragen werden. Dazu müssen Sie ggf. mit dem Einfügen-Symbol links neben dem Table View neue Eingabezeilen erzeugen.

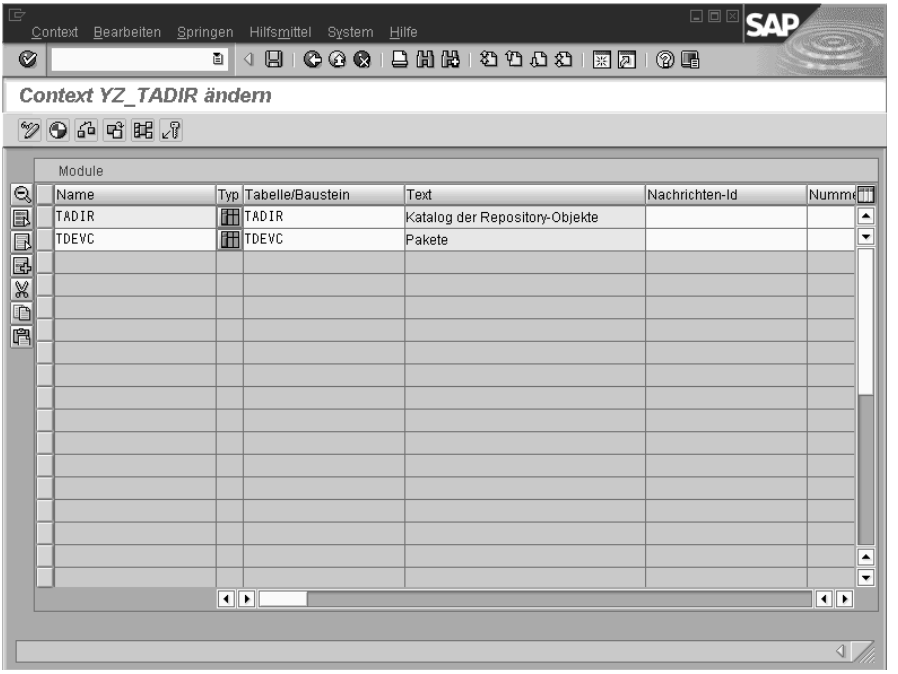

Abbildung 3.31 **© SAP AG**
Beteiligte Tabellen erfassen

Das System übernimmt die Felder dieser Tabellen in einen zweiten Table View, der die Bezeichnung *Parameter* besitzt. Er befindet sich im Übersichts-Dynpro (Abbildung 3.30) rechts unten. In diesem Table View (Abbildung 3.32) erscheinen jeweils alle Felder der Tabelle, die im *Module*-Table View per Doppelklick ausgewählt wird. Der Pfeil in der Spalte I/O legt fest, ob es sich um ein Import- oder Exportfeld handelt. Die Markierung kann per Doppelklick umgeschaltet werden. Importfelder werden mittels der Anweisung `SUPPLY` mit Suchbegriffen gefüllt. Die Anweisung `DEMAND` hingegen liest den Inhalt von Exportfeldern und stellt deren Werte dem rufenden Programm zur Verfügung. Die aktuellen Einstellungen sind bereits korrekt, da im Standardfall alle Schlüsselfelder als Import- und alle anderen als Export-Felder vorgegeben werden.

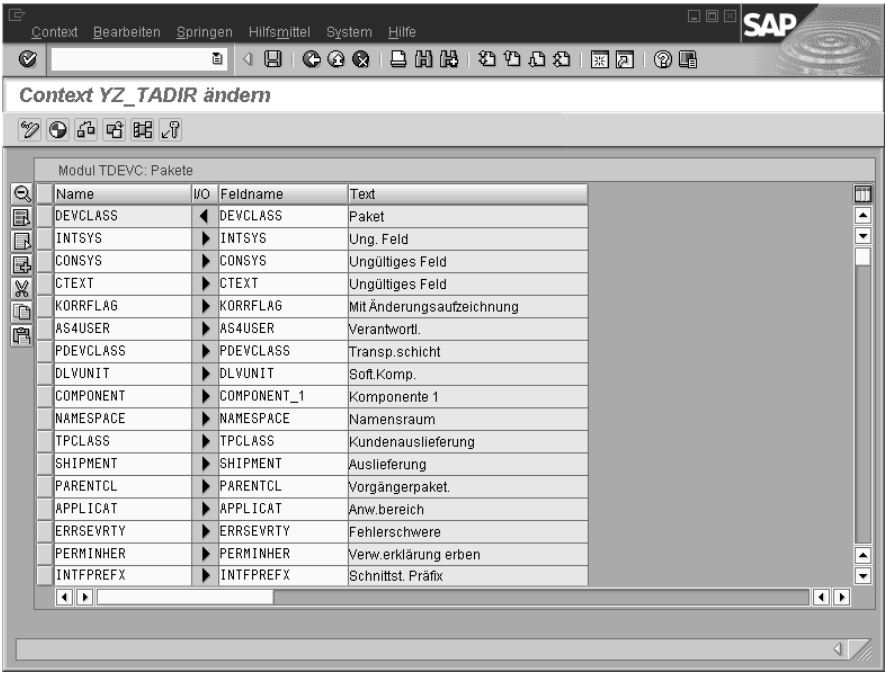

Abbildung 3.32 **© SAP AG**
Im- und Export-Parameter festlegen

Um eine Selektion in mehreren Tabellen durchzuführen, müssen Sie entweder die Importfelder entsprechend füllen oder eine Feldverbindung herstellen. Im Falle von Feldverbindungen wird ein Feld einer Tabelle automatisch mit dem Wert des verbundenen Feldes einer anderen Tabelle gefüllt. Derartige Verbindungen werden zwischen Fremdschlüsselfeldern bereits vom System automatisch hergestellt. Das trifft in diesem Fall auf das Feld `DEVCLASS` beider Tabellen zu. Für dieses Beispiel sind also keine weiteren Aktionen notwendig, der Context kann generiert (Funktion CONTEXT | GENERIEREN) und in Anwendungen benutzt werden. Die folgende Beispielanwendung demonstriert den Einsatz.

```
REPORT yz432340.

CONTEXTES yz_tadir.

DATA: c1   TYPE context_yz_tadir,
      user TYPE context_t_yz_tadir-as4user.

PARAMETERS: p_pgm LIKE tadir-pgmid DEFAULT 'R3TR',
            p_obj LIKE tadir-object DEFAULT 'PROG',
            p_nam LIKE tadir-obj_name.
```

```
SUPPLY pgmid = p_pgm
       object = p_obj
       obj_name = p_nam

  TO CONTEXT c1.

DEMAND as4user = user FROM CONTEXT c1.

WRITE: / user.
```

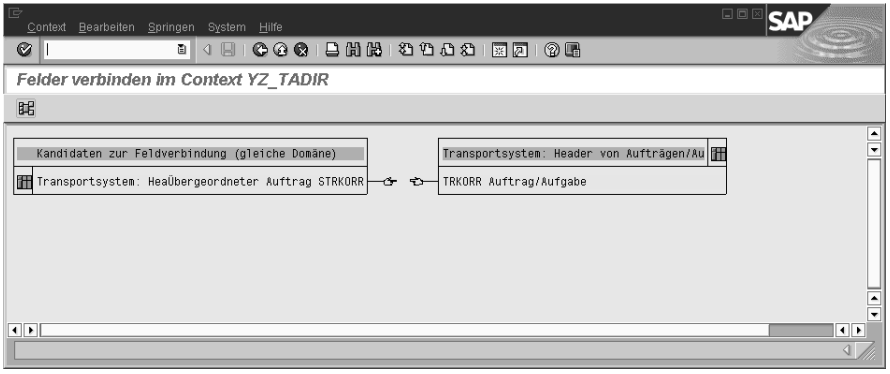

Abbildung 3.33 **© SAP AG**
Feldverbindungen herstellen

Sollte das System keine direkte Verknüpfung herstellen können, müssen Sie diese manuell herstellen. Dazu markieren Sie im Table View *Felder* das entsprechende Feld und rufen die Funktion CONTEXT | FELDER VERBINDEN auf. Im vorliegenden Beispiel existieren keine weiteren Felder, für die eine Feldverbindung hergestellt werden könnte. Die Demonstration der Funktion erfordert, dass Sie zusätzlich die Tabelle `E070` in den Context aufnehmen. Danach können Sie mit der Funktion CONTEXT | FELDER VERBINDEN (alternativ Funktionstaste F8) die Suche nach möglichen Verbindungen einleiten. Das R/3-System ermittelt anhand der Domänen der Datenfelder mögliche Verknüpfungen. Durch einen Klick auf das Tabellensymbol am Anfang jeder Zeile können Sie eine der vorgeschlagenen Verknüpfungen aktivieren. Das entsprechende Dynpro sehen Sie in Abbildung 3.33.

3.2.11 Bearbeiten von persistenten Daten-Clustern

Nicht immer sind Informationen so klar strukturiert, dass sie problemlos in Datensätzen von Tabellen abgelegt werden können. Oft wird es erforderlich, in einem Programm mehrere Datenobjekte zu benutzen, die zwar logisch zusammengehören, aber eine völlig unterschiedliche Struktur oder Größe aufweisen.

Neben herkömmlichen Datenbanktabellen stehen auch spezielle Tabellen zur Verfügung, in denen so genannte *Daten-Cluster* dauerhaft gespeichert werden können. Unter Daten-Cluster versteht man ein oder mehrere logisch zusammengehörende Felder oder interne Tabellen, die unter einem gemeinsamen Schlüssel in einer Datenbanktabelle abgespeichert werden können. Beim Speichern von internen Tabellen bleiben die Kopfzeilen unberücksichtigt, es werden nur die in der internen Tabelle gespeicherten Daten in den Daten-Cluster geschrieben.

Die zum Speichern von Daten-Clustern benutzte Datenbanktabelle muss eine spezielle Struktur aufweisen. Je nach Gestaltung der zur Ablage der Cluster benutzten Tabelle können Sie Daten-Cluster mandantenabhängig oder mandantenübergreifend abgelegen. Näheres zum Aufbau dieser Tabellen folgt in Kapitel 5.1.4 (Tabellen für Datencluster).

Das Schreiben von Daten erfolgt mit dem Kommando

```
EXPORT elements TO DATABASE table(area) ID key.
```

Bei `elements` handelt es sich um die Liste der zu speichernden Objekte. Der Name der Datenbanktabelle, in der die Daten abgelegt werden, ist mit `table` anzugeben. Innerhalb jeder Tabelle existieren verschiedene Gebiete (`area`), für die ein zweistelliger Bezeichner benutzt wird. Dieser Bezeichner ist beim Aufruf des Kommandos stets als Direktwert anzugeben. In einem Gebiet werden die einzelnen Einträge durch einen eindeutigen Schlüssel `key` voneinander unterschieden. Der Schlüssel wird als ein einzelnes Feld übergeben. Er wird durch das `EXPORT`-Kommando in die Schlüsselfelder der Datenbanktabelle aufgespalten.

Gelesen werden die Daten eines Daten-Clusters mit dem Kommando

```
IMPORT elements FROM DATABASE table(area) ID key.
```

Die Parameter entsprechen denen des `EXPORT`-Kommnados. In der Grundform des Kommandos müssen die Namen der zu lesenden Elemente mit denen der geschriebenen übereinstimmen. Das bedeutet, die Daten werden in genau den Feldern abgelegt, aus denen sie zuvor gelesen wurden. Diese Einschränkung kann mit den Optionen

```
... FROM source ...
```

bzw.

```
... TO target ...
```

aufgehoben werden, wobei diese Optionen für jedes Element separat notiert werden müssen.

Die erwähnten Kommandos `IMPORT` und `EXPORT` füllen nur die Schlüsselfelder der Tabelle und den eigentlichen Daten-Cluster. Darüber hinaus kann eine für Daten-Cluster vorgesehene Tabelle auch noch andere Felder besitzen. Diese Felder können mit den für Datenbanktabellen verfügbaren Kommandos (`SELECT`, `UPDATE`, …) bearbeitet werden.

Aus den direkt zugänglichen Feldern einer Tabelle für Daten-Cluster kann nicht auf den eigentlichen Inhalt geschlossen werden. Es ist zwar möglich, durch Lesen der Schlüsselfelder der Tabelle die Schlüssel aller Daten-Cluster zu lesen, die Namen der Objekte im Cluster können aber nur mit einem speziellen Kommando ermittelt werden.

```
IMPORT DIRECTORY
  INTO itab
  FROM DATABASE table(area) ID key.
```

Dieses Kommando schreibt die Informationen zu allen Elementen im jeweiligen Cluster in eine interne Tabelle. Diese Tabelle muss der Dictionary-Struktur `CDIR` entsprechen.

Da Länge und Typ der Elemente in einem Daten-Cluster beliebig sein können, eignen sich Daten-Cluster unter anderem zur einfachen Ablage von Binärdaten beliebiger Herkunft. Derartige Daten können z.B. per Upload-Funktion von einem Frontend in das System geschrieben werden. Ein Beispiel zum Up- und Download von Daten und zur Anwendung der Daten-Cluster finden Sie im Abschnitt 3.10.3.

3.2.12 Unterprogramme

Umfangreiche Programme zwingen zur Modularisierung. Damit ist es einerseits möglich, den Quelltext übersichtlicher zu halten, andererseits können Programmteile mehrfach verwendet werden. Die einfachste Variante zur Aufspaltung eines Programms besteht im Einsatz von Unterprogrammen. Die unter ABAP benutzten Unterprogramme ähneln weitestgehend den von Pascal bekannten Prozeduren. Sie können Parameter entgegennehmen, liefern aber keinen Funktionswert zurück.

Definition

Ein Unterprogramm wird mit der Anweisung:

```
FORM subroutine_name [parameters].
  statements
ENDFORM.
```

definiert. Wegen des Namens der Anweisungen werden derartige Unterprogramme im SAP-Sprachgebrauch auch als *Form-Routinen* oder kurz *Forms* bezeichnet. Der Name darf bis zu 30 Zeichen lang sein. Er kann zwar auch mit Ziffern und sogar Sonderzeichen beginnen, im praktischen Gebrauch sollte man allerdings die gleichen Namenskonventionen wie bei Feldnamen beachten. Unterprogrammnamen sollten also mit einem Buchstaben beginnen, möglichst selbsterklärend sein und als einziges Sonderzeichen den Unterstrich enthalten.

Nach der Definition mit FORM folgen die Anweisungen des Unterprogramms, den Abschluss bildet die Anweisung ENDFORM. In Form-Routinen können prinzipiell alle Anweisungen stehen, die auch im Hauptprogramm stehen dürfen. Einschränkungen sind nur bezüglich der weiteren Modularisierung zu beachten. So dürfen in Form-Routinen keine weiteren Anweisungen zur Modularisierung des Programms stehen, also keine Zeitpunktanweisungen und keine weiteren Deklarationen von Unterprogrammen. Die Deklaration von lokalen Daten innerhalb von Unterprogrammen hingegen ist problemlos möglich.

Bei der Deklaration von Unterprogrammen in Reports lauert eine Falle auf den Programmierer. Neben den Unterprogrammen existieren weitere Anweisungen zur Modularisierung, z. B. so genannte Zeitpunktanweisungen. Der zu diesen Anweisungen gehörende Anweisungsblock wird nicht explizit durch ein Kommando abgeschlossen. Er endet vielmehr an einer weiteren Modularisierungsanweisung oder am Programmende. Eine FORM-Anweisung beendet daher den Anweisungsblock einer eventuell vorangegangenen Zeitpunktanweisung. Diese Zusammenhänge sind leicht zu überblicken. Die Falle besteht darin, dass die Anweisungen PROGRAM oder REPORT ähnlich wie die noch zu erläuternde Zeitpunktanweisung START-OF-SELECTION arbeiten. Anweisungen eines Reports werden nur ausgeführt, wenn sie nach einer START-OF-SELECTION-Anweisung oder aber unmittelbar nach PROGRAM oder REPORT stehen. Wird zu Beginn eines Reports ein Unterprogramm definiert, werden die nach ENDFORM stehenden Anweisungen des Reports nur ausgeführt, wenn nach ENDFORM eine START-OF-SELECTION-Anweisung steht. In einfachen Reports sollte die Definition von Unterprogrammen daher am Programmende erfolgen. Umfangreichere Programme hingegen verlagern Unterprogramme und Deklarationen oft in Include-Dateien, die am Programmanfang eingebunden werden. In einem solchen Fall sollte das „Hauptprogramm“ durch eine START-OF-SELECTION-Anweisung eingeleitet werden. Mehr Informationen zur Funktion von Zeitpunktanweisungen und deren Verwendung finden Sie im Abschnitt über die Reportprogrammierung.

Aufruf von Unterprogrammen

Der Aufruf von Unterprogrammen erfolgt mittels des Kommandos:

```
PERFORM subroutine_name ... .
```

Dabei können Felder oder Tabellen des aktuellen Programms als aktuelle Parameter mitgegeben werden. Dies ist durch zwei zusätzliche Schlüsselwörter zu kennzeichnen. Falls interne Tabellen der Grundform mitgegeben werden sollen, ist die Anweisung TABLES erforderlich:

```
PERFORM subroutine_name TABLES itab1 itab2 ... .
```

Elementare Felder, Feldleisten und interne Tabellen der drei neuen Tabellenarten hingegen werden mit USING übergeben:

```
PERFORM subroutine_name USING param_1 param_2 ... .
```

Der Aufruf von Unterprogrammen kann auch über Programmgrenzen hinweg erfolgen. Man spricht dann von externen Aufrufen oder im SAP-Sprachgebrauch von externen Performs. Für diese Art des Aufrufs ist der Name des fremden Programms oder Modul-Pools in runden Klammern hinter dem Namen der Form-Routine einzutragen.

```
PERFORM subroutine_name(program) ... .
```

Eine andere Variante, die auch die dynamische Übergabe von Routinen- und Programmnamen ermöglicht, ist der Zusatz IN PROGRAM zur PERFORM-Anweisung. Entweder werden die beiden Namen direkt in der Anweisung notiert oder sie stehen in Feldern, die in der Anweisung durch Einfügen in runde Klammern zu kennzeichnen sind, also

```
PERFORM subroutine_name IN PROGRAM program ... .
```

bzw.

```
PERFORM (subroutine_name) IN PROGRAM (program) ... .
```

Parameter

Sie können an Unterprogramme Parameter übergeben. Dabei findet eine Typprüfung statt, deren Umfang sowohl von der Version Ihres R/3-Systems als auch von einigen Optionen im Programm abhängt. Als Parameter können alle bisher behandelten Datentypen außer Datenbanktabellen, also Einzelfelder, Feldleisten und interne Tabellen, Verwendung finden. Bei der Übergabe der Parameter kann zudem festgelegt werden, ob es sich um Wert- oder Referenzparameter handeln soll. Letztere ermöglichen die Rückgabe von Werten an das aufrufende Programm.

Nun konkret zur Syntax. In der FORM-Anweisung wird nach dem Schlüsselwort USING die Liste der formalen Parameter notiert. Diese enthält Namen für Felder, die nur innerhalb der Form-Routine verfügbar sind. Die formalen Parameter werden ausschließlich durch Leerzeichen voneinander getrennt.

```
FORM subroutine_name USING formal_par_1 formal_par_2 ...
formal_par_n.
```

Beim Aufruf eines Unterprogramms mit PERFORM werden aktuelle Parameter angegeben, die entsprechend der Reihenfolge in der Parameterliste den formalen Parametern zugewiesen werden. Alle nur mit USING definierten formalen Parameter sind Referenzparameter. Dies bedeutet, dass Änderungen des Wertes der formalen Parameter auch in den aktuellen Parametern des aufrufenden Pro-

gramms stattfinden. Der formale Parameter verweist auf denselben Speicherplatz wie der aktuelle Parameter.

Derartige Wertübergaben können zu unerwünschten Nebenwirkungen führen, wenn formalen Parametern Werte zugewiesen werden, obwohl die aktuellen Parameter eigentlich nicht geändert werden sollen. Dies muss nicht unbedingt in derselben Routine erfolgen. Oft ruft ein Unterprogramm weitere Unterprogramme auf und reicht dabei formale Parameter weiter. Der Ursprung eines formalen Parameters ist dann nur noch schwer zu überschauen, Nebenwirkungen können nicht mehr abgeschätzt werden. Neben den Referenzparametern kommen daher auch Wertparameter zum Einsatz. Diese sind Kopien der aktuellen Parameter, Zuweisungen werden zwar im formalen, nicht aber im aktuellen Parameter wirksam. Die Syntax für einen derartigen Aufruf ist etwas umfangreicher:

```
FORM subroutine_name USING VALUE(formal_par_1)
VALUE(formal_par_1) ... .
```

Zu guter Letzt existiert auch noch eine etwas inkonsequente Mischung zwischen Wert- und Referenzparametern, die in dieser Form einmalig sein dürfte. Die Syntax ähnelt der Definition von Wertparametern:

```
FORM subroutine_name CHANGING VALUE(formal_par_1) ... .
```

Prinzipiell sind diese Parameter Referenzparameter, ermöglichen es also, Werte an die aktuellen Parameter zu übertragen. Diese Übertragung findet allerdings erst statt, wenn die Anweisung `ENDFORM` erreicht wird. Falls das Unterprogramm mit einer Fehlernachricht (Anweisung `MESSAGE`) abbricht, werden die Inhalte der aktuellen Parameter nicht geändert. Unterprogrammabbrüche mit `EXIT` hingegen springen zur `ENDFORM`-Anweisung, wodurch die Wertübertragung ausgeführt wird.

Neben den Feldern und Feldleisten können Form-Routinen auch interne Tabellen als Parameter entgegennehmen. Hier ist allerdings keine Unterscheidung zwischen Wert- und Referenzparameter möglich, interne Tabellen werden immer als Referenz übergeben. Änderungen des Tabelleninhaltes bleiben also nach Verlassen des Unterprogramms bestehen. Ab Version 4.0 ist bei der Übergabe von Tabellen als Parameter die jeweilige Tabellenart zu berücksichtigen. Die Kennzeichnung eines formalen Parameters für eine Tabelle der Grundform erfolgt mit dem Schlüsselwort `TABLES`:

```
FORM subroutine_name TABLES itab1 itab2 ... .
```

Dabei muss die Anweisung `TABLES` vor `USING`- oder `CHANGING`-Angaben stehen. Ab Release 3.0 existieren interne Tabellen ohne Kopfzeile. Wird eine solche Tabelle als Parameter an ein Unterprogramm übergeben, so erzeugt der ABAP-Interpreter innerhalb des Unterprogramms eine Kopfzeile für diese Tabelle, was deren Bearbeitung nicht unwesentlich erleichtert.

Die ab Version 4.0 verfügbaren neuen Tabellen können nur noch als CHANGING-Parameter übergeben werden.

Wenn an ein Unterprogramm strukturierte Datentypen, also Feldleisten oder interne Tabellen übergeben werden, gehen die Strukturinformationen verloren. Lediglich für einfache Felder wird der korrekte Datentyp intern an das Unterprogramm übermittelt. Sofern die formalen Parameter eines Unterprogramms bei dessen Definition nicht typisiert werden, kann auf die Bestandteile von strukturierten Feldern nicht mehr zugegriffen werden. In älteren Versionen konnten untypisierte Felder noch als einfaches Feld vom Typ CHAR behandelt werden. In neuen, unicode-fähigen Systemen ist auch das nicht mehr möglich.

Dazu ein erstes Beispiel:

```
REPORT yz432350.

DATA: BEGIN OF fl,
  n TYPE i,
  c(10),
END OF fl.

DATA i TYPE i.

fl-n = 123.
fl-c = 'abc'.
i    = 12345.

PERFORM x USING fl i.

FORM x USING f1 f2.
  DATA: t,
        l TYPE i.

  DESCRIBE FIELD f1 TYPE t.
  DESCRIBE FIELD f1 LENGTH l IN BYTE MODE.
  WRITE: / t , l, / .

  DESCRIBE FIELD f2 TYPE t.
  DESCRIBE FIELD f2 LENGTH l IN BYTE MODE.
  WRITE: / t , l .
ENDFORM.
```

Der Datentyp des zweiten Parameters, eines einfachen Feldes, kann im Unterprogramm korrekt ermittelt werden. Die Struktur der Feldleiste hingegen ist innerhalb der Form-Routine nicht bekannt. Der Versuch, auf ein Element der Feldleiste zuzugreifen, würde eine Fehlermeldung beim Syntaxcheck erzeugen.

Durch verschiedene Optionen können Sie an Unterprogramme detaillierte Informationen über Eigenschaften von Parametern übergeben. Die Möglichkeiten zur Typisierung von Parametern wurden in der Vergangenheit bei nahezu jeder Version ergänzt. Da auch die ältesten Varianten immer noch syntaktisch korrekt sind, sollen sie hier auch noch erwähnt werden. Möglicherweise sind die entsprechenden Anweisungen in älteren Programmen oder kundenspezifischen Erweiterungen immer noch vorhanden.

Die älteste Variante beruht auf dem Schlüsselwort `STRUCTURE`. Mit dieser Anweisung können Sie einige Informationen zur Struktur von Feldleisten und internen Tabellen übergeben. Die Strukturangabe wird nach dem Parameter notiert:

```
FORM ... formal_param STRUCTURE structure.
```

Als Strukturangabe können Sie den Name einer Dictionary-Struktur (Tabelle oder Struktur), einer internen Tabelle oder einer Feldleiste benutzen. Allerdings erfolgt keine Prüfung, ob der formale Parameter wirklich diese Struktur besitzt. Überprüft wird lediglich die Länge des formalen Parameters und die Länge der Struktur. Der Parameter muss mindestens dieselbe Länge haben wie die Struktur. Ansonsten wäre es möglich, über Elemente der Struktur auf Datenbereiche zuzugreifen, die nicht mehr zum formalen Parameter gehören. Die Form-Routine des letzten Beispiels könnte also folgendermaßen erweitert werden, um den Zugriff auf die einzelnen Elemente der Feldleiste zu ermöglichen:

```
FORM x USING f1 STRUCTURE f1 f2.
...
  WRITE: / f1-n, f1-c.
```

Nicht korrekte Feldlängen bei Strukturangaben erkennt die Syntaxprüfung des ABAP-Interpreters. Dieser Fehler wird beim Prüfen eines Programms in der Entwicklungsumgebung oder vor der erstmaligen Abarbeitung erkannt. Bei der Definition von Feldleisten fügt die Steuerlogik mitunter Füllfelder ein, um Felder an Wortgrenzen auszurichten. Diese Füllfelder werden bei der Längenprüfung berücksichtigt. Mitunter kann die Summe der Feldlängen einer Struktur also größer sein als die Summe der Feldlängen der tatsächlichen Komponenten des formalen Parameters. Überprüft wird nicht die Summe der Feldlängen, sondern, wegen der eventuellen Ausrichtung der Teilfelder an Wortgrenzen, der reale Platzbedarf im Hauptspeicher.

Wie bereits erwähnt, wurden mit Release 3.0 umfangreichere Erweiterungen der Parameterprüfung wirksam. Das Schlüsselwort `STRUCTURE` wird nur noch aus Kompatibilitätsgründen mitgeführt. Es kann nur noch eingesetzt werden, um einen formalen Parameter, der eine Feldleiste sein muss, zu beschreiben. Dabei darf es sich bei der beschreibenden Struktur auch nur um eine Feldleiste handeln, interne oder Datenbanktabellen sind nicht mehr zulässig.

Statt der Anweisung STRUCTURE sollten Sie ab Release 3.0 LIKE und TYPE zusammen mit einigen anderen Optionen verwenden, um einen Parameter genauer zu beschreiben. Neben strukturierten Parametern wie Feldleisten oder internen Tabellen können auch einfache Felder geprüft werden. Welche der Optionen Sie benutzen, hängt von der Art des Vergleichsobjekts ab. Soll ein Datentyp als beschreibendes Objekt für den Parameter benutzt werden, wird TYPE eingesetzt.

```
FORM ... formal_param TYPE data_type.
```

Die Typangabe ermöglicht einige weitergehende Prüfungen der Daten, deren Umfang von der Art des beschreibenden Typs abhängt. Sofern die Typbeschreibung auf einen der vordefinierten Typen wie C oder I verweist, prüft das System lediglich auf Übereinstimmung des formalen Parameters mit dem entsprechenden Datentyp. Eine Längenprüfung findet nicht statt. Man spricht daher auch von generischer Typprüfung. Handelt es sich hingegen bei dem nach TYPE angegebene Datentyp um einen benutzerdefinierten Typ, prüft die Steuerlogik jede Komponente eines formalen Parameters sowohl bezüglich der Länge als auch des elementaren Datentyps auf Übereinstimmung mit dem jeweiligen Element der Typbeschreibung. Nicht überprüft wird allerdings die Übereinstimmung der Namen der Komponenten. Komplexe Datentypen sind nur dann identisch, wenn alle Komponenten identisch sind. Auch dazu ein Beispiel:

```
REPORT yz432360.
TYPES:
  BEGIN OF t1,
    a,b,c,
  END OF t1,

  BEGIN OF t2,
    e,f,g,
  END OF t2,

  BEGIN OF t3,
    h(3),
  END OF t3.

DATA
  a TYPE t1.

PERFORM f1 USING a.  "OK
PERFORM f2 USING a.  "OK
PERFORM f3 USING a.  "Typfehler, nach Syntaxcheck
auskommentieren

FORM f1 USING f TYPE t1.
ENDFORM.
```

```
FORM f2 USING f TYPE t2.
ENDFORM.

FORM f3 USING f TYPE t3.
ENDFORM.
```

Das gesamte Programm besteht nur aus Deklarationen, es besitzt keinerlei Funktionalität. Es muss daher auch nicht abgearbeitet werden. Von Interesse ist lediglich die Syntaxprüfung des Programms. Alle drei deklarierten Typen besitzen dieselbe Gesamtlänge. Würde in den drei Unterprogrammen die Anweisung `STRUCTURE` statt `TYPE` benutzt, wäre das Programm syntaktisch korrekt. Die durch die Typangabe veranlasste genauere Prüfung hingegen erkennt in der Form-Routine `F3` die abweichende Länge der ersten Komponente und verursacht einen Syntaxfehler.

Strukturbeschreibungen sind natürlich nicht nur für Feldleisten, sondern auch für interne Tabellen möglich. Den Einsatz der `TYPE`-Option für eine interne Tabelle der Grundform zeigt das folgende Beispiel:

```
REPORT yz432370.

TYPES:
  BEGIN OF t_line,
    zahl TYPE i,
  END OF t_line,

  t_tab TYPE t_line OCCURS 10.

DATA:
  fl TYPE t_line,
  itab TYPE t_tab.

PERFORM filltab TABLES itab USING 3.

LOOP AT itab INTO fl.
  WRITE / fl-zahl.
ENDLOOP.

FORM filltab TABLES p_tab TYPE t_tab
             USING value(p_zahl) TYPE i.
  DO 10 TIMES.
    p_tab-zahl = p_zahl.
    APPEND p_tab.
    p_zahl = p_zahl + 1.
  ENDDO.
ENDFORM.
```

Wenn Sie zur Beschreibung des Parameters keinen Datentyp, sondern ein mit DATA erzeugtes Objekt benutzen möchten, müssen Sie das Schlüsselwort TYPE durch LIKE ersetzen. Es finden identische Prüfungen statt. Das FORM-Kommando aus dem obigen Beispiel könnte also auch folgendermaßen lauten:

```
FORM filltab TABLES p_tab LIKE itab ...
```

Für eine der neuen Tabellenarten muss das Programm etwas anders aufgebaut werden. Insbesondere ist zu beachten, dass die Tabellen des neuen Typs nicht mehr nach dem Schlüsselwort TABLES übergeben werden können. Vielmehr werden Tabellen jetzt ebenso wie einfache Parameter nach USING aufgeführt, wobei Sie aber mit einem korrekten Tabellentyp typisiert werden müssen.

```
REPORT yz432380.
TYPES:
  BEGIN OF t_line,
    zahl TYPE i,
  END OF t_line,
  t_tab TYPE HASHED TABLE OF t_line
        WITH UNIQUE KEY zahl INITIAL SIZE 0.

DATA: fl TYPE t_line,
      itab TYPE t_tab.

PERFORM filltab USING itab 3.

LOOP AT itab INTO fl.
  WRITE / fl-zahl.
ENDLOOP.

FORM filltab USING p_tab TYPE t_tab
                   value(p_zahl) TYPE i.
  DATA fl TYPE LINE OF t_tab.
  DO 10 TIMES.
    fl-zahl = p_zahl.
    INSERT fl INTO TABLE p_tab.
    p_zahl = p_zahl + 1.
  ENDDO.
ENDFORM
```

Die Typprüfungen sind ab Release 3.0 so streng, dass auch zwischen Feldleisten und internen Tabellen unterschieden wird. Es ist jetzt nicht mehr möglich, eine als Parameter übergebene interne Tabelle durch eine Feldleiste zu beschreiben oder umgekehrt. Die beiden Kommandos

```
FORM filltab TABLES p_tab TYPE t_line ...
```

oder

```
FORM filltab TABLES p_tab LIKE fl ...
```

sind somit nicht korrekt und werden bei der Syntaxprüfung eines Programms beanstandet. In der täglichen Praxis wird es mitunter erforderlich, an ein Unterprogramm einen einzelnen Datensatz einer internen Tabelle zu übergeben. Falls für diesen Datensatz kein eigener Feldleistentyp zur Verfügung steht, kann mit den Optionen

```
... formal_param TYPE LINE OF table_type
```

oder

```
... formal_param LIKE LINE OF itab
```

eine Feldleiste auch durch eine interne Tabelle oder einen entsprechenden Tabellentyp beschrieben werden.

Falls notwendig, können Sie in Ausnahmefällen die Typprüfung mit

```
... TYPE ANY
```

abschalten. Es ist dann jeder Parametertyp zulässig. Analog dazu prüft

```
... TYPE TABLE
```

nur, ob der Parameter eine interne Tabelle ist oder nicht. In beiden Fällen sind dann natürlich keine Informationen über eine eventuelle Strukturierung des Parameters verfügbar.

Lokale Daten

In Unterprogrammen können lokale Datenobjekte und lokale Datentypen deklariert werden. Zur Deklaration können alle bereits beschriebenen Anweisungen (`DATA`, `TYPE`, `FIELD-SYMBOLS`, `CONSTANTS`) benutzt werden. Die so angelegten Elemente sind nur innerhalb des jeweiligen Unterprogramms gültig. Sollte in Unterprogrammen ein Feld angelegt werden, das mit gleichem Namen auch global existiert, gilt innerhalb des Unterprogramms das lokale Feld. Das globale Feld ist von den Aktionen im Unterprogramm nicht betroffen. Auf alle globalen Felder, die nicht durch gleichnamige lokale Felder verdeckt sind, kann weiterhin aus dem Unterprogramm heraus zugegriffen werden. Die Inhalte der mit `DATA` oder `CONSTANTS` angelegten lokalen Felder gehen beim Verlassen der Routine verloren. Diese Felder werden bei jedem Aufruf der Routine neu erzeugt und initialisiert. Neben den vollwertigen lokalen Datendeklarationen kann mit der Anweisung

```
LOCAL field.
```

bzw.

```
LOCAL: field_1, field_2, ... field_n.
```

ein globales Feld gegen unbeabsichtigte Veränderung geschützt werden. Diese Anweisung ist nur nach einer FORM-Anweisung sinnvoll. Sie bewirkt, dass der Inhalt des oder der Felder unmittelbar nach Eintritt in die Routine gerettet und nach Beendigung der Routine wiederhergestellt wird. Da inzwischen leistungsfähigere Möglichkeiten zur Erzeugung lokaler Daten zur Verfügung stehen, ist die Anweisung LOCAL nicht mehr zeitgemäß.

Mit Release 3.0 wird eine zusätzliche Deklarationsanweisung eingeführt, mit der in Unterprogrammen Felder mit lokaler Sichtbarkeit, aber statischer Gültigkeit erzeugt werden können. Diese Felder werden beim erstmaligen Eintritt in eine Routine erzeugt und initialisiert, bleiben aber nach dem Verlassen der Routine erhalten. Der Zugriff auf diese Felder ist nur innerhalb des Unterprogramms möglich. Erzeugt werden derartige Felder mit der Anweisung

```
STATICS field.
```

Es können, ebenso wie mit DATA, Felder, Feldleisten und interne Tabellen angelegt werden. Das folgende Beispiel demonstriert den Unterschied zwischen normalen und statischen Feldern.

```
REPORT YZ432390.

DO 10 TIMES.
  PERFORM fstat.
ENDDO.

FORM fstat.
DATA    n1 TYPE I.
STATICS n2 TYPE I.
  WRITE: / n1, n2.
  ADD 1 TO n1.
  ADD 1 TO n2.
ENDFORM.
```

3.2.13 Makros

Einige der bisher beschriebenen Anweisungen, vor allem die Deklarationen von internen Tabellen u. Ä., erfordern viel Schreibarbeit. Diese kann durch so genannte Makros etwas verringert werden. Ein Makro ist ein mit einem Namen versehener Anweisungsblock, in dem einige Platzhalter stehen können. Im Programm werden die Makros dann aufgerufen, wobei aktuelle Parameter übergeben werden. Bei der Generierung einer Anwendung werden die Makros aufgelöst und daher die Makroaufrufe durch den kompletten Quelltext ersetzt. Ein Makro wird durch die Anweisungen

```
DEFINE macro.
  statements
END-OF-DEFINITION.
```

definiert. In den Anweisungen können die Zeichen `&1` bis `&9` als Platzhalter benutzt werden. Ein Beispiel für ein Makro und dessen spätere Benutzung ist:

```
DEFINE decl_st_tab.
  DATA: BEGIN OF &1 OCCURS 0.
    INCLUDE STRUCTURE &2.
  DATA: END OF &1.
END-OF-DEFINITION.
...
DEFITAB i_aktien yzzakt.
DEFITAB i_kurse yzzkurs.
```

Das Makro `DEFITAB` erleichtert die Deklaration interner Tabellen der Urform, die von einer Dictionary-Tabelle oder einer Dictionary-Struktur abgeleitet wird. Bei der Benutzung des Makros sind nun nur noch die beiden variablen Parameter im Quelltext zu notieren.

3.2.14 Datenaustausch zwischen Anwendungen

Unter ABAP verfügt jede Anwendung, also jede Transaktion und jede der später noch zu beschreibenden Funktionsgruppen, über einen eigenen Datenbereich im Hauptspeicher. Die dort abgelegten Daten sind für andere Anwendungen nicht ohne weiteres zugänglich. In vielen Fällen ist es jedoch wünschenswert, dass Anwendungen Daten gemeinsam nutzen können. Das ist beispielsweise dann interessant, wenn mittels externer Unterprogrammaufrufe Hilfsroutinen aus anderen Reports benutzt werden. Bei der Programmierung von Anwendungen stehen dem Programmierer daher drei Möglichkeiten zur Verfügung, Daten, die im Hauptspeicher abgelegt sind, über Programmgrenzen hinweg zu nutzen. Welche Variante angewendet wird, hängt vom Programmtyp und der Verknüpfung der beteiligten Anwendungen ab.

Common Part

Ein so genannter Common Part wird mit einer speziellen Variante der `DATA`-Anweisung erzeugt. Er dient zum Datenaustausch von Anwendungen, die über externe Unterprogrammaufrufe miteinander verbunden werden. Ein Common Part enthält beliebige Datendeklarationen. Ein Programm kann mehrere solcher Bereiche enthalten, die dann einen eindeutigen Namen erhalten müssen. Wird nur ein Common-Bereich benutzt, kann der Name entfallen.

```
DATA: BEGIN OF COMMON PART [name],
  declarations
END OF COMMON PART.
```

Der gemeinsam genutzte Datenbereich muss in beiden Programmen mit identischem Namen und identischer Struktur erzeugt werden. Dies ist am einfachsten, wenn die Deklaration eines solchen Bereiches in eine eigenständige Datei aufgenommen wird. Sie erhält den Programmtyp I für *Include-Programm* und ist mit der Anweisung

```
INCLUDE filename.
```

in alle beteiligten Anwendungen einzubinden.

Get/Set-Parameter

Die Get/Set-Parameter sind dazu gedacht, einzelne Werte über Anwendungsgrenzen hinweg aufzubewahren und vor allem automatisch in Eingabefelder von Dynpros zu übertragen. Die `Get/Set`-Parameter erhalten einen bis zu zwanzigstelligen Bezeichner. Diese Bezeichner werden in der Systemtabelle `TPARA` definiert. Die Pflege dieser Tabelle ist über die Transaktion `SM32` möglich, allerdings nicht besonders komfortabel. Bequemer ist die Erstellung innerhalb des Object Navigators. Am Root-Knoten der Objektliste, also meist dem Namen des Pakets, kann per Kontextmenü über die Funktion ANLEGEN | WEITERE | SET/GET-PARAMETER-ID ein solcher Parameter angelegt, wobei Sie in einigen Popups ggf. den beschreibenden Kurztext sowie die Transporteigenschaften spezifizieren müssen (siehe Abbildung 3.34).

In einem Programm kann mit der Anweisung

```
SET PARAMETER ID parameter FIELD field.
```

ein Parameter mit einem Wert, meist einem Feldinhalt, gefüllt werden. Der Inhalt des Parameters bleibt so lange erhalten, bis sich der Anwender vom System abmeldet, also auch über Anwendungsgrenzen hinweg. Der Inhalt des Parameters kann nun mit der Anweisung

```
GET PARAMETER ID parameter FIELD field.
```

aus dem Hauptspeicher gelesen und einem Feld zugewiesen werden. Weiterhin ist es möglich, bei der Gestaltung eines Dynpros den einzelnen Feldern einen solchen Parameter zuzuweisen und unabhängig voneinander den Get- und/oder Set-Mechanismus zu aktivieren. Die Übertragung von Werten erfolgt dann automatisch. Auf diese Variante wird im Abschnitt über dialogorientierte Anwendungen genauer eingegangen.

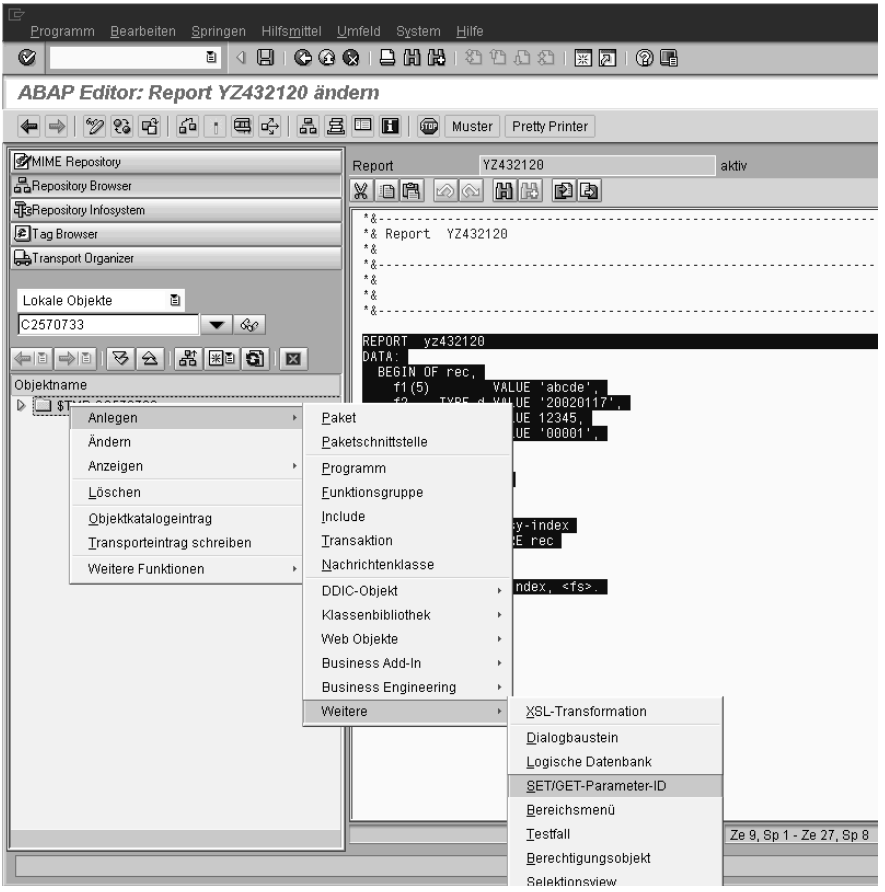

Abbildung 3.34 **© SAP AG**
Erzeugen von Get/Set-Parametern über das Kontextmenü eines Pakets

Globales Memory

Aus einem Report oder einer dialogorientierten Anwendung heraus können verschiedene Elemente (Transaktionen, Funktions- oder Dialogbausteine) mit dem Kommando `CALL` aufgerufen werden. Diese laufen dann wie ein Unterprogramm ab. Innerhalb einer solchen mit `CALL` aufgerufenen Anwendungskette können Feldinhalte oder interne Tabellen mit dem Kommando

```
EXPORT { field | itab } | (itab) TO MEMORY.
```

in einen globalen Speicherbereich gestellt werden. Dabei ist es ebenso möglich, auf eine hart kodierte Elementliste zu verzichten und stattdessen die Namen der zu übertragenden Elemente in einer internen Tabellen zu übergeben.

Mit der Anweisung

```
IMPORT { field | itab } | (itab) FROM MEMORY.
```

können die Daten aus dem Speicherbereich gelesen werden. Das Kommando EXPORT wirkt überschreibend, es gibt also kein Anfügen an den Inhalt des Speicherbereichs. Diese Einschränkung kann umgangen werden, indem die in den Speicherbereich gestellten Elemente einen Namen erhalten. Dieser wird mit dem Zusatz

```
... ID name
```

angegeben. Die namenlose Variante und die mit Namen versehenen Elemente können gemeinsam benutzt werden. Der Speicherbereich wird freigegeben, wenn das oberste Element der mit CALL verknüpften Anwendungen verlassen wird.

Transaktionsübergreifender Anwendungspuffer

Neben dem anwendungsübergreifenden Datenkontext existiert ein weiterer Puffer, der von allen Anwendungen eines Applikationsservers verwendet werden kann. Es handelt sich dabei um eine Weiterentwicklung der Kommandos EXPORT TO DATABASE und IMPORT FROM DATABASE. Die neuen Kommandos und die verwendbaren Optionen entsprechen denen der DATABASE-bezogenen Funktionen, nur dass anstelle von DATABASE das Schlüsselwort SHARED BUFFER tritt. Die Syntaxbeschreibung dieser Kommandos lautet somit

```
EXPORT { field | itab } | (itab)
  TO SHARED BUFFER table(area) ...
IMPORT { field | itab } | (itab)
  FROM SHARED BUFFER table(area) ...
DELETE { field | itab } | (itab)
  FROM SHARED BUFFER table(area) ...
```

Obwohl in den Hauptspeicher geschrieben wird, muss eine Tabelle table mit vorgegebener Struktur existieren. Deren Aufbau wurde bereits bei den DATABASE-bezogenen Funktionen erläutert.

3.2.15 Berechtigungen prüfen

In vielen Fällen soll nicht jeder Anwender alle vorhandenen Anwendungen bzw. alle Funktionen einer Anwendung ausführen dürfen. Sehr oft wird auch bezüglich der Zugriffsarten auf Daten differenziert. Das Ändern von Daten ist oft nur einem kleineren Anwenderkreis möglich als das Anzeigen. Die Genehmigung von Zugriffen auf Daten und Programme muss durch explizite Berechtigungsprüfungen realisiert werden. Als Voraussetzung dazu werden im Benutzerstammsatz so genannte Berechtigungen hinterlegt. Diese Berechtigungen stellen Ausprägungen eines Berechtigungsobjekts dar.

Wie Berechtigungsobjekte und Berechtigungen angelegt werden, geht aus Kapitel 6.2 zum Berechtigungskonzept hervor. Zum Verständnis für die Beschreibung der Prüfungen nur folgendes: Ein Berechtigungsobjekt ist eine Vorlage (Template, Typbeschreibung) für eine Berechtigung. Ein solches Objekt enthält bis zu zehn Berechtigungsfelder. Von einem Berechtigungsobjekt können Sie beliebig viele Berechtigungen abgeleiten und deren Feldern individuelle Werte zuordnen. In einem Benutzerstammsatz hinterlegen Sie die Berechtigungen mit ihren konkreten Feldwerten. Bei der Prüfung einer Berechtigung müssen Sie allerdings das zu prüfende Berechtigungsobjekt angeben. Das System ermittelt dann, welche konkreten, von diesem Objekt abgeleiteten Berechtigungen der aktuelle Anwender besitzt und prüft gegen die in diesen Berechtigungen enthaltenen Feldwerte. Zur Programmierung der Berechtigungsprüfung ist also nur die Kenntnis der Berechtigungsobjekte und der darin enthaltenen Felder erforderlich. Falls für eine Anwendung neue Berechtigungsobjekte erforderlich werden, liegt es in der Verantwortung des Programmierers, diese anzulegen und zu dokumentieren. Ohne eine Dokumentation kann der Systemadministrator keine Berechtigungen für die Anwender anlegen. Er muss wissen, welchen Feldern aus welchen Objekten Werte zugewiesen werden müssen, um den Zugriff auf bestimmte Daten oder Programmzweige zu ermöglichen.

Die Prüfung einer Berechtigung in einem Programm erfolgt mit der Anweisung

```
AUTHORITY-CHECK OBJECT authorization_object
  ID authorizationfield_1 FIELD value_1
  ...
  ID authorizationfield_x DUMMY
  ...
  ID authorizationfield_10 FIELD value_10.
```

Alle Werte können entweder als Direktwert, der in einfache Anführungsstriche einzuschließen ist, oder als Feld mit dem entsprechenden Inhalt übergeben werden. Die Anweisung ermittelt zuerst die auf dem angegebenen Berechtigungsobjekt beruhende Berechtigung des aktuellen Anwenders. Diese Berechtigung enthält ein oder mehrere Berechtigungsfelder, deren Namen jeweils nach dem Zusatz `ID` eingetragen werden. Diese Felder nehmen einen oder mehrere Werte oder Bereichsangaben auf. Nun prüft das System, ob der in der Anweisung eingetragene Prüfwert (Feld oder Konstante) im Berechtigungsfeld enthalten ist. Wenn in einem Berechtigungsobjekt mehrere Felder enthalten sind, werden die einzelnen Prüfergebnisse UND-verknüpft. Es ist möglich, dass ein Anwender mehrere Berechtigungen besitzt, die von demselben Objekt abgeleitet wurden. In diesem Fall werden die Prüfergebnisse feldweise ODER-verknüpft. Falls ein Berechtigungsfeld nicht geprüft werden soll, wird statt des `FIELD`-Zusatzes der Zusatz `DUMMY` benutzt.

Das `AUTHORITY-CHECK`-Kommando liefert das Prüfergebnis im Systemfeld `SY-SUBRC` zurück. Dabei können verschiedene Prüfergebnisse unterschieden werden. Tabelle 3.22 zeigt die möglichen Rückgabewerte.

Rückgabewert	Bedeutung
0	Berechtigung vorhanden.
4	Berechtigung nicht vorhanden.
8	Anzahl der Parameter zu groß.
12	Keine Ausprägung (Berechtigung) zum gewünschten Objekt im Benutzerstamm vorhanden.
24	Berechtigungsfelder nicht im Berechtigungsobjekt vorhanden.
28, 32, 36	Berechtigungen im Benutzerstamm sind zerstört.

Tabelle 3.22
Rückgabewerte der Berechtigungsprüfung

Es liegt in der Verantwortung des Programmierers, das Prüfergebnis auszuwerten. Das Kommando liefert lediglich einen Rückgabewert, es führt bei negativem Ausgang der Prüfung nicht automatisch zu einem Programmabbruch oder Ähnlichem. Da die Reaktion der Anwendung auf eine nicht vorhandene Berechtigung frei programmiert werden kann, ist diese Form der Berechtigungsprüfung sehr flexibel.

Die Anwendung der Berechtigungsprüfung soll an einem kleinen Beispiel etwas näher erläutert werden. Angenommen, dass in einer Anwendung in einem Dynpro die Möglichkeit geboten wird, zu dem gerade bearbeiteten Datensatz zusätzliche Informationen anzuzeigen. Dies könnte beispielsweise durch ein Popup erfolgen, das per Drucktaste aufgerufen wird. Im `OK-CODE`-Modul würden dann Anweisungen ähnlich dem folgenden Listing stehen:

```
CASE ok-code.
  WHEN 'DETA'.
    CALL SCREEN 210.
  ...
ENDCASE.
```

Vor dem Aufruf des Popups müsste die Berechtigungsprüfung erfolgen. Dazu ist ein Berechtigungsobjekt erforderlich, über dessen Feld(er) die konkreten Berechtigungen vergeben werden. Dieses Objekt soll `Y_ZZDEMO` heißen und das Feld `YPOPUP` enthalten. Wird nun vor dem `CALL SCREEN`-Befehl eine Berechtigungsprüfung ausgeführt, könnte das so aussehen:

```
CASE ok-code.
  WHEN 'DETA'.
    AUTHORITY-CHECK OBJECT 'Y_ZZDEMO'
      ID 'YPOPUP' FIELD 'X'.
```

```
    IF sy-subrc = 0.
      CALL SCREEN 210.
    ELSE.
      MESSAGE E999.
*        Sie haben keine Berechtigung für diese Funktion
    ENDIF.
  ...
ENDCASE.
```

Die Berechtigungsprüfung liefert nur dann den Rückkehrcode 0, wenn das Feld `YPOPUP` in der von `Y_ZZDEMO` abgeleiteten Berechtigung mit einem Wert oder Muster gefüllt ist, der zum Prüfwert passt. Da in der Berechtigung nur eine Ja-/Nein-Information abgelegt wird, bietet es sich an, als Freigabewert den Buchstaben „X“ zu benutzen.

Die Berechtigungsprüfung kann aber auch anders erfolgen. Es ist recht aufwändig, für jeden Anwender eine eigene Berechtigung zu pflegen und in dieser jeweils das Flag zu setzen oder zu löschen. Die Berechtigungsprüfung erlaubt auch eine andere Vorgehensweise. Im Berechtigungsfeld könnten z.B. auch die Namen der Anwender stehen, die zum Aufruf der zusätzlichen Funktion berechtigt sind. Es existiert dann nur noch eine leicht zu pflegende Berechtigung für alle potenziellen Anwender. Allerdings stellt hier die Zahl der im Berechtigungsfeld einzutragenden Anwender eine Grenze für die praktikable Anwendung dieser Art der Berechtigungsprüfung dar, wodurch sich diese Variante der Berechtigungsvergabe nur für einen kleinen Kreis von Anwendern eignet. Die Prüfung im Programm wird nur geringfügig modifiziert.

```
CASE ok-code.
  WHEN 'DETA'.
    AUTHORITY-CHECK OBJECT 'Y_ZZDEMO'
      ID 'YPOPUP' FIELD SY-UNAME.
    IF sy-subrc = 0.
      CALL SCREEN 210.
    ELSE.
      MESSAGE E999.
*        Sie haben keine Berechtigung für diese Funktion
    ENDIF.
  ...
ENDCASE.
```

Falls bei dieser Variante der Prüfung im Feld der Berechtigung ein Stern eingetragen wird, besitzen alle Anwender die erforderliche Berechtigung, da der Stern als Musterzeichen fungiert.

3.2.16 Nachrichten

Nachrichten erfüllen eine Doppelfunktion. Sie nehmen in fast allen Fällen Einfluss auf den Programmablauf, meist in der Form, dass die Bearbeitung des aktuellen Dynpros oder Selektionsbildschirmes von vorn begonnen wird. Außerdem liefern sie dem Anwender eine Information, in der auf einen Fehler hingewiesen wird.

Nachrichten im engeren Sinne sind spezielle nummerierte Textelemente. Sie werden über zwei Felder identifiziert: über die Nachrichtenklasse (zwanzigstellig, alphanummerisch) und die Nachrichtennummer (dreistellig, nummerisch). Zu jeder Nachricht kann neben dem einzeiligen Kurztext ein umfangreicher Langtext erfasst werden. Die Nachrichtentexte können bis zu vier Platzhalterzeichen `&` enthalten, die ähnlich wie in den bereits beschriebenen Titelelementen behandelt werden. Auch die Platzhalter in Nachrichten können indiziert werden (`&1` bis `&4`).

Erstellung

Gepflegt werden Nachrichten über die Transaktion `SE91` bzw. die Funktion aus dem Kontextmenü eines Entwicklungs Pakets oder die Menüfunktion ENTWICKLUNG | PROGRAMMIERUMFELD | NACHRICHTEN. Das Startbild der Transaktion `SE91` erlaubt Ihnen, eine neue Nachrichtenklasse anzulegen. Die anderen Funktionen blenden ein Popup ein, in dem Sie zunächst den Namen der anzulegenden Nachrichtenklasse erfassen müssen (siehe Abbildung 3.35). Verwenden Sie für dieses Beispiel die Nachrichtenklasse `YZ4`.

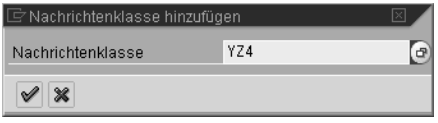

Abbildung 3.35 © SAP AG
Abfrage des Namens der Nachrichtenklasse

Anschließend gelangen Sie in die Pflegeoberfläche der Nachrichtenklasse (Abbildung 3.36).

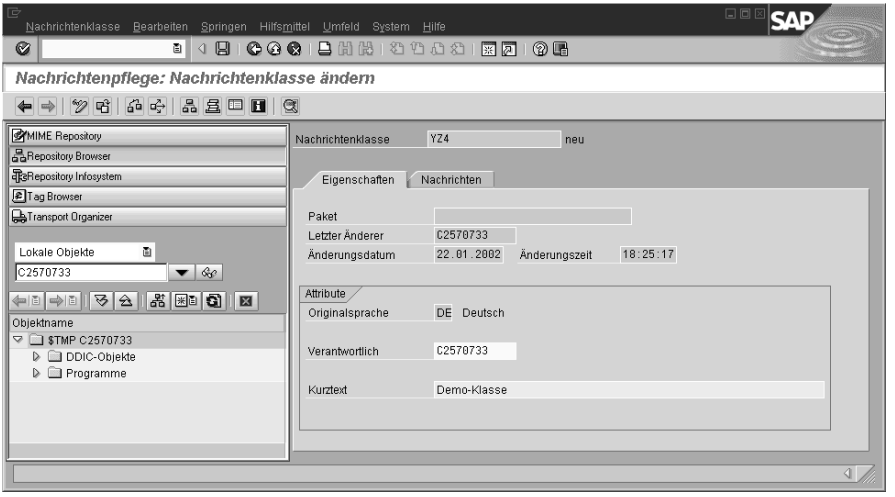

Abbildung 3.36 **© SAP AG**
Pflege der Elemente einer Nachrichtenklasse

Diese Pflegeoberfläche ermöglicht – verteilt auf zwei Karteikarten – die Pflege der Attribute der Nachrichtenklasse an sich und der einzelnen Nachrichten (Abbildung 3.37).

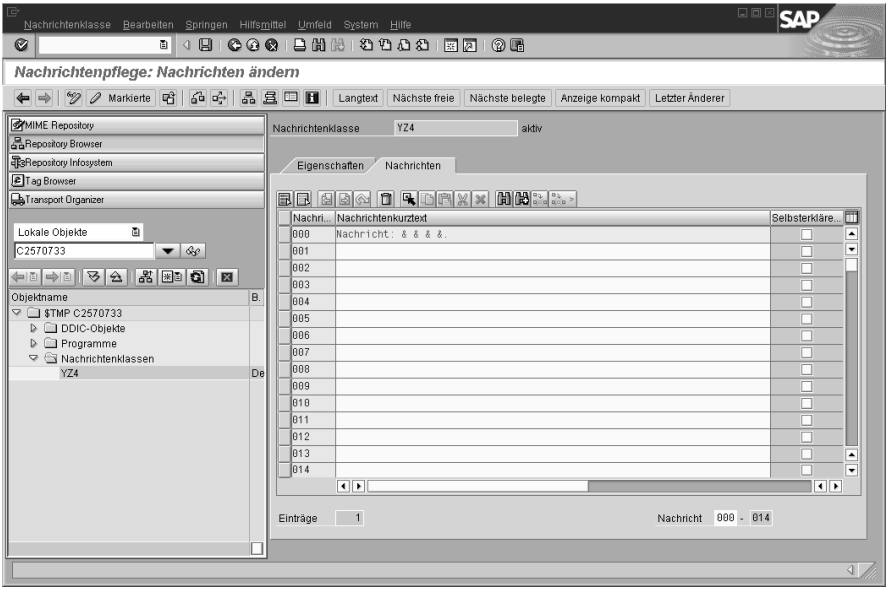

Abbildung 3.37 **© SAP AG**
Pflege der einzelnen Nachrichten

Das Pflegebild der Einzelnachrichten befindet sich zunächst im Anzeigen-Modus. Durch die Menüfunktion NACHRICHTENKLASSE | ANZEIGEN<–>ÄNDERN oder das entsprechende Symbol in der Drucktastenzeile werden die Nachrichtentexte pflegbar.

Tragen Sie den im Bild ersichtlichen Text ein. Diese Nachricht kann als Dummy benutzt werden, um beliebige Texte auszugeben. Das Flag am Ende des Nachrichtentextes legt fest, ob zur Nachricht ein Langtext erfasst werden kann. Dieser Langtext erläutert den aufgetretenen Fehler und gibt Hinweise zur Fehlerbehebung. Derartige Langtexte werden relativ häufig verwendet. Speichern Sie zunächst den soeben eingetragenen Nachrichtentext ab. Betätigen Sie dann die Schaltfläche LANGTEXT oder Ctrl-F7 in der Drucktastenzeile. Das System ruft nun den so genannten SAPScript-Editor auf (Abbildung 3.38). Mehr zur Bedienung dieses Editors erfahren Sie im Abschnitt 3.11 zur Formularverarbeitung. Im Moment reicht es aus, die vier im Bild ersichtlichen Zeilen einzutragen. Falls Sie neue Zeilen benötigen, können Sie diese problemlos durch Betätigen der ↵-Taste einfügen.

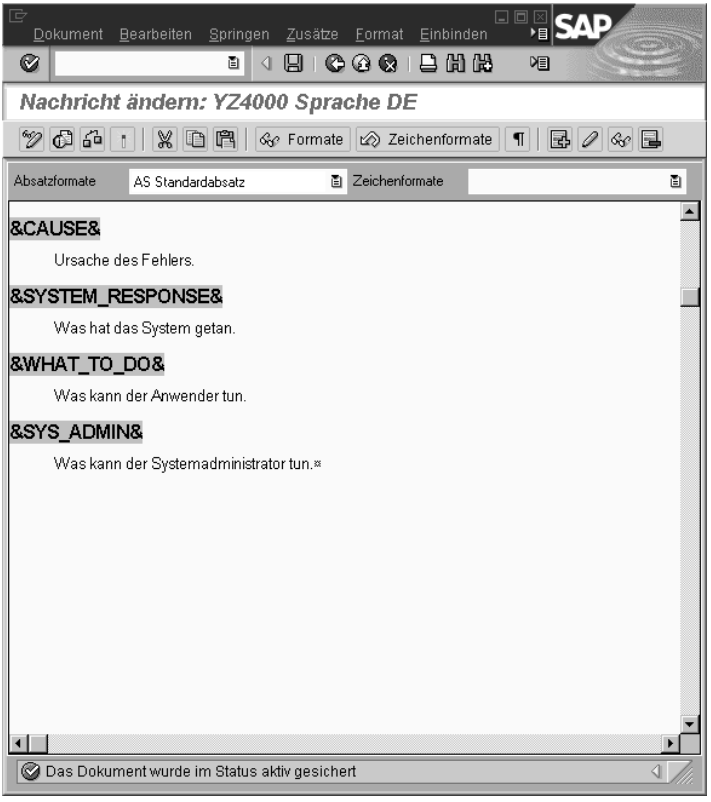

Abbildung 3.38 **© SAP AG**
Langtext einer Nachricht pflegen

Nach Eingabe der drei Zeilen speichern Sie den Text mit der Menüfunktion DOKUMENT | SICHERN | IN AKTIVER FASSUNG ab und kehren zum Pflegebild der Einzelnachrichten zurück. Die Bearbeitung der Nachricht ist damit abgeschlossen. Ohne Aktivierung werden Änderungen am Text nicht wirksam. Alternativ zur erwähnten Speicher-Funktion können Sie auch das Symbol zum Aktivieren in der Drucktastenzeile nutzen.

Wenn die Nachricht später ausgelöst wird, erscheint der Kurztext je nach Aufrufvariante in der Statuszeile oder einem Popup. Durch einen Doppelklick auf die Statuszeile oder eine Drucktaste im Popup erscheint der entsprechende Langtext (Abbildung 3.39). Sowohl in den Kurz- als auch den Langtext werden die Parameter übernommen.

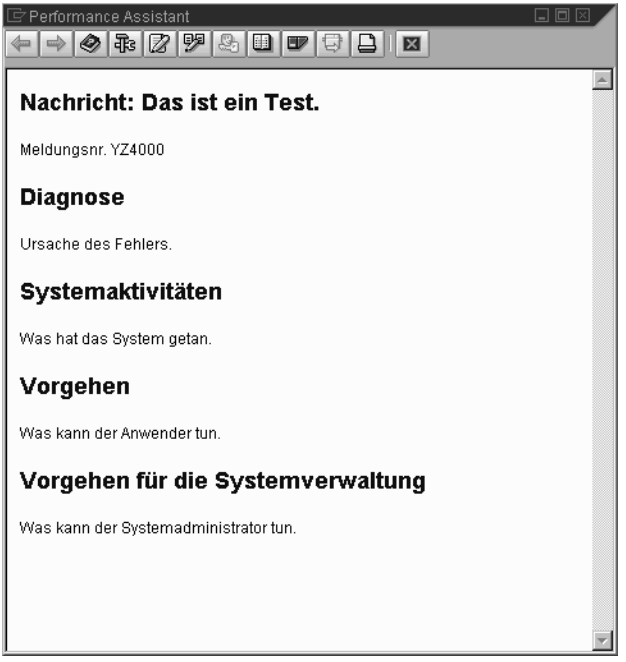

Abbildung 3.39 **© SAP AG**
Langtext der Nachricht in einer Anwendung

Einsatz von Nachrichten in der Anwendung

In einer Anwendung wird mit dem Kommando

```
MESSAGE ID class TYPE type NUMBER number.
```

eine Nachricht ausgesendet. Alle drei Parameter müssen in einfachen Anführungsstrichen stehen. Neben der Nachrichtenklasse und der Nachrichtennummer ist in diesem Kommando auch noch der Nachrichtentyp anzugeben. Für

diesen Typ existieren einige systeminterne Vorgaben, er muss also, im Gegensatz zu Nachrichtentext und Nachrichtenklasse, nicht gepflegt werden. Er ist keine Eigenschaft spezieller Nachrichten, so wie z.B. den Funktionscodes ein Funktionstyp zugeordnet werden kann. Es handelt sich beim Nachrichtentyp vielmehr um einen Kennbuchstaben, der angibt, wie die Nachricht durch das System behandelt werden soll. Der Nachrichtentyp hat Einfluss auf den Programmablauf und die Form der Anzeige der Nachricht. Tabelle 3.23 zeigt die möglichen Nachrichtentypen.

Typ	Bezeichnung	Anzeige in	Wirkung
I	Info	Popup	Programmfortsetzung nach Bestätigung (Drucktaste oder Datenfreigabe).
W	Warning	Statuszeile	Fortsetzen der Abarbeitung nach Datenfreigabe (↵-Taste).
E	Error	Statuszeile	Erneute Abarbeitung des Dynpros.
A	Abend	Popup	Abbruch der Transaktion.
S	Success	Statuszeile	Nur Anzeige des Textes.

Tabelle 3.23
Nachrichtentypen

Der am häufigsten verwendete Nachrichtentyp ist der Typ `E`. Er zeigt den Kurztext der Nachricht in der Statuszeile an und veranlasst die erneute Abarbeitung des Dynpros. Er wird benutzt, um in einem Dynpro auf fehlerhafte Eingaben hinzuweisen. Innerhalb von Dialoganwendungen wird eine Nachricht immer in einem Modul oder in Unterprogrammen, die von einem Modul aus aufgerufen wurden, ausgelöst. Falls dieses Modul durch die `FIELD`-Anweisung oder eine `CHAIN`-Kette mit explizit benannten Feldern verbunden wurde, werden bei der durch die Nachricht veranlassten erneuten Abarbeitung eines Dynpros nur diese Felder eingabebereit.

Etwas weniger rigoros in seiner Wirkung ist der Nachrichtentyp W. Er weist, wie der Typ E, in der Statuszeile auf einen Fehler hin. Allerdings kann die Bearbeitung des Dynpros nach der Bestätigung der Warnung mit der Datenfreigabetaste fortgesetzt werden. Ähnlich arbeitet der Typ I, der die Nachricht nicht in der Statuszeile, sondern in einem eigenen Popup darstellt.

Der Nachrichtentyp S zeigt nur eine Nachricht in der Statuszeile an. Da er den Programmablauf nicht beeinflusst, erscheint er erst nach Abarbeitung des aktuellen Dynpros in der Statuszeile des Folge-Dynpros. Mit Nachrichten des Typs S informiert die Anwendung den Anwender meist über die erfolgreiche Ausführung einer Aktion, z.B. „Datensatz erfolgreich gesichert".

Eine Ausnahme stellt der Nachrichtentyp A dar. Er bricht die laufende Transaktion ab. Dieser Nachrichtentyp wird benutzt, um bei schwerwiegenden Problemen, z. B. Inkonsistenzen im Datenbestand, die weitere Arbeit zu unterbinden. Der Nachrichtenkurztext wird wiederum in der Statuszeile angezeigt.

Falls ein Nachrichtenkurztext in der Statuszeile angezeigt wird, bringt ein Doppelklick auf den Text den Langtext in einem Popup zur Anzeige. Beim Typ I steht dazu eine Drucktaste im Popup zur Verfügung.

Neben der Langform des MESSAGE-Kommandos steht auch eine etwas leichter zu handhabende Kurzform zur Verfügung. Sie lautet

```
MESSAGE tnnn(kk).
```

Dabei steht *t* für den Nachrichtentyp, *nnn* für die Nummer und *kk* für die Nachrichtenklasse. Typ und Nummer stehen ohne Trennzeichen und ohne Anführungsstriche. Die Nachrichtenklasse ist in runde Klammern einzuschließen, die ebenfalls ohne Trennzeichen auf die Nachrichtennummer folgen. Die Angabe der Nachrichtenklasse kann entfallen, wenn in der PROGRAM- bzw. REPORT-Anweisung mit dem Zusatz

```
... MESSAGE-ID kk.
```

eine Vorgabe für die gesamte Anwendung gesetzt wird. Der Name der Nachrichtenklasse darf dabei nicht in Anführungsstrichen stehen. Im Programm muss die Nachrichtenklasse nur dann angegeben werden, wenn eine von der Standardvorgabe abweichende Klasse benutzt werden soll.

Falls die bereits erwähnten Platzhalter der Nachricht zur Laufzeit mit Werten gefüllt werden sollen, sind diese mit dem Zusatz

```
... WITH value_1 value_2 value_3 value_4.
```

an das MESSAGE-Kommando zu übergeben.

Weitere Optionen zum MESSAGE-Kommando existieren im Zusammenhang mit Funktionsbausteinen. Dort können mit dem MESSAGE-Kommando so genannte Ausnahmen ausgelöst werden. Diese Ausnahmen führen nicht unbedingt zum den oben beschriebenen Ereignissen wie dem Neustart des Dynpros. Vielmehr beenden sie lediglich die Abarbeitung des Funktionsbausteins. Das aufrufende Programm wird von der Fehlersituation informiert und kann selbst entscheiden, wie darauf zu reagieren ist. Beispielsweise könnte es selbst die erforderliche Nachricht aussenden. Dazu werden durch Aufruf des Kommandos MESSAGE die Systemfelder SY-MSGID mit der Nachrichtenklasse, SY-MSGTY mit dem Nachrichtentyp, SY-MSGNO mit der Nachrichtennummer sowie SY-MSGV1 bis SY-MSGV4 mit den Parametern gefüllt. Auf diese Felder kann das rufende Programm zurückgreifen.

Laufzeitfehler abfangen

Viele Kommandos lösen in Fehlersituationen Laufzeitfehler aus, die eine Anwendung mit einem Shortdump beenden. Ab Version 4.0 können ausgewählte Laufzeitfehler abgefangen werden. Damit wird es möglich, die Reaktion der Anwendung auf einen Fehler selbst zu bestimmen.

Um in einer Folge von Anweisungen Laufzeitfehler abzufangen, müssen diese Anweisungen in einen Block eingeschlossen werden, der mit der Anweisung

```
CATCH SYSTEM-EXCEPTIONS ...
```

beginnt und mit

```
ENDCATCH.
```

endet. In der Parameterliste der `CATCH`-Anweisung müssen die abzufangenden Laufzeitfehler aufgeführt werden. Dies erfolgt durch Aufzählung der jeweiligen Laufzeitfehler und Zuordnung eines Rückgabewerts. Beim Auftreten eines Laufzeitfehlers wird die aktuelle Verarbeitung unterbrochen und stattdessen die Anweisung nach `ENDCATCH` ausgeführt. Vorher wird der dem Laufzeitfehler zugeordnete Rückgabewert in die Systemvariable `SY-SUBRC` gestellt. Das Verfahren ähnelt damit der Ausnahmebehandlung bei Funktionsbausteinen. Die `CATCH-ENDCATCH`-Blöcke dürfen beliebig tief geschachtelt werden.

Die vollständige Syntax des `CATCH`-Kommandos lautet

```
CATCH SYSTEM-EXCEPTIONS {runtime_error = return_code}.
```

Die Laufzeitfehler können durch drei verschiedene Methoden angegeben werden. Zunächst existieren Fehlerklassen. Eine Fehlerklasse fasst mehrere Fehler eines Themengebietes. Wenn Sie für `runtime_error` den Name einer solchen Fehlerklasse angeben, führen alle Fehler dieser Fehlerklasse zu dem mit `return_code` angegebenen Rückgabewert. Der konkrete Laufzeitfehler kann so nicht ermittelt werden. Es ist daher auch möglich, für `runtime_error` den Namen eines einzelnen Laufzeitfehlers anzugeben. Alle anderen Fehler der Klasse dieses Laufzeitfehlers werden dann aber nicht abgefangen. Als dritte Möglichkeit können alle existierenden Laufzeitfehler durch den Bezeichner `OTHERS` angegeben werden.

Beim Einsatz des `CATCH`-Kommandos sind einige Einschränkungen zu beachten. So werden Laufzeitfehler nur innerhalb der aktuellen Aufrufebene abgefangen. Die `CATCH`-Anweisung hat keinen Einfluss auf Laufzeitfehler, die innerhalb von Unterprogrammen oder Funktionsbausteinen auftreten, welche innerhalb eines `CATCH-ENDCATCH`-Blockes aufgerufen werden. Darüber hinaus existiert für jedes ABAP-Kommando ein Satz von abfangbaren Laufzeitfehlern. Andere Laufzeitfehler werden durch `CATCH` nicht behandelt. Die abfangbaren Laufzeitfehler finden Sie in der Online-Dokumentation der Schlüsselwörter.

Es ist durchaus möglich, dass zu einem Laufzeitfehler mehrere Rückgabewerte passen. Das kann z. B. dann der Fall sein, wenn ein Laufzeitfehler in verschiedenen Fehlerklassen enthalten ist oder aber in der CATCH-Anweisung sowohl die Zuordnung eines einzelnen Laufzeitfehlers als auch der gesamten Fehlerklasse erfolgt. In diesem Fall liefert das Systemfeld SY-SUBRC nur den ersten passenden Rückgabewert.

3.2.17 Sperr- und Verbuchungskonzept

Eine SAP-Anwendung ist eine Multi-User-Anwendung. Das bedeutet, dass zu einem Zeitpunkt möglicherweise mehrere Anwender gleichzeitig auf denselben Datensatz zugreifen. Der lesende Zugriff ist dabei unproblematisch. Um die Konsistenz der Daten zu gewährleisten, muss aber verhindert werden, dass mehrere Anwender einen Datensatz gleichzeitig bearbeiten. In einer Anwendung können daher einzelne Datensätze oder ganze Gruppen von Datensätzen gegen Fremdzugriff gesperrt werden. In der praktischen Anwendung würde ein Datensatz unmittelbar nach dem Lesen gesperrt und erst nach dem Zurückschreiben auf die Datenbank wieder freigegeben werden. Versuchen andere Anwender, einen bereits gesperrten Datensatz nochmals zu sperren oder einen gesperrten Datensatz zu überschreiben, führen die jeweiligen Kommandos zu einem Fehler, der über das Systemfeld SY-SUBRC mitgeteilt wird.

Das Sperren und Freigeben (Entsperren) von Datensätzen erfolgt nicht direkt durch spezielle ABAP-Kommandos, sondern durch den Aufruf automatisch generierter Funktionsbausteine. Diese Funktionsbausteine wiederum beruhen auf der Definition eines so genannten Sperrobjekts, das zu den Dictionary-Objekten gehört und in Kapitel 5.5 beschrieben wird. Zu den in diesem Abschnitt besprochenen typischen Anweisungen für dialogorientierte Anwendungen gehören aber der Aufruf dieser Funktionsbausteine und die Auswertung der Rückgabewerte. Der folgende Ausschnitt aus einem realen Programm zeigt die Anweisung zum Sperren einer Gruppe von Datensätzen:

```
...
CALL FUNCTION 'ENQUEUE_EFMHICTR'
  EXPORTING
    fikrs = g_fikrs
  EXCEPTIONS
    foreign_lock   = 1
    system_failure = 2.

CASE sy-subrc.
  WHEN 1.
    MESSAGE E645. " bereits gesperrt
  WHEN 2.
    MESSAGE A523. " Systemfehler bei einer Sperranforderung
ENDCASE.
```

Das eigentliche Schreiben von Datensätzen erfolgt natürlich mit den bereits beschriebenen Kommandos zur Datenbankarbeit, beispielsweise `MODIFY` oder `UPDATE`. Allerdings existieren einige Verfahren, die vor allem zur Verbesserung der Performance und der Übersichtlichkeit eines Programms dienen. Es ist beispielsweise möglich, Schreibvorgänge in Unterprogramme zu verlagern und diese Programme mit dem Zusatz

```
... ON COMMIT.
```

aufzurufen. So aufgerufene Unterprogramme werden erst dann ausgeführt, wenn die Anwendung die Anweisung `COMMIT WORK` aufruft, mit der Datenbankänderungen bestätigt werden. Es ist allerdings nicht möglich, dem Unterprogramm Parameter zu übergeben. Die zu schreibenden Daten müssen daher global verfügbar sein.

Auch der Aufruf bzw. die Ausführung von Funktionsbausteinen kann vom Kommando `COMMIT WORK` abhängig gemacht werden. Dazu wird der Funktionsbaustein mit dem Zusatz

```
... IN UPDATE TASK.
```

versehen. Die mit diesem Zusatz versehenen Funktionsbausteine müssen spezielle Attribute besitzen. Diese Attribute steuern beispielsweise die Art der Ausführung, die bei sehr umfangreichen Verbuchungen auch asynchron zum Programmablauf erfolgen kann. Näheres zu dieser Form des Aufrufs eines Funktionsbausteins finden Sie in Abschnitt 3.9.

Verbuchungsbausteine werden ausgeführt, wenn das Programm die Anweisung

```
COMMIT WORK.
```

abarbeitet. Beim impliziten Commit, der automatisch bei jedem Bildwechsel stattfindet, arbeitet das System die Verbuchungsbausteine nicht ab.

3.3 Die Oberfläche

Jede Anwendung, gleich ob Report oder Dialoganwendung, verfügt über eine Oberfläche mit den Bedienelementen (Menü, Drucktastenzeile…). Unter einer Oberfläche werden mehrere so genannte Status zusammengefasst. Jeder Status stellt eine eigenständige Ausprägung der Oberfläche dar. Die verschiedenen Status unterscheiden sich durch den Aufbau der Menüs und der bereitgestellten Funktionen. Innerhalb eines Status werden die Bedienelemente mit Funktionscodes verknüpft. Funktionscodes sind Bezeichner, die an die Anwendung übermittelt und dort ausgewertet werden. Zur Bearbeitung der Oberfläche bzw. der Status wird der so genannte *Menu Painter* benutzt. Dieses Werkzeug ist über den Transaktionscode `SE41` zu erreichen. Zum Anlegen eines neuen Status für eine Anwendung kann innerhalb des Object Navigator auch die Funktion ANLEGEN |

GUI-STATUS aus dem Kontextmenü benutzt werden. Ebenfalls möglich ist der Zugriff mit Hilfe des Navigationsmechanismus des Programmeditors. Wie auch bei anderen Elementen reicht es aus, in einem Programm den Namen eines Status einzutragen und auf diesen einen Doppelklick mit der Maus auszuführen.

Eine Oberfläche bzw. ein Status kann folgende Elemente enthalten:

- Menü
- Symbolleiste
- Drucktastenzeile
- Funktionstastenbelegung
- Titel

Die ersten vier Elemente sind echte Bedienelemente, der Titel spielt nur eine passive Rolle. Es existieren verschiedene Oberflächenarten, in denen nicht immer alle Elemente benutzt werden können. Innerhalb der Oberflächenpflege können Sie entweder status- oder elementorientiert arbeiten. Statusorientiert bedeutet, dass alle Elemente eines Status (Menü, Drucktasten, Funktionstasten...) gleichzeitig in einer Arbeitsoberfläche zur Verfügung stehen. Bei der elementorientierten Arbeitsweise können jeweils die Menüs oder die Funktionstastenbelegungen aller Status einer Oberfläche gemeinsam angezeigt und bearbeitet werden. Darüber hinaus sind einige Elemente (Funktionscodes, Titel) innerhalb der Oberfläche global, können aber auch von der Arbeitsoberfläche jedes Status aus gepflegt werden.

3.3.1 Die Oberflächentypen

Ein Status existiert nicht für sich allein, sondern wird immer von einem Dynpro oder einem Report aus benutzt. Beim Anlegen eines neuen Status muss daher ein Statustyp angegeben werden. Zur Zeit existieren drei Grundtypen sowie eine zusätzliche Eigenschaft.

Dialogstatus

Dieser Statustyp stellt eine Oberfläche für normale Dialoge (früher Dynpro genannt), die im Vollbild-Modus laufen, zur Verfügung. Er enthält alle erwähnten Gruppen von Bedienelementen, also ein Menü, die Symbolleiste, die Funktionstastenbelegung und die Drucktasten. Status diesen Typs werden auch für Reports benutzt. Die in älteren Versionen der R/3-Software vorhandene Unterscheidung zwischen Status für Dialoganwendungen und Reports findet an anderer Stelle statt.

Dialogfenster

Dynpros des Typs *Dialogfenster* können als Popup ausgeführt werden. Solche Popups verfügen nicht über ein Menü, sondern werden nur über die Funktions- und Drucktasten bedient. Für solche Dynpros ist ein Status mit dem Typ *Dialogfenster* anzulegen. Auch hier wird nicht mehr zwischen Dialoganwendungen und Listen unterschieden.

Kontextmenü

Kontextmenüs werden durch einen Klick der rechten Maustaste eingeblendet. Im Normalfall werden dabei alle mit Funktionstasten verbundene Funktionscodes des aktuellen Status aufgelistet. Im Zusammenhang mit dem Einsatz spezieller Controls im Dynpro und der objektorientierten Programmierung können auch spezielle Kontextmenüs erzeugt werden, die einen exakt definierten Inhalt besitzen. Innerhalb des herkömmlichen Programmiermodells spielen sie keine Rolle.

3.3.2 Funktionscodes

Die Bezeichnung Funktionscode ist etwas irreführend, denn es handelt sich um ein komplexes Element mit mehreren Eigenschaften. Ein Funktionscode wird einem Bedienelement (Menüeintrag, Schaltfläche, Funktionstaste) zugeordnet. Dabei übernimmt das Bedienelement einige Eigenschaften des Funktionscodes, z. B. den darzustellenden Text. Bei der Betätigung des Bedienelements wird der Bezeichner des Funktionscodes an die Anwendung übergeben. Die Art der Übergabe unterscheidet sich bei Reports und Dialoganwendungen.

Die Funktionscodes aller Status einer Oberfläche sind in einer gemeinsamen Liste enthalten. Damit nicht alle Funktionscodes in allen Status ausgelöst werden können, besteht die Möglichkeit, einen Funktionscode in jedem Status getrennt zu aktivieren oder zu deaktivieren. Wenn der Funktionscode, der einer Menüfunktion oder einer Drucktaste zugeordnet ist, deaktiviert wird, erscheint das entsprechende Element noch in der Oberfläche, kann aber nicht ausgelöst werden. Optisch werden deaktivierte Elemente durch eine graue statt eine schwarze Beschriftung gekennzeichnet.

Der Name eines Funktionscodes hat eine maximale Länge von 20 Stellen, wobei im Menu Painter momentan nur 10 Stellen gepflegt werden können. Die Bezeichner der Funktionscodes können freizügig gewählt werden. Sie sollten lediglich vermeiden, Funktionscodes zu verwenden, die durch das System selbst abgefangen werden. Das sind z. B. jene, die mit dem Zeichen „%“ beginnen oder auch die Funktionscodes P+, P++, P– und P––.

Ein Funktionscode besitzt verschiedene Eigenschaften. Die bedeutsamsten sind der *Funktionstyp*, die *Textart*, der *Text* und die *Direktwahl*. Der Funktionstyp be-

stimmt, wie der Funktionscode durch das System behandelt wird. Tabelle 3.24 zeigt die verfügbaren Funktionstypen. Die Textart und der Text haben Einfluss auf die Darstellung des Funktionstextes in den Menüeinträgen und Drucktasten. Einem Funktionscode kann in jedem Status ein anderer Text zugeordnet werden. Die Direktwahl definiert eine Taste, mit welcher der Funktionscode in einem aufgeklappten Pulldown-Menü direkt ausgewählt werden kann. Alle Eigenschaften können am einfachsten in der *Funktionsliste* bearbeitet werden.

Funktionstyp	Bedeutung
Leerzeichen	Voreinstellung, normaler Funktionscode.
E	Exit-Kommando (Auswertung durch ein AT-EXIT-COMMAND-Modul).
H	Interne Verwendung.
P	Lokale GUI-Funktion.
S	Systemfunktion.
T	Starten einer Transaktion.

Tabelle 3.24
Funktionstypen

Funktionscodes des Standardtyps beenden die Eingabe in ein Dynpro und starten dessen PAI-Abschnitt. Ähnlich funktionieren die Exit-Funktionscodes. Sie starten als einzige die Abarbeitung von PAI-Modulen mit dem Zusatz `AT EXIT-COMMAND`. Dabei werden die automatischen Dynproprüfungen (z.B. für Muss-Eingabefelder) nicht ausgeführt. Details dazu entnehmen Sie bitte dem Abschnitt über die Dialogprogrammierung.

Falls der direkte Sprung zu einer anderen Transaktion ausgelöst werden soll, kann dies durch den Funktionstyp T erreicht werden. Der Funktionscode wird dabei als Bezeichner der zu startenden Transaktion gewertet. Auch dabei finden keinerlei Prüfungen statt. Der Funktionstyp S ist, wenn überhaupt, nur in der Testphase einer Anwendung von Bedeutung. Mit diesem Funktionstyp werden Systemfunktionen ausgelöst, zu denen beispielsweise das Einschalten des Debuggers gehört. Der Kennbuchstabe der Systemfunktion wird dabei als Funktionscode benutzt. Der Funktionstyp H löst den Zeitpunkt `ON HELP-REQUEST` aus. Der Funktionstyp P hingegen wird nicht von der Ablauflogik des Dynpros, sondern direkt von einem Dynpro-Control (im Moment nur vom Tab Strip) verarbeitet. Von entscheidender Bedeutung sind in praktischen Anwendungen nur die Funktionstypen für normale und für Exit-Funktionen.

Für jeden Funktionscode können Sie zwei Texte erfassen. Einer der Texte wird im Menü und ggf. für die Beschriftung von Funktions- und Drucktasten verwendet. Der andere Text ist nur für Drucktasten sinnvoll. Er erscheint in einem

separaten kleinen Popup, wenn der Mauszeiger eine kurze Zeit über der Schaltfläche verharrt. Neben der Verwendung eines Textes kann für Funktionscodes, die einer Schaltfläche zugeordnet werden, auch ein Icon und wahlweise ein zusätzlicher Text gepflegt werden.

All diese Einstellungen sind statisch. Falls der Text eines Funktionscodes dynamisch zur Laufzeit modifiziert werden soll, so ist auch das möglich. Dazu muss zunächst die Textart des Funktionscodes geändert werden. Als einziges Eingabefeld steht Ihnen dann ein Feld zur Verfügung, in dem Sie den Namen eines Datenfeldes eintragen. In diesem Datenfeld muss zur Laufzeit die Bezeichnung des Funktionscodes stehen.

In einem Status können bis zu 35 Drucktasten definiert werden, die in der Drucktastenzeile allerdings keinen Platz finden, wenn sie mit Texten belegt sind. Die Darstellung als Icon ermöglicht es daher, anstelle der Texte in den Drucktasten Symbole anzuzeigen.

Falls für eine Drucktaste ein dynamischer Text vereinbart wurde, kann im entsprechenden Datenfeld zur Laufzeit auch der Identifikator für ein Icon stehen. Er erscheint dann anstelle oder zusammen mit dem eigentlichen Text der Drucktaste. Diese dynamische Zuweisung von Icons hat einen Nachteil. Der Icon-Identifikator wird nur beim Anzeigen der Drucktaste ausgewertet, nicht aber bei der Anzeige von Menüeinträgen. Sollte also ein Funktionscode, der über einen dynamischen Text verfügt, sowohl im Menü als auch in der Drucktastenzeile verwendet werden, wird ein eventuell zugewiesenes Icon nur in der Drucktaste korrekt angezeigt.

Die Bedienung des eigentlichen Menüs kann auch über die Tastatur erfolgen. Zur Beschleunigung der Auswahl werden in den Menüeinträgen einzelne Buchstaben durch einen Unterstrich gekennzeichnet. Ein so gekennzeichneter Menüeintrag kann durch die Tastenkombination ALT+Buchstabentaste gewählt werden. Mit der so genannten Direktauswahl wird im Menu Painter festgelegt, welche Buchstaben in jedem Menü zur Schnellauswahl dienen sollen.

3.3.3 Bedienung des Menu Painter

Die Funktionscodes werden durch Betätigung eines der Bedienelemente der Oberfläche ausgelöst. Die Bedienelemente können jeweils einer von vier Gruppen zugeordnet werden:

- Menü,
- Drucktastenzeile,
- Symbolleiste,
- Funktionstasten.

Der Inhalt der Symbolleiste ist vorgegeben, er kann nicht verändert werden. Allerdings können alle Symbole außer der Datenfreigabetaste beliebig aktiviert

oder deaktiviert werden. Dies erfolgt durch die Zuordnung von Funktionscodes zu speziellen Funktionstasten oder direkt zu den Symbolen. Die konkrete Variante ist von der SAP-Version abhängig.

Der Menu Painter kann, wie bereits erwähnt, auf unterschiedliche Art und Weise aufgerufen werden. Das Einstiegsbild der Transaktion SE41 (Abbildung 3.40) ähnelt der Oberfläche anderer Pflegeprogramme der Entwicklungsumgebung.

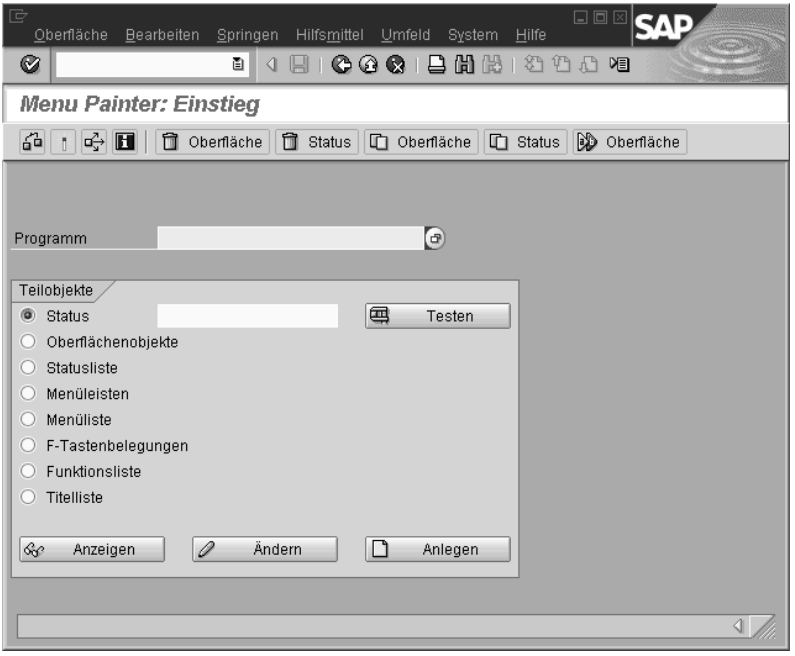

Abbildung 3.40 **© SAP AG**
Einstiegsbild des Menu Painter

In einem Eingabefeld tragen Sie den Namen des Programms ein, dessen Oberfläche bearbeitet werden soll. Das konkrete Element der Oberfläche selektieren Sie durch eine Reihe von Auswahlfeldern. Als Einziges dieser Elemente verfügt der Status über mehrere Ausprägungen, so dass Sie für den direkten Sprung zu einem Status auch noch dessen Namen eintragen müssen. Die konkrete Aktion (Anzeigen, Ändern, Anlegen) lösen Sie durch eine Drucktaste innerhalb des Dynpros aus. Beim Anlegen eines neuen Status erfragt das System in einem Popup den Status-Typ.

Falls Sie den Status aus dem Object Navigator heraus anlegen möchten, haben Sie zwei Möglichkeiten. Zunächst können Sie in der Objektliste einer Anwendung den Cursor auf den Programmnamen platzieren und die ANLEGEN-Funktion aus dem Kontextmenü aufrufen. In einem Popup (Abbildung 3.41) wählen Sie den Namen des Status und den konkreten Status-Typ. Zum selben Popup ge-

langen Sie auch, falls mindestens ein Status existiert und sie am Eintrag GUI-STATUS der Objektliste die Menüfunktion ANLEGEN auswählen.

Zu guter Letzt können Sie natürlich auch den Navigationsmechanismus der Entwicklungsumgebung benutzen, um einen Status per Doppelklick zu erzeugen.

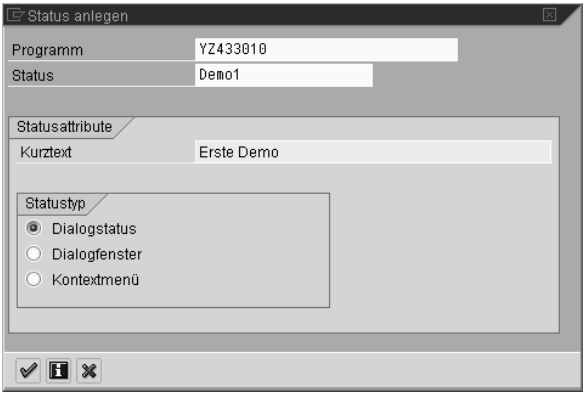

Abbildung 3.41 **© SAP AG**
Namen und Typ für einen Status vergeben

Nach Bestätigung dieses Popups gelangen Sie schließlich in die eigentliche, in Abbildung 3.42 dargestellte Pflegeoberfläche des Menu Painters. Zur besseren Übersicht wurde hier die Vollbild-Ansicht genutzt, der Navigationsbereich also ausgeblendet.

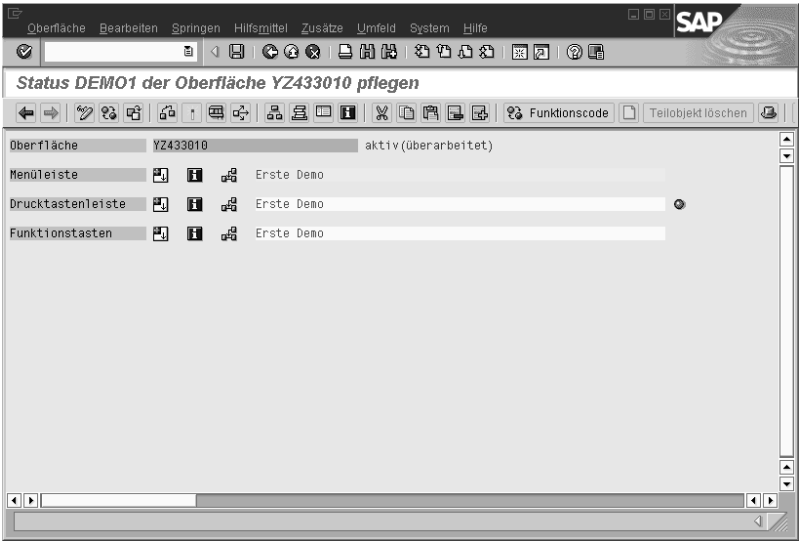

Abbildung 3.42 **© SAP AG**
Pflegebild für Status

Die verschiedenen Teilelemente können Sie separat ein- und ausblenden, um auf dem Bildschirm die Übersicht zu wahren. Zum Öffnen eines Teilbereichs klicken Sie mit der Maus auf das quadratische Symbol mit dem grünen Pluszeichen. Bild 3.43 zeigt Ihnen den geöffneten Arbeitsbereich für Menüs.

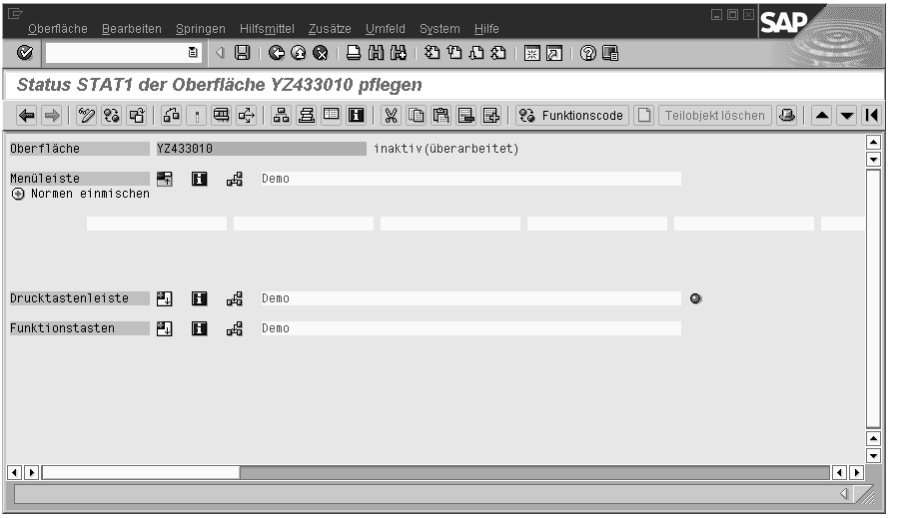

Abbildung 3.43 **© SAP AG**
Status mit leerem Menü-Bereich

In der endgültigen Anwendung fügt das R/3-System an das Menü automatisch die beiden Einträge SYSTEM und HILFE an. Der Inhalt dieser beiden Pulldown-Menüs kann nicht verändert werden.

Menüs

In einem neu angelegten Status existieren noch keine Menüfunktionen. Alle Eingabefelder sind leer. Sie können durch einen Klick auf das runde Symbol mit dem grünen Pluszeichen und der Beschriftung „Normen einblenden“ eine für Dialoganwendungen empfohlene Menübelegung einblenden. Diese Vorgabe entspricht dem SAP-Style Guide. Nach Möglichkeit sollten Sie sich bei Eigenentwicklungen an dieser Empfehlung orientieren. Bild 3.44 zeigt den Menu Painter mit dem eingemischten Menüeinträgen und einem geöffneten Pulldown-Menü. Die Pulldown-Menüs werden durch einen Doppelklick auf den Namen des Teilmenüs geöffnet und geschlossen.

In dieser Ansicht finden Sie drei verschiedene Arten von Eingabefeldern. In der ersten Zeile des Arbeitsbereiches befinden sich die Einträge für die Pulldown-Menüs. Jedes der dort befindlichen Felder nimmt den Namen eines solchen Menüs auf. Es handelt sich bei diesem Eintrag nur um einen Bezeichner. Mit diesen Einträgen können Sie noch keinen Funktionscode auslösen.

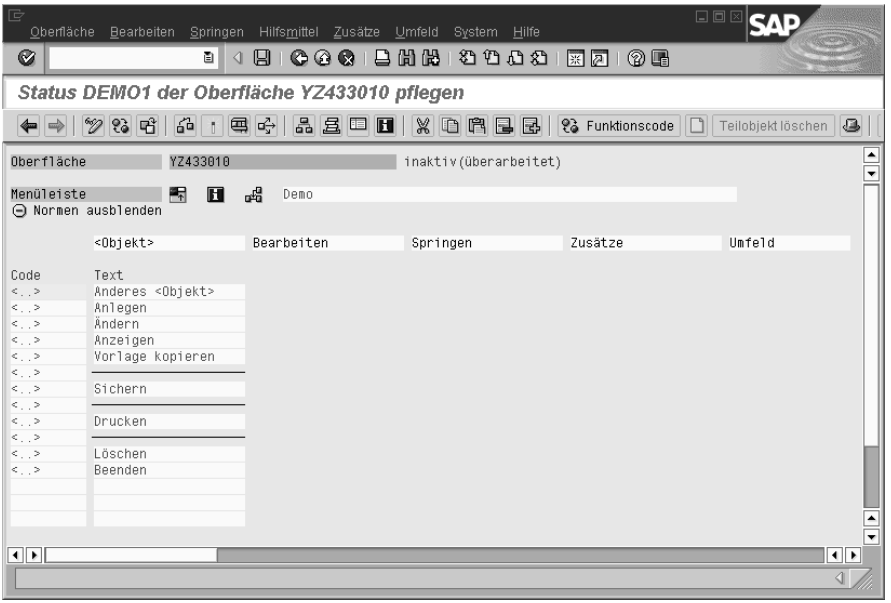

Abbildung 3.44 **© SAP AG**
Status mit eingeblendetem Menü

Die Bezeichner können beliebig verändert werden. Beim ersten dargestellten Eintrag <OBJEKT> müssen Sie dies sogar tun. Bei den in spitze Klammern eingeschlossenen Zeichenketten handelt es sich um Platzhalter, die durch den Programmierer durch konkrete Inhalte ersetzt werden müssen. Eine Dialoganwendung bezieht sich üblicherweise auf ein betriebswirtschaftliches Objekt, z. B. eine Rechnung oder eine Bestellung. Im ersten Pulldown-Menü befinden sich laut Style Guide grundlegende Funktionen zur Bearbeitung dieses Objekts, wie auch das eingeblendete Pulldown-Menü zeigt. Die Bezeichnung dieses Teilmenüs müssen Sie daher gemäß dem zu bearbeitenden Objekt wählen. Bei Bedarf können Sie weitere Pulldown-Menüs einfügen. Dies erfolgt mit der Menü-Funktion BEARBEITEN | EINFÜGEN | EINTRAG.

Unterhalb der Bezeichner können Sie das dazugehörige Menü einblenden. Ein solches Menü besteht im Menu Painter aus einer zweispaltigen Tabelle. In der ersten Spalte erfassen Sie den Bezeichner des Funktionscodes, in der zweiten den Text des Menüeintrages. Beim Einfügen der Vorlage werden einige Menüeinträge eingefügt, wobei aber noch kein Funktionscode zugeordnet wird. Die Zeichenkette <..> für den Funktionscode stellt einen Platzhalter dar. Dieser Funktionscode und somit alle dazugehörigen Menüeinträge erscheinen nicht im Menü der fertigen Anwendung. Sie können nun die bereits eingeblendeten Menüfunktionen durch Eintragen eines realen Funktionscodes aktivieren oder auch eigene Funktionscodes mit eigenen Texten hinzufügen. Nach der Eingabe

eines in der Funktionsliste noch nicht existierenden Funktionscodes wird dieser automatisch angelegt. Falls Sie einen Funktionscode in mehreren Status benutzen, können Sie ihm in jedem Status andere Texte zuweisen.

Da die Arbeitsoberfläche des Menu Painter natürlich auch ein Dynpro ist, werden Einträge erst dann ausgewertet, wenn der PAI-Teil des aktuellen Dynpros, also des Menu Painter, ausgeführt wird. Dies erfolgt nach Betätigen der Datenfreigabetaste oder Auswahl einer Menüfunktion bzw. Funktionstaste. In diesem Moment werden alle neu erfassten Funktionscodes mit eventuell vorhandenen Bezeichnungen aus der Funktionsliste ergänzt. Für neue Funktionscodes, zu denen noch keine Bezeichnung existiert, erfragt der Menu Painter in einem Popup die entsprechenden Texte.

Mittels einer Menüfunktion BEARBEITEN | EINFÜGEN | TRENNLINIE kann in einem Pulldown-Menü ein trennender Querstrich eingefügt werden. Falls ein Menüeintrag keinen Funktionscode auslösen, sondern ein untergeordnetes Pulldown-Menü öffnen soll, bleibt die Spalte mit dem Funktionscode frei. Lediglich in der zweiten Spalte ist die Bezeichnung des neuen Menüs einzutragen. Ein Doppelklick auf diesen Eintrag öffnet das neue Menü. Zur Laufzeit des Programms werden Einträge in Pulldown-Menüs, die auf weitere Menüs verweisen, mit einem kleinen, nach rechts gerichteten Dreieck gekennzeichnet. Das Löschen von Elementen ist über eine weitere Menüfunktion BEARBEITEN | EINTRAG | LÖSCHEN oder ein Symbol in der Drucktastenleiste bzw. die Funktionstaste Ctrl F9 möglich.

Weitere wichtige Menüfunktionen zum Bearbeiten eines Menüeintrages (siehe Tabelle 3.25) sind die Funktionen zum Einfügen neuer Elemente. Sie sind alle im Menü BEARBEITEN | EINFÜGEN zu finden.

Funktion	Beschreibung
Eintrag	Fügt einen neuen Eintrag vor der aktuellen Cursor-Position ein.
Trennlinie	Fügt vor der Cursorposition eine Trennlinie ein.
Include-Menü	Dynamisches Einfügen eines Menüs aus einem anderen Status.
Menü mit dynamischem Text	Fügt ein Pulldown-Menü ein, dessen Text dynamisch gesetzt werden kann.
Funktion mit dynamischem Text	Fügt eine Menüfunktion ein, deren Text dynamisch gesetzt werden kann.

Tabelle 3.25
Menüfunktion zum Einfügen neuer Elemente

Drucktastenleiste

Unterhalb des Arbeitsbereiches für das Menü finden Sie den Bereich zur Pflege der Drucktasten (Abbildung 3.45). Drucktasten erscheinen als Schaltfläche mit Text oder als Symbol in der Drucktastenzeile des Dynpros.

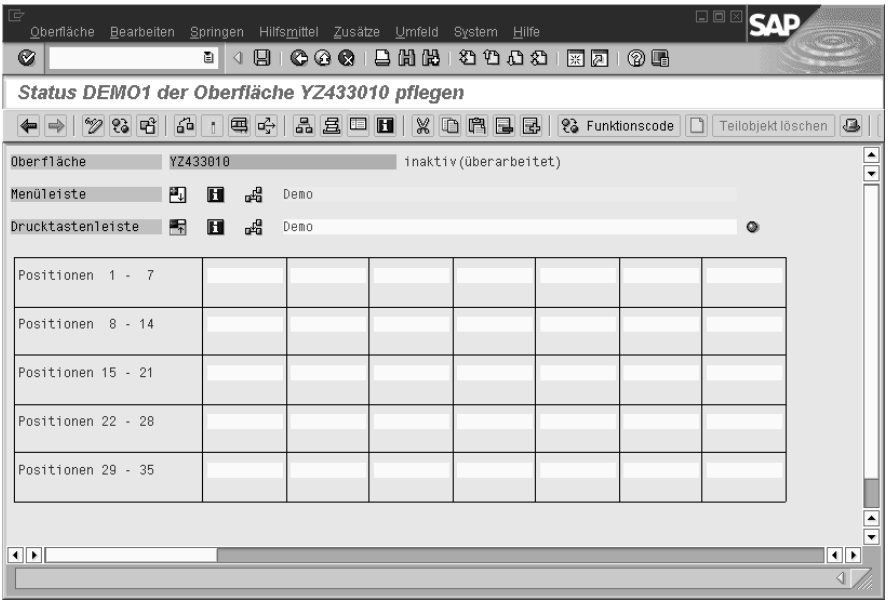

Abbildung 3.45 **© SAP AG**
Status mit geöffneter Drucktastenleiste

Um einen Eintrag in die Drucktastenzeile des Status zu übernehmen, tragen Sie den gewünschten Funktionscode in eines der Eingabefelder ein. In diesem Fall erscheint in der Drucktastenzeile eine Drucktaste mit dem Text, der dem Funktionscode zugeordnet wurde. Sie können stattdessen ein Icon vorgeben. Dies erfolgt am einfachsten durch Pflege der Funktionsliste. Diesem Vorgang ist ein eigener Abschnitt gewidmet. Beachten Sie bitte, dass Drucktasten nur zu Funktionscodes angelegt werden können, die mit einer Funktionstaste verbunden wurden.

Funktionstasten

Zur Beschleunigung der Auswahl wichtiger Funktionen stehen so genannte *Funktionstasten* zur Verfügung. Der Begriff Funktionstaste bezeichnet im SAP-Sprachgebrauch nicht unbedingt nur die speziellen Funktionstasten der Tastatur (meist F1 bis F12), sondern alle Tasten, die einen Funktionscode auslösen können. Dies sind Tasten (z.B. Funktionstasten F1 bis F12 der Tastatur) oder Tastenkombinationen (z.B. Control-Taste + alphanummerische Taste), mit denen Funktionscodes direkt ausgelöst werden können.

Die verfügbaren Funktionstasten werden in vier Gruppen eingeteilt. Zum einen existieren Funktionstasten mit systemweit vorgegebener Bedeutung. Diese Tasten (F1, F4 und F10) werden durch das System erkannt und ausgewertet. Sie erfordern daher keinen Funktionscode und sind immer aktiv.

Die zweite Gruppe bilden Tasten, die mit speziellen Symbolen der Symbolleiste korrespondieren. Diese Tasten sollen ebenfalls systemweit einheitliche Funktionen auslösen, wobei aber die Zuordnung eines Funktionscodes und die Implementierung der realen Funktionalität in der Verantwortung des Programmierers liegen. Die Zuordnung von Symbol zu Taste ist fest vorgegeben. Der Menu Painter bietet daher die jeweiligen Funktionstasten auch nicht zur separaten Bearbeitung an. Er stellt stattdessen für die verbindlich festgelegten Funktionen zusätzlich die Symbolleiste zur Bearbeitung bereit, in der diesen Funktionen und damit sofort einer der konkret festgelegten Tasten ein Funktionscode zugeordnet wird. Die Aktivierung dieser Funktionstasten erfolgt durch Zuweisen eines Funktionscodes zum Symbol (vergleiche Abbildung 3.46). Dazu ist die Eingabehilfe zu nutzen. Auf diese Weise werden die korrekten Funktionscodes aus dem zur Verfügung stehenden Vorrat gesucht.

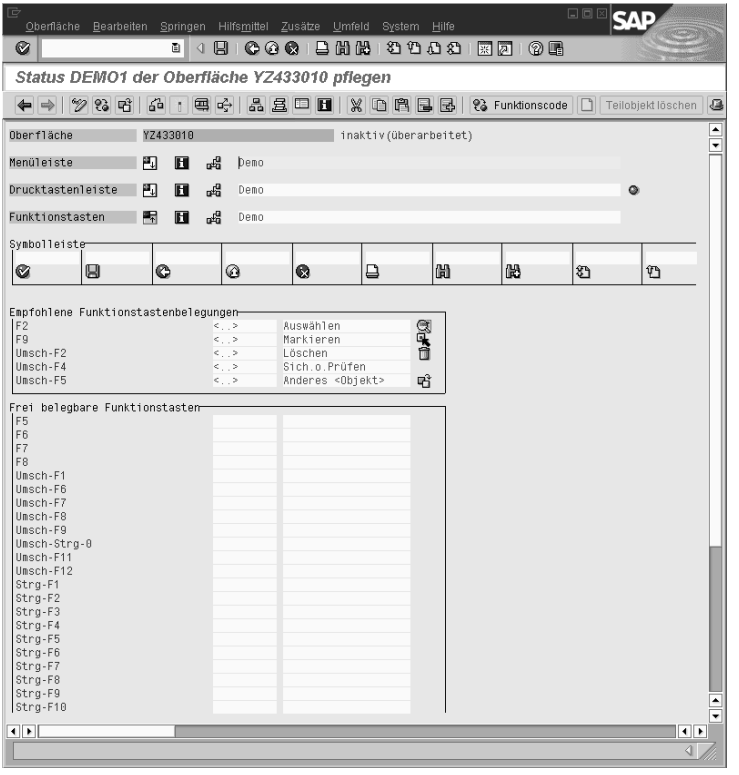

Abbildung 3.46 **© SAP AG**
Status mit Feldern zur Funktionstastenpflege

Tabelle 3.26 zeigt die Tasten der beiden bisher genannten Gruppen und die übliche Zuordnung.

Funktionstaste	Symbol	Bedeutung	Funktionscode notwendig
F1	Fragezeichen	Kontextsensitive Hilfe	Nein
F3	Grüner Pfeil	Zurück	Ja
F4	Lupe	Eingabehilfe	Nein
F10, ALT	Kein Symbol	Menüleiste aktivieren	Nein
F11	Gelbes Ordnersymbol	Speichern	Ja
F12	Rotes Kreuz	Abbrechen	Ja
F15	Gelber Pfeil	Beenden	Ja
F21, Ctrl-PgUp	Doppelter Pfeil nach oben	Zurück zur ersten Seite	Zum Teil
F22, PgUp	Einfacher Pfeil nach oben	Eine Seite zurück	Zum Teil
F23, PgDn	Einfacher Pfeil nach unten	Eine Seite vor	Zum Teil
F24, Ctrl-PgDn	Doppelter Pfeil nach unten	Vorwärts zur letzten Seite	Zum Teil

Tabelle 3.26
Funktionstasten mit vordefinierter Bedeutung

Die dritte Gruppe wird von Funktionstasten gebildet, für die SAP eine bestimmte Verwendung empfiehlt, z.B. Löschen oder Selektieren. In dieser Gruppe wird der Funktionscode direkt zugeordnet. Für diese Funktionstasten steht kein Platz in der Symbolleiste zur Verfügung. Für einige der Tasten existiert allerdings ein vorgegebenes Symbol, das in der Drucktastenleiste erscheint, falls der Funktionscode einer Drucktaste zugeordnet wird.

Die letzte Gruppe bilden die frei verwendbaren Funktionstasten. Sie werden ebenso wie die Funktionstasten der dritten Gruppe bearbeitet. In diesem Bereich werden nochmals 36 Funktionstasten bereitgestellt. In der fertigen Anwendung blendet ein Druck auf die rechte Maustaste ein Popup-Menü ein, in dem die verfügbaren Funktionstasten erscheinen und per Mausklick ausgewählt werden können.

Alle Einträge für Funktionstasten, die mit einem Funktionscode versehen werden können oder müssen, werden ebenso wie die Menüeinträge bearbeitet. Bei der Zuweisung sollten nur Funktionscodes benutzt werden, die auch im Menü

verfügbar sind. Eine Anwendung sollte stets komplett über das Menü bedient werden können.

Durch die Standardvorgaben können einige der Funktionstasten automatisch mit Funktionscodes belegt werden. Vor allem in Staus für Listen werden viele Funktionscodes verwendet, die das System selbst auswertet. Änderungen dieser Vorgaben sind zwar möglich, würden aber dem Anwender die recht komfortable Bearbeitung von Listen unnötig erschweren.

Vorlagen einbinden

Die verschiedenen Anwendungstypen (Dialoganwendung, Report) unterscheiden sich in ihrer Bedienung. Außerdem existieren Reports mit spezieller Funktionalität, z.B. interaktive Reports mit einer Baumdarstellung. In jeder dieser Anwendungen existieren bestimmte Funktionen, die direkt durch das Laufzeitsystem und nicht durch die von Ihnen geschriebene Anwendung ausgewertet werden sollen (z.B. Drucken einer Liste, Öffnen von Knoten in einer Baumdarstellung oder Blättern in der Ausgabeliste). Für diese Funktionen müssen Sie vorgegebene Funktionscodes auslösen. Zur Arbeitserleichterung existiert daher die Möglichkeit, den Status mit vordefinierten Funktionscodes zu füllen. Dazu bietet das System einige vorgefertigte Varianten für Oberflächen an. Diese werden als *Vorlagen* bezeichnet. Um eine Vorlage einzubinden, benutzen Sie die Menüfunktion ZUSÄTZE | VORLAGE ABGLEICHEN. Innerhalb eines Popups (Abbildung 3.47) können Sie nun eine der Standard-Vorlagen auswählen oder aber einen existierenden Status eines anderen Programms übernehmen.

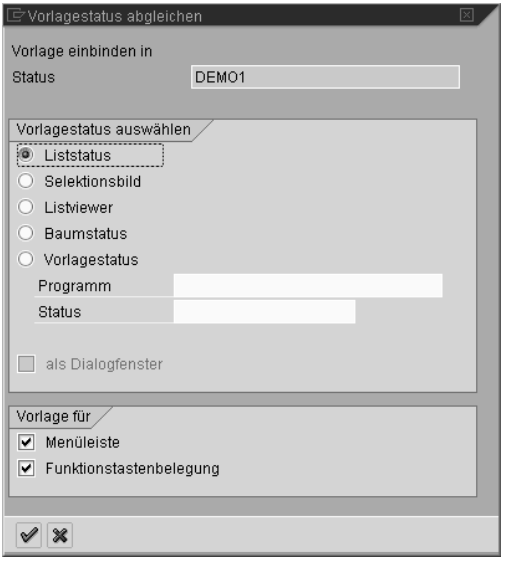

Abbildung 3.47 **© SAP AG**
Einbinden einer Vorlage

Funktionsliste

Das bereits beschriebene Pflegebild stellt den zentralen Teil des Menu Painter dar. Neben diesem Pflegebild besitzt die so genannte Funktionsliste eine entscheidende Bedeutung. Mit diesen beiden Pflegebildern lassen sich nahezu alle Tätigkeiten innerhalb der Oberflächenpflege ausführen. Zur Funktionsliste gelangen Sie mit der Menüfunktion SPRINGEN | OBJEKTLISTEN | FUNKTIONSLISTE. Abbildung 3.48 zeigt Ihnen die Funktionsliste.

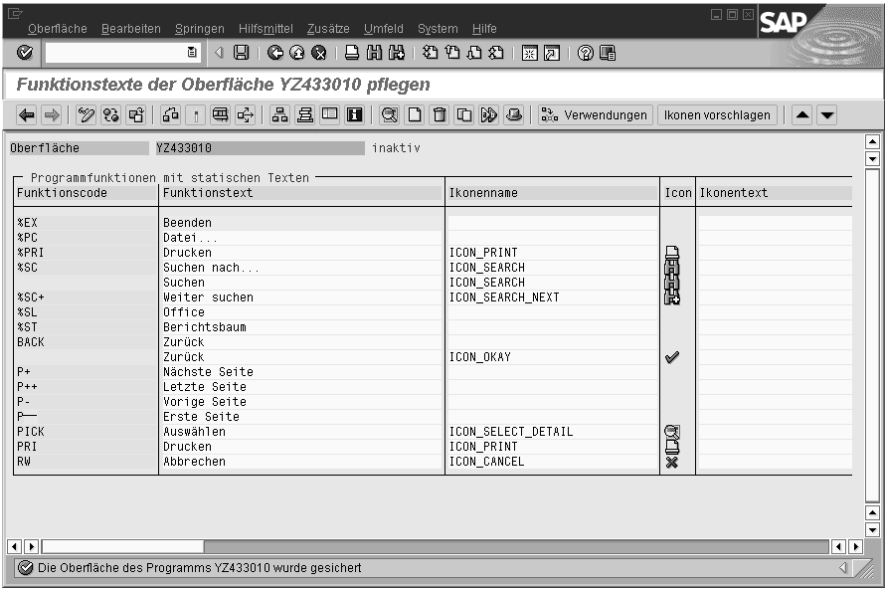

Abbildung 3.48 **© SAP AG**
Liste der Funktionscodes

Die Liste enthält alle Funktionscodes, die in den verschiedenen Status der Oberfläche benutzt werden und zeigt alle Eigenschaften des jeweiligen Funktionscodes an. Falls zu einem Funktionscode mehrere Texte existieren, finden Sie in dieser Tabelle auch mehrere Zeilen vor.

Wie bereits erwähnt, können Funktionscodes sowohl mit statischen als auch mit dynamischen Texten versehen werden. Bei letztgenannter Variante entfällt allerdings die Möglichkeit, dem Funktionscode ein Icon zuzuordnen. Die Funktionsliste besteht daher aus zwei Teilen. Im ersten erscheinen die Funktionscodes mit statischen Texten, im zweiten die mit dynamischer Textzuweisung.

Alle Eigenschaften eines Funktionscodes können in einem Popup gepflegt werden, dass Sie durch einen Doppelklick auf die Zeile des Funktionscodes auslösen können. Abbildung 3.49 zeigt dieses Popup.

Abbildung 3.49 **© SAP AG**
Pflege der Eigenschaften eines Funktionscodes

Drei Eingabefelder dieses Popups dienen zur Pflege der diversen Texte. Das Feld FUNKTIONSTEXT enthält den Text für Menüeinträge. Dieser Text erscheint auch in den Drucktasten, sofern dem Funktionscode kein Icon zugewiesen wurde. Im Feld IKONENTEXT pflegen Sie einen optionalen Text, der zusammen mit dem Icon in der Drucktaste erscheint. Das Feld Infotext nimmt einen ebenfalls optionalen Text auf, der in einem kleinen gelben Fenster erscheint, wenn der Mauszeiger eine kurze Zeit über einer Drucktaste verharrt. Im Feld Direktanwahl tragen Sie ein Zeichen ein, mit dem die Funktion in einem aufgeklapptem Pulldown-Menü über die Tastatur aufgerufen werden kann.

Die Rolle des Funktionstyps wurde bereits erläutert. Da der Wertevorrat für dieses Feld fest vorgegeben ist, steht eine Eingabehilfe bereit (Abbildung 3.50).

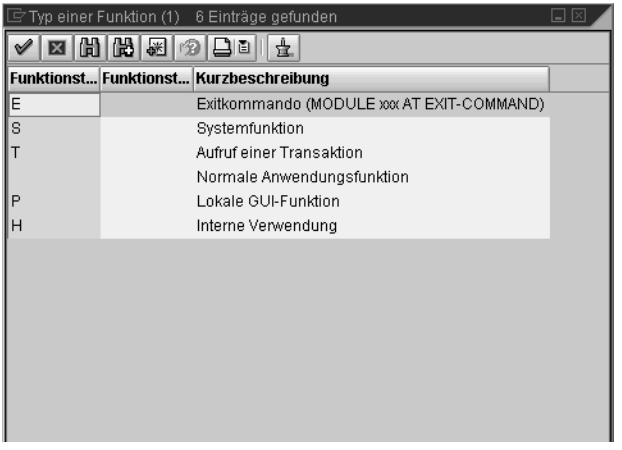

Abbildung 3.50 **© SAP AG**
Eingabehilfe für den Funktionstyp

Falls Sie dem Funktionscode ein Icon zuordnen möchten, sollten Sie die entsprechende Eingabehilfe (Abbildung 3.51) verwenden. Der Vorrat an Icons und zum Teil auch die grafische Gestaltung sind stark von der R/3-Version abhängig.

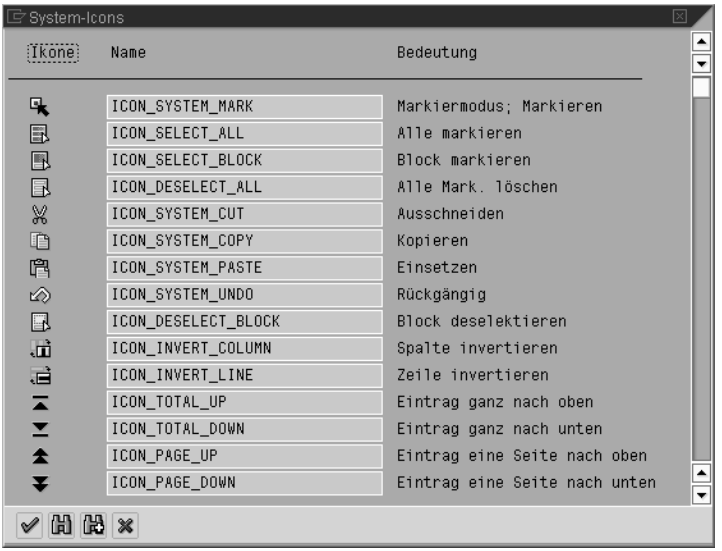

Abbildung 3.51 **© SAP AG**
Eingabehilfe für Icons

Die Drucktaste TEXTART ÄNDERN im Eigenschaften-Popup ermöglicht den Wechsel zwischen der statischen und der dynamischen Textzuweisung. Bei Betätigen dieser Taste ändert sich der Aufbau des Popups, da für Funktionscodes mit dynamischer Textzuweisung nur der Feldname für die Textübergabe und der Funktionstyp gepflegt werden können (Abbildung 3.52).

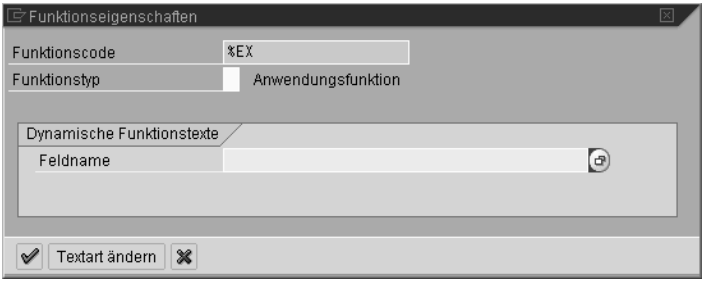

Abbildung 3.52 **© SAP AG**
Eigenschaften-Popup für Funktionscodes mit dynamischer Textzuweisung

Auch in diesem Popup existiert die Schaltfläche zum ändern der Textart. Sie bewirkt hier natürlich den Wechsel von dynamischer zu statischer Textzuweisung.

Sie können in diesem Popup sowohl zwischen einigen vordefinierten Status wählen als auch Bezug auf einen bereits existierenden Status eines beliebigen Programms nehmen.

Titelliste

Ein Titel existiert neben der eigentlichen Oberfläche. Im Repository Browser existiert daher eine eigenständige Rubrik mit der Bezeichnung *GUI-Titel*. Ein Titel ist ein einzeiliger Text, der zur Laufzeit des Programms in die Titelzeile des Bildschirmfensters gestellt werden kann. Eine Oberfläche kann über mehrere Titel verfügen, die durch einen zehnstelligen Bezeichner identifiziert werden. Innerhalb der Oberflächenpflege können Sie mit der Menüfunktion SPRINGEN | OBJEKTLISTEN | TITELLISTE problemlos in die Pflege der Titel verzweigen. Abbildung 3.53 vermittelt einen Eindruck von der Pflegeoberfläche.

Abbildung 3.53 **© SAP AG**
Titelliste

3.3.4 Die Oberfläche für Listen

Das R/3-System stellt für Reports automatisch eine Oberfläche mit einigen Status bereit. Je nach Zustand des Reports (Bearbeitung des Selektionsbildes, Ausgabe der Liste) wird automatisch ein passender Status gesetzt, sofern der Programmierer nicht explizit einen eigenen Status verwendet. Die in den verschiedenen Standardstatus erzeugten Funktionscodes werden automatisch durch das System ausgewertet. Die Tabellen 3.27 bis 3.29 zeigen alle durch das System verarbeiteten Funktionscodes geordnet nach logischen Gesichtspunkten. Nicht alle dieser Funktionscodes sind in allen Status verfügbar. Die Funktionscodes werden zunächst durch das System ausgewertet, wobei das System für einige Funktionscodes Zeitpunktanweisungen prozessiert, über die dann eine individuelle Auswertung möglich ist. Näheres zur Übergabe und Auswertung der Funktionscodes in Reports erfahren Sie im Abschnitt 3.4.3.

Funktionscode	Funktion	Vorgesehene Funktionstaste
%CH	Überschriftenpflege	
%EX	Beenden	F15
%GD	Grafik	
%PC	Abspeichern der Liste auf Frontend-System	
%SC	Suchen nach Zeichenkette	
%SL	Sprung zum Office-Menü	
%ST	Reportingbaum	
BACK	Zurück	F3
FSET	Wertemenge	
RSET	Varianten	
LEAV	Beenden	
ONLI	Programm starten (vom Selektionsbildschirm aus)	
RW	Eine List-Stufe zurück	F12

Tabelle 3.27
Funktionscodes der Listenverarbeitung: Sprung zu anderen Programmteilen

Funktionscode	Funktion	Vorgesehene Funktionstaste	Funktions-code
P	Erste Seite		
P+	Eine Seite vorwärts	F23	
P++	Sprung zum Ende der Liste	F24	
P-	Eine Seite rückwärts	F22	
P--	Sprung zum Anfang der Liste	F21	
PFnn	Funktionstaste PFnn gedrückt	Fn	AT PFnn
PICK	Zeile ausgewählt	F2	AT LINE-SELECTION
PL++	Sprung zur letzten Zeile des aktuellen Abschnitts		
PL+n	Springe n Zeilen vorwärts		
PL--	Sprung zur ersten Zeile des aktuellen Abschnitts		
PL-n	Springe n Zeilen rückwärts		
PP+(n)	Springe einen (n) Abschnitt(e) vorwärts		
PP++	Sprung zum Anfang der letzten Seite		
PP-(n)	Springe einen (n) Abschnitt(e) vorwärts		
PP--	Sprung zum Anfang der Liste		
PPn	Sprung zum Anfang von Abschnitt n		
PRI	Liste drucken	F13	
PS++	Sprung zur letzten Zeichenspalte der Liste		
PS+n	Springe n Zeichenspalten nach rechts		

Tabelle 3.28
Funktionscodes in der Listenverarbeitung: Bewegen in der Liste

Funktionscode	Funktion	Vorgesehene Funktionstaste	Funktions-code
PS--	Sprung zur ersten Zeichenspalte der Liste		
PS-n	Springe n Zeichenspalten nach links		
PSnn	Springe zu Zeichenspalte n		

Tabelle 3.28
Funktionscodes in der Listenverarbeitung: Bewegen in der Liste (Fortsetzung)

Funktions-code	Funktion	Vorgesehene Funktionstaste
ALLS	Alle Selektionen.	Ctrl-B
DBAC	Zurück	
DCAN	Abbrechen	
DELA	Selektion ganz löschen.	
DELS	Einen Eintrag der Selektion löschen.	F14, Ctrl-14
DSAV	Übernehmen	
DYNS	Freie Abgrenzungen.	F16
E	Zurück	F3
ECAN	Abbrechen	F12
ENDE	Beenden	F15, Ctrl-15
FC01	Anwenderfunktion 1	F19
FC02	Anwenderfunktion 2	F20
FEWS	Ausgewählte Selektionen.	Ctrl-D
FSET	Komplexe Abgrenzung.	
GET	Variante holen.	F17
GOON	Weiter	
JOBS	In Job einstellen.	

Tabelle 3.29
Funktionscodes in der Listenverarbeitung: Selektionsbildschirm

Funktions-code	Funktion	Vorgesehene Funktionstaste
NFIE	Neue Feldauswahl.	
ONLI	Ausführen	F8
OPTI	Selektionsoptionen	F2
PRIN	Ausführen + Drucken.	F13, Ctrl-13
SAVE	Als Variante sichern.	
SCRH	Hilfe Selektionsbild.	
SJOB	Im Hintergrund ausführen.	F9
SPOS	Als Variante sichern.	F11
VATT	Attribute	
VBAC	Zurück	

Tabelle 3.29
Funktionscodes in der Listenverarbeitung: Selektionsbildschirm (Fortsetzung)

Um in Reports eigene Funktionscodes verwenden zu können, müssen diese in einen Status aufgenommen werden. Bei Ausführung des Reports ist dann der Status mit dem bereits bekannten Kommando `SET PF-STATUS` zu setzen. Falls für einen Report ein neuer Status gesetzt werden soll, ist er dem Report zuzuordnen. Sie legen dazu mit dem Menu Painter einen neuen Status an und wählen eine passende Vorlage. Anschließend ergänzen Sie den Status mit Ihren eigenen Funktionscodes. Sie können ggf. auch existierende Funktionscodes löschen, um dem Endanwender die Ausführung bestimmter Funktionen zu verbieten. Sie müssen dabei aber gewährleisten, dass die Grundfunktionalität der List-Verarbeitung (Rollen, Beenden) sichergestellt ist.

3.3.5 Kommandos für Titel und Oberfläche

Oberfläche setzen

Ein Dynpro muss in fast allen Fällen mit einem Status der Oberfläche verbunden werden, damit der Anwender Programmaktivitäten auslösen kann. Dies erfolgt mit dem Kommando

```
SET PF-STATUS status.
```

Der Name des Status ist bis zu zwanzig Zeichen lang. Er muss in Großbuchstaben notiert werden und in einfachen Anführungsstrichen stehen. Nachdem ein Status gesetzt wurde, bleibt er so lange aktiv, bis ein neuer Status gesetzt oder mit

```
SET PF-STATUS SPACE.
```

ein Standardstatus aufgerufen wird. Dieser ist auch aktiv, wenn kein Status eingestellt wird. In Anwendungen kann ein Status beliebig oft, auch innerhalb ein und desselben Dynpros, gewechselt werden. Der Name des jeweils aktuellen Status steht im Systemfeld SY-PFKEY.

Mitunter sollen in einem Status einige Funktionscodes vorübergehend deaktiviert werden. Erlaubt ein Dynpro beispielsweise das satzweise Blättern in einem Datenbestand mit den Funktionen SATZ VOR und SATZ RÜCK, ist es sinnvoll, am Anfang bzw. Ende des Datenbestandes jeweils eine der beiden Funktionen auszuschalten. Auch beim Fehlen von Berechtigungen für spezielle Funktionen (z.B. Löschen) sollten diese deaktiviert werden, so dass sie von nicht autorisierten Anwendern gar nicht erst aufgerufen werden können. Dieses Problem ließe sich natürlich mit diversen unterschiedlichen Status lösen, wobei der Programmier- und Verwaltungsaufwand recht hoch werden kann. In einem Status können Funktionscodes deshalb auch zur Laufzeit deaktiviert werden, ohne den Status insgesamt ändern zu müssen. Die Menüeinträge der deaktivierten Funktionscodes erscheinen zwar weiterhin im Menü, werden dort aber in grauer Farbe dargestellt und können nicht angewählt werden.

Das Deaktivieren von Funktionscodes erfolgt mit dem Parameter EXCLUDING des SET PF-STATUS-Kommandos. Nach diesem Zusatz können sowohl ein einzelner Funktionscode als auch eine interne Tabelle stehen, die beliebig viele Funktionscodes enthalten kann. Diese Tabelle muss folgende Struktur haben:

```
DATA: BEGIN OF FCODE_TAB OCCURS 5,
  FCODE LIKE SY-UCOMM,
END OF FCODE_TAB.
```

Beispiele für den Aufruf des so erweiterten Kommandos sind:

```
SET PF-STATUS 'STAT1' EXCLUDING 'FNC1'.
```

oder

```
REFRESH FCODE_TAB.
FCODE_TAB-FCODE = 'FNC1'.
APPEND FCODE_TAB.
FCODE_TAB-FCODE = 'FNC2'.
APPEND FCODE_TAB.
SET PF-STATUS 'STAT1' EXCLUDING FCODE_TAB.
```

Das letzte Beispiel erfordert, dass die interne Tabelle FCODE_TAB in den globalen Daten des Modul-Pools angelegt wird. Ein Status wird üblicherweise zum Zeitpunkt PBO gesetzt. In seltenen Fällen kann es sinnvoll sein, im PAI-Teil in Abhängigkeit vom aktuellen Funktionscode einen neuen Status zu setzen oder den vorhandenen zu modifizieren. Aus Gründen der Übersichtlichkeit sollte dies nur in Ausnahmefällen erfolgen.

Titel setzen

Jedes Bildschirmfenster verfügt über eine Titelzeile. Auch dieser Titel wird zum Zeitpunkt PBO gesetzt. Dies erfolgt mit der Anweisung

```
SET TITLEBAR title.
```

Der Titel wird dabei durch einen dreistelligen Identifikator angegeben. Im Programm muss der Identifikator in einfache Anführungsstriche eingeschlossen werden. Er kann daher leicht mit einem Direktwert verwechselt werden. Falls bei Abarbeitung des Kommandos kein Titel existiert, verwendet das System eine Standardvorgabe.

Diese Titel werden im Menu Painter gepflegt. Es ist aber einfacher, beim Notieren des Quellcodes das Werkzeug zur Titelpflege mittels des Navigationsmechanismus aufzurufen. Zunächst wird der korrekte Bezeichner notiert, anschließend wird ein Doppelklick auf diesen Bezeichner ausgeführt.

Ein Titel kann bis zu neun Platzhalter für Parameter enthalten. Diese bestehen entweder nur aus dem Zeichen `&` oder aber diesem Zeichen und einer laufenden Nummer von 1 bis 9. Beim Setzen des Titels können die einzufügenden Werte nach dem Zusatz `WITH` übergeben werden.

```
SET TITLEBAR title WITH parameter_1 ... parameter_9.
```

Die nummerierten Platzhalter werden durch den Parameter an der entsprechenden Stelle im Kommando ersetzt, die nicht nummerierten Platzhalter hingegen werden der Reihe nach von links beginnend den Parametern des `SET`-Kommandos zugewiesen. Steht für einen Platzhalter kein Parameter zur Verfügung, wird ein Leerzeichen eingetragen. Falls im Titel ein `&`-Zeichen stehen soll, muss es in der Titelpflege doppelt angegeben werden (`&&`).

Der endgültige Titel wird vom System zusätzlich zur Anzeige im Bildschirmfenster auch noch in das Systemfeld `SY-TITLE` gestellt.

3.4 Standardreports

Die bisher vorgestellten Online-Programme ermöglichten die Erprobung einiger ABAP-Kommandos. Auch wenn dabei bereits eine Tabelle der Datenbank ausgelesen wurde – um vollwertige Reports im engeren Sinne des Wortes handelte es sich bei diesen Beispielen noch nicht. Der Begriff Report bezeichnet im R/3-System einen speziellen Programmtyp mit einer recht eng umrissenen Aufgabe. Reports sollen Daten aus einer oder mehreren Tabellen auslesen, aufbereiten und darstellen. Die Programmiersprache ABAP wurde nicht als Universalsprache, sondern als Werkzeug zur Entwicklung des R/3-Systems entworfen. Daher verfügt sie über einige Kommandos, deren Funktionalität auf die Anforderungen der Listenprogrammierung zugeschnitten wurde. Die effektive Verwendung die-

ser Anweisungen und die Unterstützung der Listengenerierung durch die interne Logik des R/3-Systems erzwingen natürlich auch eine auf deren Eigenschaften zugeschnittene Programmierweise. Nach der Einführung in einige wichtige Kommandos mit globaler Bedeutung in den vorangegangenen Abschnitten soll der nun folgende konkret die Anforderungen der Report-Programmierung und die dazu erforderlichen Anweisungen erläutern. Getreu dem Anliegen des Buches, vor allem die Konzepte und Zusammenhänge der Programmiersprache ABAP zu erläutern, finden Sie in diesem Abschnitt zunächst die Kommandos, die Einblicke in die Struktur und die Arbeitsweise eines Reports ermöglichen. Weitere Kommandos, die zwar hilfreich, aber für die Funktion eines Reports nicht zwingend notwendig sind, finden Sie am Ende dieses Abschnitts.

Die zunächst nicht dialogfähigen Reports wurden im Laufe der kontinuierlichen Weiterentwicklung des R/3-Systems und der Programmiersprache ABAP durch einige sehr spezielle Anweisungen aufgewertet. Diese ermöglichen es dem Programmierer, innerhalb eines Reports einige Aktionen des Anwenders (z.B. die Betätigung von Funktionstasten) zu erkennen und mit individuellen Programmfunktionen zu versehen. Damit verfügen Reports über einige interaktive Fähigkeiten. Reports werden daher oft in nicht interaktive (Standard-) bzw. in interaktive Reports eingeteilt. Obwohl Letztere lediglich eine Sonderform der ersten Variante sind, sollen sie wegen der neuen Anweisungen und der damit verbundenen Änderung der Programmierweise in einem separaten Abschnitt erläutert werden.

3.4.1 Datenaufbereitung

Umfangreiche Listen werden üblicherweise durch Zwischenüberschriften und -summen gegliedert. Oft gehen in eine Liste Daten aus mehreren Tabellen ein, die über Schlüsselfelder miteinander verbunden wurden, unter Umständen sind also je nach Datensatz oder Datensatzgruppe unterschiedliche Aktionen mit den Daten erforderlich. Unter ABAP existieren einige Funktionen, mit denen typische Aktivitäten zur Datenaufbereitung schnell und sehr einfach programmiert werden können.

Gruppenbildung

Innerhalb der Bearbeitung einer internen Tabelle oder des noch zu beschreibenden Extract-Datenbestandes mittels einer `LOOP`-Schleife können Programmaktivitäten vom Wechsel des Inhaltes eines Tabellenfeldes abhängig gemacht werden. Dabei ist allerdings die Verwendung des Zusatzes `WHERE` in der `LOOP`-Anweisung nicht möglich. Die in diesem Unterabschnitt beschriebenen Kommandos sind nicht prinzipiell an Reports gebunden. Sie könnten auch in Dialoganwendungen eingesetzt werden. Dort besteht aber sehr selten die Notwendigkeit zum Einsatz solcher Kommandos. Aus diesem Grund finden Sie die Beschreibung an dieser Stelle.

Unterschiedliche Datensätze, in denen ein oder mehrere Felder einen gleich bleibenden Wert haben, werden zu Gruppen zusammengefasst, für die gemeinsame Anweisungen ausgeführt werden. Die gruppenbezogenen Programmteile können sowohl vor als auch nach einer Gruppe ausgeführt werden. Die Anweisungen zur Ausführung einer gruppenbezogenen Bearbeitung lauten:

```
AT NEW field.
...
ENDAT.
```

bzw.

```
AT END OF field.
...
ENDAT.
```

Die Anweisungen innerhalb des AT-ENDAT-Blocks werden ausgeführt, wenn das angegebene Feld oder ein in der Struktur der Feldleiste weiter links stehendes Feld seinen Wert ändert. Dabei kann über die Optionen NEW bzw. END OF festgelegt werden, ob der Kommandoblock beim Lesen des ersten oder letzten Satzes einer zusammengehörigen Gruppe ausgeführt werden soll. Der Name der zu bearbeitenden Tabelle ist der LOOP-Anweisung bekannt. Daher müssen in den AT-Anweisungen innerhalb der LOOP-Schleife nur die eigentlichen Feldnamen ohne Tabellennamen notiert werden. Die AT-Kommandos können daher nur auf die in der LOOP-Anweisung benannte Tabelle angewendet werden. Unabhängig von Feldnamen arbeiten die Kommandos

```
AT FIRST.
...
ENDAT.
```

und

```
AT LAST.
...
ENDAT.
```

Sie erlauben es, Kommandos unmittelbar bei Beginn des ersten oder nach Beendigung des letzten Schleifendurchlaufes der LOOP-Schleife auszuführen.

Trotz syntaktischer Ähnlichkeiten sind diese gruppenbildenden Anweisungen keine der später noch zu beschreibenden Zeitpunktanweisungen. Sie sind für die interne Steuerlogik keine Sprungziele, die zu bestimmten Zeitpunkten angesprungen werden, sondern lediglich ein Symbol für komplexere Operationen. Die Wirkung dieser Kommandos könnte durch Zwischenspeichern eines Datensatzes und Vergleich mit dem aktuellen Datensatz in einer IF-Anweisung nachgebildet werden. Die Kommandos zur Gruppenbildung werden daher auch nicht durch die Steuerlogik aufgerufen, sondern sie ordnen sich in den sequenziellen Ablauf der Kommandos innerhalb der LOOP-Schleife ein. Dies ist bei der

Platzierung der Anweisungen im Zusammenhang mit Ausgabeanweisungen zu beachten. Ebenfalls zu beachten ist, dass der Datenbestand gegebenenfalls sortiert werden muss, sofern Sie keine der automatisch sortierten Tabellen benutzen. Die Anweisungen zur Gruppenbildung arbeiten natürlich nur dann korrekt, wenn alle Datensätze, die in einer Gruppe in der Liste erscheinen sollen, auch hintereinander gelesen werden.

Die Anweisungen zur Gruppenbildung nebst ihren Eigenarten sollen für die folgenden beiden Programme näher beschrieben werden. In der ersten Version werden alle Datensätze für eine Umsatzstatistik ausgegeben, wobei für jeden Mitarbeiter eine Untergruppe gebildet wird. Die zur Arbeit notwendige interne Tabelle wird hier direkt mit Daten gefüllt, im praktischen Einsatz würde natürlich eine Datenbanktabelle ausgelesen werden.

```
REPORT  yz434010.

* interne Tabelle für Demo
DATA: BEGIN OF sales OCCURS 10,
  name(20),
  date TYPE d,
  turnover TYPE i,
END OF sales.

* Füllen der Tabelle mit 6 Datensätzen
sales-name = 'MEIER'.
sales-date = '19950703'.
sales-turnover = 300.
APPEND sales.

sales-name = 'MEIER'.
sales-date = '19950703'.
sales-turnover = 200.
APPEND sales.

sales-name = 'MEIER'.
sales-date = '19950704'.
sales-turnover = 100.
APPEND sales.

sales-name = 'LEHMANN'.
sales-date = '19950704'.
sales-turnover = 110.
APPEND sales.

sales-name = 'LEHMANN'.
sales-date = '19950704'.
sales-turnover = 220.
```

```
APPEND sales.
sales-name = 'LEHMANN'.
sales-date = '19950705'.
sales-turnover = 330.
APPEND sales.

* Schleife über interne Tabelle
LOOP AT sales.

*   Ausgabe einer Überschrift im ersten Schleifendurchlauf
  AT FIRST.
    WRITE: / 'Datum', 19 'Umsatz', / .
  ENDAT.

*  Zwischenüberschrift bei neuem Namen
  AT NEW name.
    WRITE: / sales-name.
    WRITE: / '============================='.
  ENDAT.

*  Ausgabe eines Datensatzes
  WRITE: /(10) sales-date, sales-turnover.

*  Zwischensumme ausgeben
  AT END OF name.
    SUM.
    WRITE: / '============================='.
    WRITE: / 'Summe:', 12 sales-turnover, /.
  ENDAT.

*  Im letzten Schleifendurchlauf Endsumme
  AT LAST.
    SUM.
    WRITE: /, /,  'Gesamt:' , 12 sales-turnover.
    WRITE: /, /10 '*** ENDE ***'.
  ENDAT.
ENDLOOP.
```

Das Demoprogramm arbeitet die zuvor erzeugte und gefüllte interne Tabelle in einer Schleife ab. Im ersten Schleifendurchlauf, der mittels `AT FIRST` erkannt wird, erzeugt die Anwendung einen bescheidenen Tabellenkopf. Dies soll nur zur Demonstration des `AT FIRST`-Kommandos dienen, in vollwertigen Reports werden Tabellen- und Seitenköpfe auf andere Weise erzeugt. Nach dem `AT FIRST`-Kommando wertet `AT NEW NAME` einen eventuellen Gruppenwechsel aus und erzeugt eine aus dem Namen des Verkäufers bestehende Zwischenüberschrift. Daran anschließend kann der interessierende Inhalt des Datensatzes ausgegeben werden.

Es ist unbedingt notwendig, auf die richtige Reihenfolge von AT- und WRITE-Kommandos zu achten. Die Zwischenüberschrift muss vor Ausgabe des eigentlichen Datensatzes erzeugt werden. Würden die AT FIRST- und WRITE-Anweisungen im obigen Beispiel vertauscht werden, würde zunächst der Datensatz und danach die Zwischenüberschrift in der Liste erscheinen.

Im AT END OF-Kommando erscheint eine neue Anweisung SUM. Mit dieser Anweisung werden alle nummerischen Felder aller Datensätze der jeweiligen Gruppe aufsummiert und in der Kopfzeile der Tabelle bereitgestellt. Nach der Anweisung SUM ist im Feld verk-umsatz also die Summe aller Umsätze eines Verkäufers enthalten. Auch für AT END OF gilt die Abhängigkeit von der Reihenfolge der Anweisungen im Programm. Diese Anweisung darf erst nach der Ausgabe des Datensatzes stehen. Abbildung 3.54 zeigt die vom Programm erstellte Liste.

Abbildung 3.54 **© SAP AG**
Ausgabe des Demoprogramms YZ434010

Nicht nur das in der AT-Anweisung benannte Feld, sondern auch alle links davon stehenden bewirken einen Gruppenwechsel, wenn sich der Feldinhalt ändert. Sie können dies zunächst testen, indem Sie im ersten Demoprogramm zur

Gruppenbildung nicht nach dem Feld NAME, sondern nach datum gruppieren. Obwohl sich der dritte und vierte Datensatz nicht durch das Datum unterscheiden, wird wegen des in der Datenstruktur weiter links stehenden Feldes name ein Gruppenwechsel ausgelöst. Sowohl Struktur als auch Sortierung der internen Tabelle müssen also exakt auf die spätere Auswertung zugeschnitten sein. Das gilt insbesondere, wenn verschiedene AT-Anweisungen geschachtelt werden sollen. Auch dazu ein kurzes Beispiel: Es soll nach dem Namen gruppiert werden, außerdem sollen innerhalb einer Gruppe Datensätze mit gleichem Datum zusammengefasst werden. Das folgende Listing zeigt nur die neue LOOP-Schleife, der Rest des Programms muss nicht geändert werden. Sie können YZ434010 also kopieren und müssen nur innerhalb der LOOP-Anweisung einige Modifikationen vornehmen:

```
REPORT yz434020.
...
LOOP AT sales.
  AT NEW date.
    AT NEW name.
      WRITE: / sales-name.
      WRITE: / '============================'.
    ENDAT.
  ENDAT.

  AT END OF date.
    SUM.
    WRITE: /(10) sales-date, sales-turnover.
    AT END OF name.
      SUM.
      WRITE: / '============================'.
      WRITE: / 'Summe:', 12 sales-turnover, /.
    ENDAT.
  ENDAT.

ENDLOOP.
```

Die äußeren AT-Anweisungen müssen sich auf das niederwertigste Feld – hier datum – beziehen, damit jeder Gruppenwechsel erkannt wird. Innerhalb der Datums-Gruppenwechsel kann dann ermittelt werden, ob zusätzlich auch ein Gruppenwechsel beim Feld name auftrat. Würde in der äußeren AT-Anweisung der Name als Kriterium für einen Gruppenwechsel herangezogen, würde nur bei einem Wechsel des Namens auch auf ein wechselndes Datum geprüft. Die Zusammenfassung von Datensätzen mit identischem Datum würde so nicht stattfinden.

Die Ausgabe einzelner Datensätze muss entfallen, da für jedes Datum nur ein Summensatz angezeigt werden soll. Die WRITE-Anweisung hinter AT END OF DATUM bezieht sich nicht auf einen konkreten Datensatz der Tabelle, vielmehr

wird der durch SUM behandelte Kopfsatz verwendet. Da durch SUM aber nur numerische Felder bearbeitet werden, ist der Inhalt des Datumsfeldes korrekt. Abbildung 3.55 zeigt die Ausgabe des modifizierten Programms.

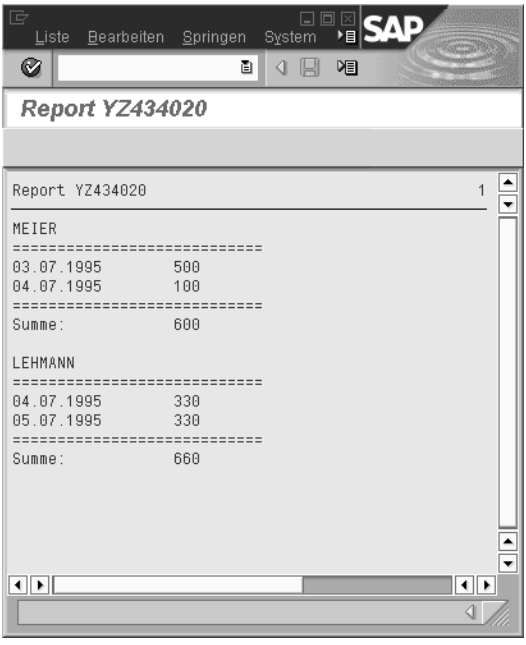

Abbildung 3.55 **© SAP AG**
Liste mit zusammengefassten Sätzen

Interessant an diesem Beispiel ist neben den geschachtelten AT-Anweisungen die Verwendung des SUM-Kommandos. Ohne weitere Angaben summiert es hinter AT END OF DATUM alle Umsätze zu einem Datum auf, während nach AT END OF NAME alle Umsätze des jeweiligen Verkäufers berücksichtigt werden. Das SUM-Kommando steht in engem Zusammenhang mit den AT-Kommandos. Aus diesem Beispiel ist indirekt auch ersichtlich, dass z. B. SUM ein sehr komplexes Kommando ist, das nur aus dem Quelltext und den AT-Anweisungen heraus erkennt, welche Datensätze aufzusummieren sind.

EXTRACT-Datenbestände

In die Ausgabe eines Reports gehen oft die Daten aus mehrere Tabellen ein, wobei die einzelnen Ausgabezeilen sehr häufig in einer anderen Reihenfolge erscheinen als durch die Primärschlüssel der Tabellen vorgegeben. Dazu ein praxisrelevantes, aber etwas vereinfachtes Beispiel: Alle Rechnungen eines bestimmten Zeitraums (Rechnungskopf inklusive aller Einzelposten) sollen, sortiert nach Lieferant und Rechnungsnummer, in einer Liste ausgegeben wer-

den. Die Liste soll für Lieferant und Rechnung jeweils eine Zwischenüberschrift und Zwischensummen enthalten. Die Ermittlung der Daten ist noch relativ einfach. Zunächst werden in der Tabelle mit den Belegköpfen die Rechnungen gesucht, die in den gewünschten Zeitraum fallen. Über die Rechnungsnummer werden die einzelnen Belegzeilen gesucht, über die Kreditorennummer die Angaben zum Lieferanten. In der Liste soll der Lieferant das oberste Ordnungskriterium sein, durch die Datenstruktur wird aber die Rechnungsnummer als Hauptmerkmal vorgegeben. Die gewünschte Liste kann also nicht sofort ausgegeben, sondern muss zunächst unter Zuhilfenahme einer internen Tabelle aufbereitet werden. Folgender als Pseudocode formulierter Ablauf wäre denkbar:

```
DATA: ITAB OCCURS 1000,
  LIEFNR,
  LIEFNAME,
  RECHNR,
  RECHDAT,
  ARTNR,
  ARTBEZ,
  MENGE,
  EINZELPREIS,
  SUMME,
END OF ITAB.

SELCECT * FROM RECHKOPF WHERE DATUM IN Zeitraum.
  SELECT SINGLE * FROM LIEFERANT WHERE LIEFNUM = RECHKOPF-
LIEFNUM.
  SELECT * FROM RECHZEILE WHERE RECHNUM = RECHKOPF-RECHNUM.
    MOVE ... . " Füllen der internen Tabelle.
  ENDSELECT.
ENDSELECT.

SORT ITAB.

LOOP AT ITAB.

  AT NEW LIEFNR.
    " Daten zum Lieferanten ausgeben
    AT NEW RECHNR.
      " Rechnungskopf ausgeben
    ENDAT.
  ENDAT.

   " Belegzeile ausgeben
  AT END OF ... .
    ...
  ENDAT.
ENDLOOP.
```

Neben der Verwendung der internen Tabellen kennt ABAP für Datenkombinationen dieser Art noch eine weitere Variante, so genannte EXTRACT-Datenbestände. Die Anwendung von EXTRACT-Datenbeständen ist gewöhnungsbedürftig und erfordert eine etwas ausführlichere Erklärung.

Jeder Report kann genau einen solchen Datenbestand verwalten. Dieser erhält daher auch keinen Namen, sondern wird implizit bei Verwendung spezieller Kommandos benutzt. Folgende Arbeitsschritte sind notwendig:

❶ Zunächst werden ein oder mehrere *Feldgruppen* benannt, die in den Datenbestand eingehen sollen. Dabei handelt es sich nur um die Deklaration der Namen dieser Feldgruppen. Die Namen sind, im Rahmen der für Bezeichner vorgegebenen Einschränkungen, wahlfrei. Eine Feldgruppe HEADER wird vom System zwar nicht automatisch erzeugt, besitzt aber eine vordefinierte Bedeutung, falls sie innerhalb eines Reports angelegt wird. Wegen der vordefinierten Bedeutung wird sie üblicherweise immer aufgenommen.

❷ In einem zweiten Schritt werden den Feldgruppen beliebige Daten- oder Tabellenfelder zugeordnet. Eine Feldgruppe wird damit zum Symbol für ein oder mehrere Felder. Sie ist nicht mit einer Feldleiste vergleichbar, der Zugriff auf Elemente der Feldgruppe ist nicht möglich (bzw. führt nicht zum gewünschten Erfolg).

❸ Es folgt die Datenauswertung. Immer, wenn alle Felder, die zu einer Feldgruppe gehören, mit korrekten Werten belegt sind, können die Feldgruppe und damit alle ihr zugewiesenen Felder in den EXTRACT-Datenbereich geschrieben werden. Dabei wird der Inhalt der Felder, die in der Feldgruppe HEADER enthalten sind, automatisch mit in den Datenbestand geschrieben. In diese Feldgruppe werden daher üblicherweise alle Schlüsselfelder aufgenommen.

❹ Nach Aufbau des Datenbestands kann dieser sortiert werden. Dabei werden alle Felder der Feldgruppe HEADER berücksichtigt. Nach dem Sortieren können keine zusätzlichen Werte in den Datenbestand aufgenommen werden.

❺ Nach einer eventuellen Sortierung erfolgt die endgültige Auswertung. Diese verläuft ähnlich wie die einer internen Tabelle, insbesondere sind die bereits erwähnten Anweisungen zur Gruppenbildung verfügbar. Nach einer Sortierung und/oder Auswertung können ebenfalls keine Werte mehr in den EXTRACT-Datenbestand aufgenommen werden.

Für jeden der Arbeitschritte existieren neue Kommandos bzw. Varianten bereits beschriebener Kommandos. Sie sollen nachfolgend, geordnet nach der Reihenfolge der Arbeitsschritte, erläutert werden.

Deklaration von Feldgruppen

Mit der Deklaration der Feldgruppen werden zunächst deren Namen bekannt gemacht. Die zugehörige Anweisung ist dementsprechend einfach:

```
FIELD-GROUPS: fieldgroup_1, fieldgroup_2, ... fieldgroup_n.
```

In der Syntaxbeschreibung steht sofort die Doppelpunktvariante der Anweisung, da die Deklaration einer einzelnen Feldgruppe zwar möglich, aber wohl selten sinnvoll ist. Natürlich kann die Anweisung ohne den Doppelpunkt auch zur Deklaration einer einzelnen Feldgruppe benutzt werden.

Felder zu Feldgruppen zuordnen

Für die Zuordnung von Feldern zu einer Feldgruppe wird eine spezielle Variante des Kommandos `INSERT` verwendet.

```
INSERT fieldlist INTO fieldgroup.
```

In der Feldliste (nicht Feldleiste!) werden alle in die Feldgruppe aufzunehmenden Felder aufgeführt. Die einzelnen Feldnamen werden jeweils durch ein Leerzeichen getrennt. Für `INSERT` ist hier ebenfalls eine Doppelpunktvariante möglich, mit der in einer `INSERT`-Anweisung Zuweisungen zu mehreren Feldgruppen erfolgen können:

```
INSERT: fieldlist_1 INTO fieldgroup_1,
        fieldlist_2 INTO fieldgroup_2,
              ...
        fieldlist_n INTO fieldgroup_n.
```

Obwohl hier das Kommando `INSERT` benutzt wird, handelt es sich noch nicht um eine Datenmanipulation. Es werden lediglich Verweise auf die Felder bzw. Feldnamen in die Feldgruppe eingefügt, keine Daten!

Füllen des Datenbestands

Mittels der Anweisung

```
EXTRACT fieldgroup.
```

wird der aktuelle Inhalt der Felder, die über `INSERT` der Feldgruppe zugeordnet wurden, in den `EXTRACT`-Datenbestand übertragen. Falls eine Feldgruppe `HEADER` existiert, werden deren Felder automatisch mit übertragen. Ein `EXTRACT`-Kommando erzeugt einen Datensatz im Datenbestand. Dabei bleiben die Felder der Feldgruppen, die nicht im `EXTRACT`-Kommando aufgeführt wurden, leer.

Auch `EXTRACT` kann durch Verwendung des Doppelpunkts, wie viele andere Kommandos, mehrere Parameter verarbeiten:

```
EXTRACT: fieldgroup_1, fieldgroup_2, ... fieldgroup_n.
```

Sortieren des Datenbestands

Das Sortieren erfolgt in der einfachsten Variante nur durch Angabe von

```
SORT.
```

Ohne Angabe eines Namens bezieht sich dieses Kommando automatisch auf den EXTRACT-Datenbestand. Mit einigen Optionen können die Sortierordnung und die zur Sortierung benutzten Felder beeinflusst werden. Die ausgewählten Felder müssen aber alle zur Feldgruppe HEADER gehören.

Sollten Felder, nach denen sortiert werden soll, leer sein, werden diese Felder bzw. Datensätze vor allen anderen eingeordnet. Die Reihenfolge von Datensätzen mit identischem Sortierbegriff ist unbestimmt!

Auswerten des Datenbestands

Ein EXTRACT-Datenbestand wird durch eine LOOP-ENDLOOP-Schleife bearbeitet.

```
LOOP.
   statements
ENDLOOP.
```

Auch hier fehlt, wie bei SORT, nach LOOP ein Name. Dies ist das Zeichen dafür, dass der interne Datenbestand zu bearbeiten ist. Innerhalb der Schleife werden die mit EXTRACT in den Datenbestand geschickten Felder nacheinander bereitgestellt. Die Inhalte des jeweiligen Datensatzes im EXTRACT-Bestand werden zurück in die ursprünglichen Felder übertragen und können dort ausgewertet werden. Dabei werden aber nur Felder verändert, für die im EXTRACT-Bestand auch Daten existieren. Alle anderen Feldinhalte bleiben unverändert! Diese Eigenschaft macht es erforderlich, die Gültigkeit von Daten mittels spezieller Anweisungen zu testen. Dazu dienen die Anweisungen zur Feldgruppenermittlung.

Gruppenbildung und Feldgruppenbezug

Zunächst sind die üblichen – bereits weiter oben beschriebenen – Anweisungen wie AT FIRST oder AT NEW zur Gruppenbildung verfügbar. Sie beziehen sich auf die Datenfelder. Mit der Anweisung

```
AT fieldgroup.
...
ENDAT.
```

wird die Ausführung der Anweisungen innerhalb des Blocks davon abhängig gemacht, ob der aktuelle Datensatz mit einer EXTRACT-Anweisung für eben diese Feldgruppe erzeugt wurde. Eine Variante macht die Ausführung zudem davon abhängig, ob der auf den aktuellen Satz folgende Datensatz einer konkret angegebenen zweiten Feldgruppe entspricht.

```
AT fieldgroup_1 WITH fieldgroup_2.
...
ENDAT.
```

Damit können beispielsweise Kopfsätze ohne Folgesätze ausgeblendet werden.

Diese ziemlich theoretischen Erläuterungen lassen sich mit einem Beispiel recht anschaulich demonstrieren. Im Gegensatz zu anderen Demoprogrammen werden hier der einfacheren Verweise wegen die Zeilennummern mit abgedruckt. Der zur Auswertung ausgewählte Datenbestand wird im Beispiel erzeugt und in internen Tabellen bereitgestellt. Insofern unterscheidet sich das Beispiel vom bereits vorgestellten Pseudocode. Allerdings steht ja nicht das Lesen von Daten aus Tabellen im Mittelpunkt dieses Beispiels, sondern deren Auswertung mit Hilfe eines EXTRACT-Datenbestandes.

```
 1  REPORT  yz434030.
 2
 3  DATA NUM TYPE I.
 4
 5  DATA: BEGIN OF LIEF OCCURS 10,
 6    LNUM(4) TYPE N,
 7    NAME(20),
 8  END OF LIEF.
 9
10  DATA: BEGIN OF RECH OCCURS 10,
11    RECHNR(4) TYPE N,
12    LNUM LIKE LIEF-LNUM,
13  END OF RECH.
14
15  DATA: BEGIN OF RECHZEIL OCCURS 10,
16    RECHNR LIKE RECH-RECHNR,
17    ARTIKEL(20),
18  END OF RECHZEIL.
19
20  LIEF-LNUM   = '1'.   LIEF-NAME = 'forcont'. APPEND LIEF.
21
22  LIEF-LNUM   = '2'.   LIEF-NAME = 'SAP'.  APPEND LIEF.
23
24  RECH-RECHNR = '123'. RECH-LNUM = '1'.    APPEND RECH.
25
26  RECH-RECHNR = '234'. RECH-LNUM = '2'.    APPEND RECH.
27
28  RECH-RECHNR = '345'. RECH-LNUM   = '1'.  APPEND RECH.
29
30  RECHZEIL-RECHNR  = '234'. RECHZEIL-ARTIKEL = 'SAP R/3'.
31  APPEND RECHZEIL.
32
```

```
33  RECHZEIL-RECHNR  = '123'. RECHZEIL-ARTIKEL =
                                              'Development'.
34  APPEND RECHZEIL.
35
36  RECHZEIL-RECHNR  = '123'. RECHZEIL-ARTIKEL = 'factory'.
37  APPEND RECHZEIL.
38
39  RECHZEIL-RECHNR  = '123'. RECHZEIL-ARTIKEL = 'Project
                                                ABC'.
40  APPEND RECHZEIL.
41
42  RECHZEIL-RECHNR  = '345'. RECHZEIL-ARTIKEL = 'factory'.
43  APPEND RECHZEIL.
44
45  FIELD-GROUPS: HEADER, LIEFERANT, ARTIKEL.
46
47  INSERT: LIEF-LNUM RECH-RECHNR NUM INTO HEADER,
48          LIEF-NAME                 INTO LIEFERANT,
49          RECHZEIL-ARTIKEL          INTO ARTIKEL.
50
51  LOOP AT RECH.
52
53    MOVE SPACE TO LIEF.
54    LIEF-LNUM = RECH-LNUM.
55    READ TABLE LIEF.
56    IF SY-SUBRC = 0.
57
58      EXTRACT LIEFERANT.
59      LOOP AT RECHZEIL WHERE RECHNR = RECH-RECHNR.
60        EXTRACT ARTIKEL.
61        ADD 1 TO NUM.
62      ENDLOOP.
63
64    ENDIF.
65
66  ENDLOOP.
67
68  CLEAR: LIEF, RECH, RECHZEIL, NUM.
69
70  LOOP.
71    WRITE: / LIEF-LNUM, RECH-RECHNR,
72             NUM, LIEF-NAME,
73             RECHZEIL-ARTIKEL.
74
75  CLEAR: LIEF, RECH, RECHZEIL, NUM.
76
77  ENDLOOP.
```

```
 78
 79  SORT.
 80
 81  LOOP.
 82
 83    AT LIEFERANT WITH ARTIKEL.
 84      AT NEW LIEF-LNUM.
 85        WRITE: /,  'Lieferant:', LIEF-NAME.
 86      ENDAT.
 87    ENDAT.
 88
 89    AT NEW RECH-RECHNR.
 90      WRITE: / RECH-RECHNR.
 91    ENDAT.
 92
 93    WRITE: /5 RECHZEIL-ARTIKEL.
 94
 95    AT END OF RECH-RECHNR.
 96      WRITE: /.
 97    ENDAT.
 98
 99   CLEAR: LIEF-LNUM, RECH-RECHNR,
100          LIEF-NAME, RECHZEIL-ARTIKEL.
101
102  ENDLOOP.
```

Im Beispiel werden in den Zeilen 3 bis 43 zunächst alle Felder und internen Tabellen deklariert und mit Daten gefüllt. Der Inhalt dieser Tabellen entspricht in etwa dem einleitend erwähnten Problem. Dieser Abschnitt des Programms enthält keine neuen Anweisungen. In Zeile 45 werden 3 Feldgruppen `HEADER`, `LIEFERANT` und `ARTIKEL` deklariert und in den Zeilen 47 bis 49 mit Daten- bzw. Tabellenfeldern verbunden. In die Feldgruppe `HEADER`, deren Inhalt für die Sortierung des Datenbestandes erforderlich ist, werden die Felder für die Lieferantennummer und die Rechnungsnummer aufgenommen. Darüber hinaus wird eine laufende Nummer mitgeführt, die eine geordnete Reihenfolge der einzelnen Datensätze in der Liste gewährleisten soll.

Nach diesen vorbereitenden Arbeiten kann der Aufbau des Datenbestandes folgen. Zwei verschachtelte `LOOP`-Schleifen und eine `READ`-Anweisung in den Zeilen 51 bis 66 simulieren die `SELECT`-Anweisungen des Pseudocodes. Die äußere Schleife bearbeitet alle Rechnungs-Kopfzeilen. Mittels einer `READ`-Anweisung werden zu jedem Rechnungskopf die Daten zum Lieferanten gesucht. Falls diese existieren, wird in Zeile 58 der Inhalt der zur Feldgruppe `LIEFERANT` gehörenden Felder in den `EXTRACT`-Datenbestand gestellt. Dabei werden die Felder der Feldgruppe `HEADER` automatisch mit übertragen. Die Anweisungen in den Zeilen 59 bis 62 lesen die einzelnen Belegzeilen jeder Rechnung und stellen diese ebenfalls als separate Datensätze in den `EXTRACT`-Bestand.

Den Aufbau des EXTRACT-Bestands und den Inhalt der einzelnen Datensätze können Sie dank der Wirkung der Zeilen 68 bis 77 erkennen. Zunächst werden die Datenfelder initialisiert. Anschließend wird der Datenbestand mittels einer LOOP-Schleife ausgelesen und der Inhalt der interessierenden Felder ausgegeben. Da in der LOOP-Schleife nur die Felder aktualisiert werden, die wirklich im Datenbestand enthalten sind, werden die Datenfelder in jedem Schleifendurchlauf initialisiert. Nur so wird deutlich, welche Daten in jedem Datensatz des EXTRACT-Bestands enthalten sind. Dieser Abschnitt des Programms dient nur zur Demonstration des Datenbestands.

Die echte Auswertung der Daten wird in Zeile 79 durch das Sortieren der Datensätze eingeleitet. In einer umfangreicheren Schleife in den Zeilen 81 bis 103 wird dann schließlich der sortierte Datenbestand aufgelistet. Die AT-Anweisung in Zeile 83 sorgt dafür, dass nur Lieferanten-Datensätze bearbeitet werden, auf die ein Artikel-Datensatz folgt. Wenn dies der Fall ist, prüft die darauf folgende AT-Anweisung, ob eine Kopfzeile für den Lieferanten ausgegeben werden muss und erzeugt diese bei Bedarf. Alle weiteren Anweisungen sind leicht verständlich. Im Falle einer neuen Rechnungsnummer wird auch diese angezeigt, anschließend werden alle Belegzeilen dieser Rechnung aufgelistet. Am Ende einer jeden Rechnung erzeugt der Report eine trennende Leerzeile.

Übungen

1. Erstellen Sie einen eigenen Report für das zweite Gruppenwechsel-Beispiel und führen Sie ihn aus.
2. Ändern Sie im ersten Beispielreport zum Gruppenwechsel die Reihenfolge der Anweisungen AT FIRST und WRITE. Führen Sie den Report aus und analysieren Sie das Ergebnis.
3. Schreiben Sie das erste Beispiel so um, dass das Datum anstelle des Namens zur Gruppenbildung benutzt wird.
4. Ändern Sie im zweiten Beispiel die Reihenfolge der AT-Anweisungen. Analysieren Sie das Ergebnis.
5. Entfernen Sie im Beispiel zum EXTRACT-Kommando das Feld NUM (Zeile 47) aus der Deklaration der Feldgruppe. Vergleichen Sie das Ergebnis mit dem des unveränderten Programms.
6. Entfernen Sie im Beispiel zum EXTRACT-Kommando die Anweisung CLEAR in der ersten Schleife (Zeile 75). Vergleichen Sie die Inhalte der Feldleiste im Vergleich zur Ausführung mit CLEAR.
7. Entfernen Sie im Beispiel zum EXTRACT-Kommando die Zeile 26. Es existiert dadurch ein Lieferant ohne Lieferung. Was passiert?

3.4.2 Ereigniskonzept im Standardreporting

Die Programmiersprache ABAP arbeitet ereignisgesteuert. Die interne Ablauflogik des Systems unterscheidet bei Abarbeitung eines Programms – sowohl bei Reports als auch bei dialogorientierten Anwendungen – verschiedene Zeitpunkte wie z.B. die Beendigung der Eingabe in ein Dynpro bzw. erkennt spezielle Ereignisse, etwa das Erreichen des Seitenendes einer Liste. Es existiert eine Reihe von Ereignissen unterschiedlicher Typen. Der Programmierer kann den Ereignissen einen speziellen Programmcode zuordnen, muss es aber nicht. Die Ereignissteuerung erinnert vom Ansatz her zunächst an die Programmierung unter grafischen Oberflächen wie z.B. Windows. Bei näherer Betrachtung werden aber erhebliche Unterschiede in der Realisierung der Ereignissteuerung ersichtlich. Eventuell vorhandene Kenntnisse aus dem Windows-Bereich sind hier wenig hilfreich.

Die Verbindung von Ereignis zu Programmcode erfolgt über so genannte Zeitpunktanweisungen. Tritt in einer Anwendung ein Ereignis auf, so sucht die Steuerlogik im Programm nach einer zu diesem Ereignis gehörenden Zeitpunktanweisung und arbeitet den auf diese Anweisung folgenden Programmcode ab. Das Ende des Anweisungsblocks wird dabei durch das Programmende, eine andere Zeitpunktanweisung oder die Definition eines Unterprogramms, also die Anweisung `FORM`, gebildet.

In der SAP-Dokumentation werden die bereits beschriebenen Anweisungen zur Gruppenbildung im Zusammenhang mit echten Zeitpunktanweisungen erwähnt. Trotz nahezu identischer Syntax bestehen wesentliche Unterschiede bezüglich der Funktionalität. Aus diesem Grund werden beide Anweisungstypen getrennt behandelt.

Zeitpunktanweisungen lassen sich sehr genau bestimmten Abschnitten oder Teilen einer Anwendung zuordnen. Im Standardreporting stehen einige Zeitpunkte zur Verfügung, die hauptsächlich durch spezielle Ereignisse während der Ausgabe auf die Liste und mit der Bearbeitung des Selektionsbildschirms angesprochen werden. Die einfachsten und sehr häufig verwendeten Zeitpunkte sind

```
TOP-OF-PAGE.
```

und

```
END-OF-PAGE.
```

Wie der Name bereits sagt, werden diese Zeitpunkte bzw. die dazugehörigen Anweisungen dann ausgeführt, wenn eine neue Seite begonnen oder eine Seite abgeschlossen wird. Auf diese Weise können Überschriften oder Fußzeilen in die Liste eingefügt werden.

Ein Report kann mit so genannten logischen Datenbanken zusammenarbeiten. Dies sind Elemente, die eine Datenbeschaffung in einer oder mehreren Tabellen durchführen, dabei Verknüpfungen zwischen den Tabellen beachten und die gelesenen Daten in einer definierten Reihenfolge an den eigentlichen Report übergeben. Der Report als solcher übernimmt dann nur noch die Auswertung und Ausgabe der Daten, die durch die logische Datenbank zur Verfügung gestellt werden. Auf logische Datenbanken wird wegen der Komplexität des Themas in einem eigenen Unterabschnitt eingegangen. Die bisher verwendeten Reports verwendeten keine logische Datenbank. Unmittelbar zu Beginn eines Programms und vor dem Lesen des ersten Satzes einer logischen Datenbank wird das Ereignis

```
START-OF-SELECTION.
```

ausgelöst, nach dem Lesen des letzten Datensatzes das Ereignis

```
END-OF-SELECTION.
```

Sollte der Report nicht mit einer logischen Datenbank arbeiten, ist der Zeitpunkt `START-OF-SELECTION` – falls explizit angegeben – gleichbedeutend mit dem Beginn des eigentlichen Hauptabschnitts des Reports. Existiert in einer Anwendung die Zeitpunktanweisung `START-OF-SELECTION`, werden zunächst die Anweisungen ab `REPORT` oder `PROGRAM`, dann die Anweisungen ab der expliziten Startanweisung ausgeführt. Eine `START-OF-SELECTION`-Anweisung in Programmen, die keine logische Datenbank verwenden, ist vor allem erforderlich, um das „Hauptprogramm" von den Anweisungen zu weiteren Zeitpunkten, z.B. `TOP-OF-PAGE`, abzugrenzen.

3.4.3 Selektionen und Parameter

Das SAP-System ist für die Bearbeitung sehr großer Datenmengen vorgesehen. Die Zahl der Datensätze in einer Tabelle kann durchaus die Millionengrenze überschreiten. Listen mit einer so großen Zahl von Einträgen sind praktisch wertlos, Hilfsmittel zur Selektion spezieller Datensätze werden zwingend erforderlich. Das eigentliche Selektionswerkzeug, die `SELECT`-Anweisung, wurde bereits vorgestellt. Aber diese Anweisung erfordert Parameter in der `WHERE`-Klausel, die der Report wegen fehlender Dialogfähigkeit nicht ohne weiteres vom Anwender erfragen kann. Einen Ausweg bietet der Selektionsbildschirm. Dies ist eine automatisch generierte Eingabemaske, in die der Anwender eigene Selektionskriterien eintragen kann. Abbildung 3.56 zeigt als Beispiel für einen derartigen Selektionsbildschirm den des Infosystems der Entwicklungsumgebung.

Welche Eingabefelder im Selektionsschirm erscheinen sollen, entscheidet der Programmierer durch zwei spezielle Anweisungen im Quelltext des Reports. Das Selektions-Dynpro wird dann ohne weiteres Zutun des Programmierers durch die interne Steuerlogik des Systems erzeugt und vor der Ausführung des Reports abgearbeitet.

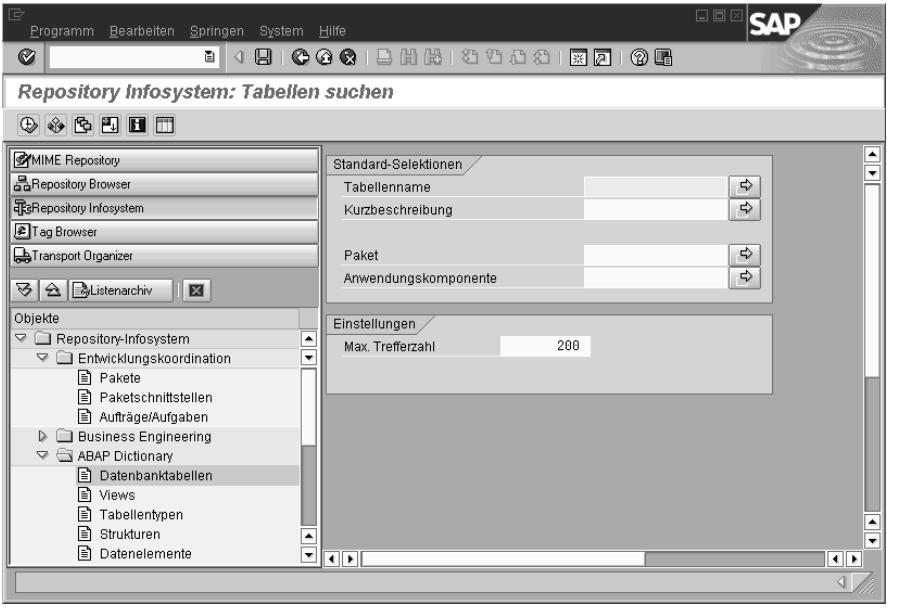

Abbildung 3.56 **© SAP AG**
Selektions-Dynpro eines Reports

Das Aussehen des Selektionsbildschirmes kann in gewissen Grenzen modifiziert werden. Dies erfolgt durch spezielle Kommandos, gegebenenfalls auch in Verbindung mit Zeitpunktanweisungen.

Bis zur Version 3.x waren Selektionsbilder den Reports vorbehalten. Ab Version 4.0 können Sie in allen Programmarten Selektionsbilder definieren und benutzen. An den nachfolgend beschriebenen Kommandos ändert sich dadurch nichts. Allerdings kommen einige neue Anweisungen zur Definition und zum Aufruf der Selektionsbilder hinzu, deren Erklärung Sie am Ende dieses Abschnitts finden.

Die Bedienung des Selektionsbildschirmes ist erklärungsbedürftig. In die diversen Eingabefelder werden zunächst die gewünschten Suchbegriffe eingetragen. Mit der Menüfunktion SPRINGEN | VARIANTEN | ALS VARI. SICHERN kann der gegenwärtige Inhalt des Selektionsschirms unter einem eindeutigen Namen als so genannte Variante gesichert werden. Alle so gesicherten Varianten sind über die Menüfunktion SPRINGEN | VARIANTEN | HOLEN oder die gleichnamige Drucktaste (sofern aktiviert) später wieder zugänglich.

Die Ausführung des Reports erfolgt nur, wenn dies per Kommando explizit gefordert wird. Dazu stehen die drei Menüfunktionen PROGRAMM | AUSFÜHREN, PROGRAMM | AUSFÜHREN + DRUCKEN und PROGRAMM | IM HINTERGRUND AUSFÜHREN sowie eine Drucktaste in der Drucktastenzeile zur Verfügung.

Parameter

Die einfachste Variante, einem Report einen Wert zu übergeben, sind die so genannten Parameter. Ein Parameter wird im Report mit der Anweisung

```
PARAMETERS parameter.
```

erzeugt. Mittels des Doppelpunkt-Mechanismus ist die gleichzeitige Deklaration mehrerer Parameter möglich. Ein Parameter erscheint im Selektionsbildschirm als einfaches Eingabefeld (siehe Abbildung 3.60, MAXIMALE TREFFERZAHL), in das ein Wert eingetragen werden kann. Der Name eines Parameters muss den gleichen Bedingungen genügen wie der eines Datenfeldes. Außerdem darf er maximal acht Zeichen lang sein. Innerhalb des Reports kann ein Parameter wie ein einfaches Datenfeld benutzt werden. Anders ausgedrückt, ein Parameter ist ein Datenfeld, das vom Anwender des Reports vor dessen Ausführung in einer Eingabemaske mit einem Wert belegt werden kann.

Wie bereits beschrieben, besitzt ein Feld einige Eigenschaften wie Typ und Länge. Natürlich muss auch ein Parameter solche Eigenschaften besitzen, da er ja im Report auch nur ein Datenfeld ist. Diese Eigenschaften können mit diversen Zusätzen zur `PARAMETERS`-Anweisung definiert werden. Fehlen diese, wird ähnlich wie bei `DATA` ein Parameter mit dem Datentyp C und der Länge 1 angelegt. Am einfachsten und häufigsten ist der Bezug auf ein existierendes Feld mit `LIKE`, z.B. in der Form

```
PARAMETERS parameter LIKE field.
```

Als Bezugsfeld dient oft ein Tabellenfeld, was später eine einfache Auswertung des Parameters ermöglicht. Falls für das Tabellenfeld so genannte Fremdschlüsselbeziehungen gepflegt wurden, stehen automatisch eine Eingabehilfe mit F4 und eine Wertprüfung bereit. Neben `LIKE` kann auch der Bezug auf einen vordefinierten oder selbst deklarierten Datentyp erfolgen:

```
PARAMETERS parameter TYPE type.
```

Da keine Einschränkungen bezüglich des Datentyps bestehen, der Datentyp P für gepackte Zahlen aber zusätzlich noch die Angabe der Nachkommastellen erfordert, existiert für diesen Fall ein spezieller Zusatz:

```
PARAMETERS parameter TYPE P DECIMALS decimal_places.
```

Für einige Parameter kann es sinnvoll sein, Vorgabewerte zu setzten. Diese werden bei Deklaration eines Parameters mit dem Zusatz `DEFAULT` übergeben, z.B.

```
PARAMETERS parameter DEFAULT default_value.
```

Soll in ein Parameterfeld unbedingt eine Eingabe durch den Anwender erfolgen, kann dies durch den Zusatz `OBLIGATORY` erzwungen werden.

```
PARAMETERS parameter OBLIGATORY.
```

Neben alphanummerischen Eingabefeldern kennen grafische Oberflächen auch so genannte Ankreuzfelder (Checkboxes) und Auswahlfelder (Radiobuttons). Erstere sind Felder, die unabhängig voneinander markiert (angekreuzt) werden können. Sie erlauben so das Ankreuzen mehrerer Einträge einer Liste o. Ä. Auswahlfelder hingegen werden zu Gruppen zusammengefasst. In einer Gruppe kann stets nur eines der Auswahlfelder markiert werden. Auf diese Weise ist die Wahl aus einer Menge von Objekten möglich.

Derartige Felder können auch auf Selektionsbildern erzeugt werden. Für Ankreuzfelder dient dazu die Variante

```
PARAMETERS parameter AS CHECKBOX.
```

für Auswahlfelder hingegen

```
PARAMETERS parameter RADIOBUTTON GROUP group.
```

Im letzteren Fall ist für alle zusammengehörigen Felder ein identischer Gruppenname anzugeben. Nach der Eingabe in ein Selektionsbild haben die markierten Felder den Inhalt „X", die anderen enthalten ein Leerzeichen. Das nachfolgende Beispiel demonstriert einige Parameterdeklarationen. Dieses Beispiel kann ausgeführt werden. Nach dem Start erscheint der Parameterbildschirm, in dem Werte eingetragen werden können. Anschließend ist die Menüfunktion PROGRAMM | AUSFÜHREN zu wählen. Erst dadurch wird die Eingabe im Parameterbildschirm beendet und der eigentliche Report ausgeführt. Dieser zeigt die Inhalte aller Parameter an.

```
REPORT yz434030 NO STANDARD PAGE HEADING.

TABLES: tadir.
DATA m_feld.

PARAMETERS p_einf.
PARAMETERS p_feld LIKE tadir-author.
PARAMETERS p_kurs TYPE p DECIMALS 2.
PARAMETERS p_obl LIKE m_feld OBLIGATORY.

PARAMETERS: c1 AS CHECKBOX,
            c2 AS CHECKBOX,
            c3 AS CHECKBOX.

PARAMETERS: r1 RADIOBUTTON GROUP g1,
            r2 RADIOBUTTON GROUP g1,
            r3 RADIOBUTTON GROUP g1.

START-OF-SELECTION.
  WRITE: / 'P_EINF:', p_einf.
  WRITE: / 'P_FELD:', p_feld.
```

```
WRITE: / 'P_KURS:', p_kurs.
WRITE: / 'P_OBL :', p_obl.
WRITE: /.
WRITE: / 'Checkbox   :', c1, c2, c3.

WRITE: /.
WRITE: / 'Radiobutton:', r1, r2, r3.
```

Neben den bisher erwähnten Optionen zum Kommando PARAMETERS stehen weitere zur Verfügung, wobei die Anzahl der unterstützten Optionen von der Version der R/3-Software abhängig ist. Eine Aufstellung der in Version 4.6B verfügbaren Zusätze finden Sie in Tabelle 3.30. Einige der Optionen können bzw. müssen miteinander kombiniert werden. Außerdem zieht die Verwendung einiger der Optionen zusätzliche Programmiertätigkeit nach sich.

Option	Beschreibung
DEFAULT value	Vorgabewert setzen
TYPE type	Parameter mit angegebenem Datentyp
LENGTH len	Vorgabe einer Feldlänge
DECIMALS dec	Dezimalstellen für Datentyp P
LIKE field	Parameter durch anderes Feld beschreiben
MEMORY ID pid	Parameter mit Get/Set-Parameter verbinden
MATCHCODE OBJECT mobj	Eingabehilfe durch Matchcode
MODIF ID id	Modifikationsgruppe festlegen
NO-DISPLAY	Keine Anzeige auf Selektionsbild, aber Datenübergabe bei SUBMIT möglich
LOWER CASE	Groß- / Kleinschreibung ermöglichen
OBLIGATORY	Muss-Eingabe
AS CHECKBOX	Darstellung als Ankreuzfeld
RADIOBUTTON GROUP radi	Darstellung als Auswahlfeld
FOR TABLE dbtab	Parameter für Datenbanktabelle
FOR NODE node	Parameter für Knoten einer logischen Datenbank
AS SEARCH PATTERN	Parameter füllt Suchhilfe-Tabelle

Tabelle 3.30
Optionen des Kommandos PARAMETERS

Option	Beschreibung
VALUE-REQUEST	Werthilfe über F4-Taste möglich
HELP-REQUEST	Allgemeine Hilfe über F1 möglich
VISIBLE LENGTH len	Ausgabelänge setzen
VALUE CHECK	Feldinhalt gegen Wertebereich eines Tabellenfeldes prüfen
LIKE (g)	Parameter-Eigenschaften dynamisch beschreiben
AS LISTBOX	Darstellung als Drop-Down-Listbox
USER-COMMAND ucom	Bei Änderung Unterprogramm aufrufen.

Tabelle 3.30
Optionen des Kommandos PARAMETERS (Fortsetzung)

Selektionen

Neben den einfachen Parametern stellt ABAP so genannte Selektionstabellen zur Verfügung. Selektionstabellen sind zunächst interne Tabellen mit einem vorgegebenen Aufbau (siehe Tabelle 3.31).

Feld	Aufgabe	Feldlänge
SIGN	Kennzeichen, ob selektierte Datensätze zur Ergebnismenge gehören (I) oder von dieser ausgeschlossen werden sollen (E)	1
OPTION	Vergleichsoperator (EQ, NE, CP, NP, GE, LT, LE, GT)	2
LOW	Muster oder unterer Grenzwert eines Bereiches	Wie Datenfeld
HIGH	Obere Grenze eines Bereichs	Wie Datenfeld

Tabelle 3.31
Aufbau von Selektionstabellen

Über das Selektionsbild wird die Selektionstabelle mit einem oder mehreren Einzelmustern oder Bereichsangaben gefüllt. Die dazu notwendige Funktionalität wird durch die interne Steuerlogik bereitgestellt, es sind keinerlei Programmierarbeiten erforderlich. Im Selektionsbild stellt das System für eine Selektionstabelle zunächst maximal zwei Eingabefelder bereit. Nach einem Klick auf die Schaltfläche neben dem am weitesten rechts liegenden Eingabefeld er-

scheint, je nach Releasestand, ein zusätzliches Dynpro, in dem weitere Datensätze für die Selektionstabelle erfasst werden können, oder ein Popup, wie sie Abbildung 3.57 zeigt. Hier können die Suchbegriffe in separaten Registerkarten als Einzelwert oder Bereichsangabe eingegeben werden. Neben der Suche auf Übereinstimmung (grüne Markierung an der Registerkarte) ist auch die Angabe von Ausschlusskriterien (rote Markierung) möglich. Die hier eingetragenen Werte finden sich später in den Spalten `LOW` (Einzelwert bzw. untere Grenze) und `HIGH` (obere Grenze) der Selektionstabelle wieder. Die Einträge, für die auf Übereinstimmung gesucht wird, erhalten in der Selektionstabelle im Feld `SIGN` den Wert „I“, Ausschlusskriterien werden durch ein „E“ in diesem Feld gekennzeichnet.

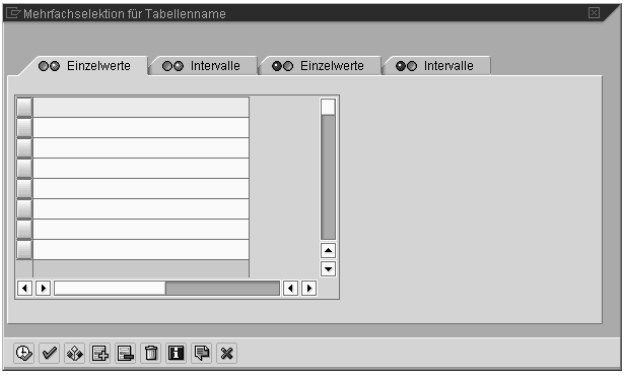

Abbildung 3.57 **© SAP AG**
Einfügen von Werten in eine Selektionstabelle

Durch einen Klick auf das Markiersymbol einer Tabellenzeile bzw. die Schaltfläche ÜBERNEHMEN gelangen Sie zu einem zweiten Popup (Abbildung 3.58), in dem ein Vergleichsoperator für die aktuelle Zeile ausgewählt werden kann.

Abbildung 3.58 **© SAP AG**
Vergleichsoperator wählen

Die Vergleichsoperatoren werden als zweistelliges Kürzel im Feld OPTION der Selektionstabelle abgelegt.

Erzeugt werden Selektionen ähnlich wie Parameter. Die Deklarationsanweisung lautet hier

```
SELECT-OPTIONS selection FOR field | (field).
```

Für die Namen von Selektionen gilt wiederum die Einschränkung auf die Länge von maximal acht Zeichen. Die obligatorische Bezugnahme auf ein Daten- oder Tabellenfeld bestimmt einige Eigenschaften der Felder LOW und HIGH der Selektionstabelle und damit der Eingabefelder auf dem Selektionsbildschirm wie z. B. die Länge. Besitzt das Datenbankfeld eine Länge von drei Zeichen, ist auch das Eingabefeld auf dem Selektionsbildschirm nur drei Zeichen lang. Das Feld, für das eine Selektion angelegt werden soll, kann direkt oder dynamisch übergeben werden.

Bei Bestätigung der Eingabe wird geprüft, ob die Werte oder Muster dem Datentyp des referierten Feldes entsprechen. In Selektionen, die mit einem Feld des Typs N (nummerische Zeichenkette) verbunden wurden, können beispielsweise nur die zugelassenen nummerischen Zeichen eingetragen werden. Alle anderen Zeichen führen zu einer Fehlermeldung durch das System. Außerdem ermöglicht die in Kürze zu beschreibende Anweisung CHECK eine Kurzschreibweise bei der Auswertung einer Selektionstabelle, bei der stets das Feld aus der Deklarationsanweisung geprüft wird.

Auch das Kommando SELECT-OPTIONS kennt eine Reihe von Optionen, von denen einige hier näher vorgestellt werden sollen.

```
... DEFAULT value_low [ TO value_high ] [ OPTION op ] [ SIGN s ]
```

Für eine Selektion können Sie einen Vorgabewert setzen. Da eine Zeile der Selektionstabelle aus vier Einzelfeldern besteht, verfügt die entsprechende Option auch über bis zu vier Parameter. Mittels

```
... OBLIGATORY
```

erzwingen Sie eine Eingabe für die untere Grenze der Selektion. Die anderen Optionen entnehmen Sie bitte Tabelle 3.32.

Nicht immer ist es möglich oder sinnvoll, Selektionstabellen durch den Anwender füllen zu lassen. Die Performance von SELECT-Anweisungen kann beispielsweise durch Selektionstabellen optimiert werden, da über Selektionstabellen eine recht detaillierte Vorauswahl der Datensätze durch die Datenbank ermöglicht wird. Selektionstabellen können daher auch innerhalb einer Anwendung durch das Programm selbst erzeugt und gefüllt werden, ohne dass ein Selektionsbild notwendig ist. Die Deklaration einer Selektionstabelle erfolgt mittels der Anweisung

```
RANGES table FOR field.
```

Option	Bemerkung
`DEFAULT v_low [ TO v_high ] [ OPTION op ] [ SIGN s ]`	Vorgabewerte setzen.
`MEMORY ID pid`	Selektion mit Get/Set-Parameter verbinden.
`MATCHCODE OBJECT mobj`	Selektion mit Matchcode verbinden.
`MODIF ID id`	Modifikationsgruppe zuweisen.
`NO-DISPLAY`	Keine Anzeige im Selektionsbild. Wertübergabe bei `SUBMIT` möglich.
`LOWER CASE`	Groß-/Kleinschreibung ermöglichen.
`OBLIGATORY`	Muss-Eingabe
`NO-EXTENSION`	Es kann nur eine Zeile erfasst werden.
`NO INTERVALS`	Nur das `LOW`-Feld erscheint auf dem ersten Selektionsbildschirm, alle weiteren Eingaben erfolgen im zweiten Selektionsbild.
`NO DATABASE SELECTION`	Keine freie Abgrenzung in der logischen Datenbank.
`VALUE-REQUEST [ FOR LOW | HIGH ]`	Werthilfe mit F4 ermöglichen.
`HELP-REQUEST [ FOR LOW | HIGH ]`	Allgemeine Hilfe mit F1 ermöglichen.
`VISIBLE LENGTH len`	Ausgabelänge setzen.

Tabelle 3.32
Optionen des Kommandos SELECT-OPTIONS

Sie erzeugt eine Selektionstabelle mit dem vorgegebenen Aufbau. Gefüllt wird diese mit den üblichen Anweisungen für interne Tabellen, vorrangig also mit `APPEND`.

Ab 4.x steht ein weiteres Kommando zur Verfügung, um Selektionstabellen anzulegen. Es fügt sich syntaktisch besser in das allgemeine Datenkonzept ein, da es sich um eine Variante der `DATA`-Anweisung handelt.

```
DATA seltab ( TYPE | LIKE ) RANGE OF ( type | field )
  [ INITIAL SIZE n   ]
  [ WITH HEADER LINE ].
```

Ebenfalls ab Version 4.x besteht die Möglichkeit, für Selektionstabellen auch Typdefinitionen zu erzeugen. Dazu dient das Kommando

```
TYPES seltab_type ( TYPE | LIKE ) RANGE OF ( type | field )
  [ INITIAL SIZE n ].
```

Auswertung von Parametern und Selektionen

Der bevorzugte Einsatzzweck von Selektionen und Parametern ist die Übergabe von Suchbegriffen an den Report. Diese müssen natürlich im Report bei der Selektion von Datensätzen ausgewertet werden. Dazu existieren im Wesentlichen zwei Möglichkeiten:

- Einbau in die WHERE-Klausel.
- Verwendung in der CHECK-Anweisung.

Die CHECK-Anweisung wurde bereits erwähnt. Sie ermöglicht es, Parameter oder Selektionen innerhalb beliebiger Schleifen auszuwerten. Diese Schleifen können – müssen aber nicht – SELECT-ENDSELECT-Schleifen sein. Bezüglich der Performance ist es allerdings wesentlich günstiger, die Auswahl von Datensätzen durch entsprechende Gestaltung der SELECT-Anweisung auszuführen und nicht innerhalb der Schleife mit CHECK.

Wenn eine Selektion mittels LIKE von einem Datenbankfeld abgeleitet wurde, kann für die Notation der Bedingung eine Kurzschreibweise benutzt werden.

Die Anweisung

```
CHECK selection.
```

stellt in einer SELECT-Schleife den Bezug zum Datenbankfeld automatisch her. Sie entspricht der Langform

```
CHECK table_field IN selection.
```

Parameter können in einem direkten Vergleich verwendet werden, wobei verschiedene Vergleichsoperatoren (siehe Anweisung IF) möglich sind.

```
CHECK table_field = parameter.
```

Parameter können aber auch in einer WHERE-Klausel stehen. Sie werden dort wie andere Felder oder direkt kodierte Konstanten benutzt. Da in Parametern zumeist Suchmuster eingetragen werden, die auch Musterzeichen enthalten können, kommt als Vergleichsoperator in vielen Fällen nur die Anweisung LIKE ernsthaft in Betracht. Diese Anweisung erwartet als Musterzeichen die beiden Zeichen „%“ bzw. „_“, die üblicherweise durch den Anwender verwendeten

Musterzeichen sind allerdings der Stern „*“ und das Pluszeichen „+“. Vor Nutzung eines Parameters in der WHERE-Klausel sind die Musterzeichen „*“ und „+“ durch „%“ und „_“ zu ersetzen. Dazu stellt ABAP die Anweisung TRANSLATE zur Verfügung. Die Syntax dieser Anweisung lautet:

```
TRANSLATE string USING pattern.
```

Die Anweisung TRANSLATE ersetzt in einer Zeichenkette Zeichen gemäß der Musterzeichenkette. In der Musterzeichenkette stehen das zu ersetzende und das neue Zeichen paarweise nacheinander. Um in einem Parameter die Musterzeichen zu tauschen, ist die Anweisung

```
TRANSLATE parameter USING '*%+_'.
```

erforderlich. Für die Musterzeichen der Selektionstabelle, welche in der WHERE-Klausel mit der Anweisung IN geprüft werden kann, ist eine derartige Transformation nicht nötig. Bei der Verwendung von Selektionstabellen ist eine Beschränkung seitens des Systems zu beachten. Die Inhalte in der Selektionstabelle werden durch die interne Ablauflogik des Systems zum Aufbau einer echten SQL-SELECT-Anweisung verwendet, die aus mehreren einzelnen Bedingungen besteht. Diese Anweisung darf eine bestimmte Größe (zur Zeit etwa 8 KByte) nicht übersteigen. Die Zahl der Datensätze in einer Selektionstabelle ist daher zumindest theoretisch begrenzt. Die Verwendung von Selektionstabellen bietet gegenüber den Prüfungen mit CHECK innerhalb einer SELECT-ENDSELECT-Schleife allerdings eine wesentlich bessere Performance, da weniger Datensätze auf der Datenbank selektiert und per Netz übertragen werden müssen. Das folgende Beispiel demonstriert die beiden wichtigsten Möglichkeiten, Parameter bzw. Selektionen auszuwerten.

```
REPORT yz434050.
DATA tadir TYPE tadir.

PARAMETERS p_autor LIKE tadir-author OBLIGATORY.
SELECT-OPTIONS s_name FOR tadir-obj_name.

TRANSLATE p_autor  USING '*%+_'.

SELECT * FROM tadir
  WHERE author LIKE p_autor.
  CHECK s_name.
  WRITE: / tadir-pgmid, tadir-obj_name.
ENDSELECT.
```

Dieses Programm ermittelt über die SELECT-Anweisung zunächst alle Programme der Programmierer, deren Name auf das Muster im Parameter p_autor passt, und prüft diese dann einzeln darauf, ob sie im Namensbereich liegen, den die Selektionstabelle bestimmt. Beachten Sie, dass im Parameter für den Namen unbedingt eine Eingabe erfolgen sollte. Bleibt das Feld leer, findet die SELECT-

Anweisung keine Datensätze. Praktischerweise sollte der Parameter daher als OBLIGATORY gekennzeichnet werden oder aber mit DEFAULT einen Vorgabewert erhalten.

Natürlich können Sie beide Prüfungen in die SELECT-Anweisung legen oder mit CHECK realisieren. Dies hat zum Teil erheblichen Einfluss auf das Laufzeitverhalten. Auch die Vergleichsbedingungen müssen dazu umformuliert werden. Das nachfolgende Listing zeigt eine zweite Variante desselben Programms:

```
REPORT yz434060.
DATA tadir TYPE tadir.

PARAMETERS p_autor LIKE tadir-author.
SELECT-OPTIONS s_name FOR tadir-obj_name.

SELECT * FROM tadir
  WHERE obj_name IN s_name.
  CHECK tadir-author CP p_autor.
  WRITE: / tadir-pgmid, tadir-obj_name.
ENDSELECT.
```

Neben den alphanummerischen Eingaben können Parameter auch Auswahl- und Ankreuzfelder bereitstellen. Die Parameter enthalten dann entweder ein Leerzeichen oder ein „X“. Derartige Parameter dienen natürlich nicht direkt zur Datenbankselektion. Sie werden vielmehr in Steueranweisungen wie IF oder CASE zur Modifikation des Programmablaufs ausgewertet.

Bearbeiten des Selektionsbildschirms

Aus den bisher vorgestellten Beispielen wird ersichtlich, dass die einzelnen Parameter und Selektionen in der Reihenfolge ihrer Deklaration jeweils in einer einzelnen Zeile auf dem Selektionsbildschirm erscheinen. Als Bezeichner dient jeweils der programminterne Name des Parameters oder der Selektion.

Einige Anweisungen erlauben es, das Erscheinungsbild des Selektionsbildschirms zu beeinflussen. Zunächst ist es möglich, den Selektionen und Parametern erläuternde Texte zuzuordnen. Diese werden mit der Transaktion gepflegt, mit der auch die nummerierten Textelemente erfasst werden können. Es handelt sich um die Transaktion SE32. Sie kann z.B. vom Editor aus mit der Menüfunktion SPRINGEN | TEXTELEMENTE erreicht werden, was das Erfassen von Texten während der Bearbeitung des Programms ermöglicht. Die so erfassten Texte können ebenfalls übersetzt werden, wodurch Selektionsbildschirme mehrsprachig werden.

Weitere Kommandos erlauben es, die Anordnung der Felder auf dem Bildschirm zu beeinflussen. Die Gestaltung des Selektionsbildschirms erfolgt über verschiedene Varianten des Kommandos SELECTION-SCREEN. Die beiden Kommandos

```
SELECTION-SCREEN BEGIN OF LINE.
...
SELECTION-SCREEN END OF LINE.
```

bilden einen Anweisungsblock, in dem der Zeilenvorschub ausgeschaltet ist. Alle in diesem Block deklarierten Parameter erscheinen in einer Zeile. Selektionen dürfen in einem solchen Block nicht deklariert werden. Um die Eingabefelder innerhalb einer Zeile horizontal ausrichten zu können, wird innerhalb des oben erwähnten Blocks mit

```
SELECTION-SCREEN POSITION position.
```

die Ausgabeposition für eine nachfolgende Parameterdeklaration gesetzt. Unmittelbar nach einer solchen Anweisung muss also Sinnvollerweise ein Parameter deklariert werden. Bei dieser Form der Positionierung gehen die Bezeichner allerdings verloren. Eine weitere Variante des `SELECTION-SCREEN`-Kommandos erlaubt daher die Platzierung von Text in Selektionsbildern. Das Kommando lautet:

```
SELECTION-SCREEN COMMENT format text.
```

Für die Formatangabe sind zwei Varianten möglich. Die erste entspricht der schon bei `WRITE` erläuterten Angabe von Länge und Ausgabespalte. Als zweite Möglichkeit stehen für die Positionen, in denen das `LOW`- und das `HIGH`-Feld einer Selektion angezeigt werden, die Symbole `POS_LOW` und `POS_HIGH` zur Verfügung. Die letztgenannte Variante kann auch bei `SELECTION-SCREEN POSITION` genutzt werden.

Der Text kann ebenfalls auf zwei Arten definiert werden. Zum einen kann für den Text ein Textelement (kein Selektionstext!) stehen. Dessen Wert wird über die bereits beschriebene Methode gepflegt. Er ist statisch, Modifikationen zur Laufzeit sind nicht möglich. Die zweite Variante besteht in der Angabe eines Feldnamens, der an dieser Stelle aber maximal acht Zeichen lang sein darf. Dieses Feld wird durch die `SELECTION-SCREEN`-Anweisung automatisch deklariert! Zur Laufzeit muss das Textfeld vor Darstellung des Selektionsbilds mit einem Wert belegt werden. Dies ist nur durch Verwendung selektionsbildbezogener Zeitpunktanweisungen (z.B. `INITIALIZATION`) möglich. Die Demonstration dieser Form der Anweisung erfolgt daher erst im nächsten Abschnitt.

Umfangreiche Selektionsbildschirme können durch Leerzeilen oder Trennstriche aufgelockert werden. Mit

```
SELECTION-SCREEN SKIP n.
```

werden *n* Leerzeilen eingefügt. Fehlt die Angabe der Leerzeilen, wird als Vorgabewert 1 verwendet. Die Anweisung

```
SELECTION-SCREEN ULINE format.
```

hingegen fügt einen Trennstrich ein. Die Formatangabe entspricht der bei `SELECTION-SCREEN COMMENT` verwendbaren. Das nachfolgende Beispiel demonstriert einige einfache Anweisungen zur Gestaltung des Selektionsbildschirms. Es besitzt keinerlei Funktionalität, erzeugt also keine Ausgabeliste.

```
REPORT yz434070.
SELECTION-SCREEN SKIP.

SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN COMMENT 5(10) t1.
SELECTION-SCREEN POSITION 20.
PARAMETERS c1 AS CHECKBOX.

SELECTION-SCREEN POSITION 40.
PARAMETERS c2 AS CHECKBOX.

SELECTION-SCREEN POSITION 60.
PARAMETERS c3 AS CHECKBOX.
SELECTION-SCREEN END OF LINE.

SELECTION-SCREEN ULINE.

SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN COMMENT 5(15) t2.
SELECTION-SCREEN POSITION pos_low.
PARAMETERS r1 RADIOBUTTON GROUP g1.

SELECTION-SCREEN POSITION pos_high.
PARAMETERS r2 RADIOBUTTON GROUP g1.
SELECTION-SCREEN END OF LINE.

INITIALIZATION.
  t1 = 'Checkbox'.
  t2 = 'Radio Button'.
```

Eine weitere Gestaltungsmöglichkeit stellt die Zusammenfassung von Feldern zu Gruppen dar. Derartige Gruppen können durch einen Rahmen voneinander abgegrenzt werden. Gruppen sowie Rahmen können ineinander geschachtelt werden, wobei momentan bis zu fünf Ebenen möglich sind. Die Abmessungen der Rahmen werden vom System automatisch festgelegt. Die Bildung von Gruppen ermöglicht zunächst die übersichtlichere Gestaltung von Selektionsbildschirmen. Darüber hinaus können mit Hilfe spezieller Zeitpunktanweisungen (siehe folgendes Kapitel) Programmaktivitäten speziell für eine Gruppe von Parametern oder Selektionen ausgeführt werden. Die Gruppierung erfolgt mit den Kommandos

```
SELECTION-SCREEN BEGIN OF BLOCK block.
```

und

```
SELECTION-SCREEN END OF BLOCK block.
```

Durch diese beiden Kommandos erfolgt lediglich eine Gruppierung der eingeschlossenen Parameter oder Selektionen. Der Zusatz

```
... WITH FRAME
```

zur BEGIN OF BLOCK-Anweisung erzeugt einen Rahmen. Ein Titel wird schließlich durch

```
... WITH TITLE title.
```

eingefügt. Der Titel wird dabei auf dieselbe Art und Weise erzeugt wie ein Kommentar, also entweder über ein nummeriertes Textelement oder ein automatisch erzeugtes Datenelement. Ein einfaches Beispiel dazu zeigt das nächste Listing.

```
REPORT yz434080.

SELECTION-SCREEN BEGIN OF BLOCK a WITH FRAME.

SELECTION-SCREEN BEGIN OF BLOCK b WITH FRAME TITLE t1.
PARAMETERS p1.
SELECTION-SCREEN END OF BLOCK b.

SELECTION-SCREEN BEGIN OF BLOCK c WITH FRAME TITLE t2.
PARAMETERS: r1 RADIOBUTTON GROUP g1,
            r2 RADIOBUTTON GROUP g1.
SELECTION-SCREEN END OF BLOCK c.

SELECTION-SCREEN END OF BLOCK a.

INITIALIZATION.
  t1 = 'Gruppe B'.
  t2 = 'Gruppe C'.
```

Neben den üblichen Eingabeelementen kann ein Selektionsbildschirm auch Drucktasten besitzen. Über spezielle Zeitpunktanweisungen kann ermittelt werden, welche Drucktaste im Selektionsschirm gedrückt wurde. Anhand der Drucktasten können dann innerhalb des Reports alternative Programmzweige ausgeführt werden. Beispielsweise erlauben viele Reparaturprogramme einen Testlauf, in dem auszuführende Änderungen zunächst nur angezeigt, nicht aber ausgeführt werden. Der Selektionsschirm stellt dazu die beiden Drucktasten TESTEN und ÄNDERN bereit.

Drucktasten können durch zwei unterschiedliche Methoden erzeugt werden. In der eigentlichen Drucktastenleiste lassen sich mit der Anweisung

```
SELECTION-SCREEN FUNCTION KEY n.
```

bis zu zwei zusätzliche Drucktasten einfügen. Dabei steht *n* für die Ziffern 1 oder 2. Bei Betätigung erzeugen diese Tasten den Funktionscode FC01 bzw. FC02. Wie der Funktionscode ausgewertet wird, demonstriert der nächste Abschnitt. Im Zusammenhang mit den zusätzlichen Funktionscodes und deren Auswertung muss im Report die Deklaration einer Dictionary-Struktur erfolgen. Dies ist einer der Fälle, in denen die TABLES-Anweisung noch notwendig ist.

```
TABLES SSCRFIELDS.
```

Diese Struktur wird zur Kommunikation des Selektionsbildschirms mit dem umgebenden Programm benutzt. Im Falle der Drucktasten wird im Initialisierungsteil (Zeitpunkt INITIALIZATION) den beiden Feldern SSCRFIELDS-FUNCTXT_01 bzw. SSCRFIELDS-FUNCTXT_02 der Text zugewiesen, den die beiden Drucktasten erhalten sollen.

Mit der zweiten Methode können beliebig viele Drucktasten innerhalb des Selektionsbildschirms erzeugt werden. Dies geschieht mit dem Kommando

```
SELECTION-SCREEN PUSHBUTTON format name
  USER-COMMAND function_code.
```

Das Format entspricht dem bereits beim Zusatz COMMENT erläuterten Format. Damit werden die Position und die Breite der Drucktaste festgelegt. Über den Parameter *name* legen Sie den Text der Drucktaste fest. Auch dafür gilt das bereits bei COMMENT erläuterte Verfahren. Es wird also entweder ein nummeriertes Textelement oder ein zum Initialisationszeitpunkt belegtes Datenelement benutzt. Der durch die Drucktaste auszulösende Funktionscode ist ein vierstelliger alphanummerischer Bezeichner. Er wird als Parameter *function_code* an die Anweisung übergeben. Er ist ohne Apostroph zu notieren.

Das folgende Beispiel zeigt, wie beide Arten von Drucktasten erzeugt werden. Der zwangsweise erforderliche Parameter dient hier zur Anzeige des durch die Drucktaste ausgelösten Funktionscodes. Gleichzeitig leitet das Beispiel zur Problematik der Zeitpunktanweisungen für Selektionsbildschirme über, von denen zwei bereits ohne nähere Erläuterung benutzt werden.

```
REPORT yz434090.

TABLES sscrfields.

PARAMETERS p1 LIKE sscrfields-ucomm.

SELECTION-SCREEN SKIP 3.
SELECTION-SCREEN FUNCTION KEY 1.
SELECTION-SCREEN FUNCTION KEY 2.
SELECTION-SCREEN PUSHBUTTON  5(15) pb1 USER-COMMAND 0001.
SELECTION-SCREEN PUSHBUTTON 25(15) pb2 USER-COMMAND 0002.
SELECTION-SCREEN PUSHBUTTON 45(15) pb3 USER-COMMAND 0003.
```

```
INITIALIZATION.
  sscrfields-functxt_01 = 'Button 1'.
  sscrfields-functxt_02 = 'Button 2'.
  pb1 = 'Button 3'.
  pb2 = 'Button 4'.
  pb3 = 'Button 5'.

AT SELECTION-SCREEN.
  p1 = sscrfields-ucomm.
```

Zeitpunkte für Selektionsbildschirme

Die bisherigen Selektionsbildschirme wurden durch die Steuerlogik des Systems abgearbeitet. Die Steuerlogik sorgt beispielsweise dafür, dass die in Varianten abgespeicherten Vorgabewerte in die Parameter und Selektionen übertragen werden, wenn der Anwender einen Report über eine solche Variante aufruft. In diesen Prozess kann der Programmierer eingreifen. Während der Erzeugung und Bearbeitung des Selektionsbildschirms prozessiert das System drei Ereignisse, die mit Hilfe entsprechender Zeitpunktanweisungen die Ausführung individueller Anweisungen ermöglichen. Diese Zeitpunkte sind `INITIALIZATION`, `AT SELECTION-SCREEN OUTPUT` und `AT SELECTION-SCREEN`.

Der Zeitpunkt `INITIALIZATION` ist der erste Zeitpunkt, der beim Start eines Reports überhaupt prozessiert wird. Dies erfolgt, nachdem der Selektionsbildschirm erstellt und mit Vorgabewerten der `PARAMETERS`-Anweisungen gefüllt wurde. Diese können hier also überschrieben werden. Die Übertragung von Werten aus Varianten erfolgt erst nach `INITIALIZATION`, würde also Vorgaben überschreiben. Der Zeitpunkt `INITIALIZATION` wird bei jedem Programmaufruf nur einmal ausgelöst.

Unmittelbar vor Anzeige des Selektionsbildschirms wird der Zeitpunkt `AT SELECTION-SCREEN OUTPUT` ausgelöst. Zwischen diesem Zeitpunkt und der Bearbeitung des Selektionsbildschirms durch den Anwender finden keine weiteren Modifikationen durch die Steuerlogik statt. An dieser Stelle können daher nochmals Werte überschrieben oder andere Modifikationen des Selektionsbildschirms wie Ausblenden von Feldern etc. ausgeführt werden. Der Zeitpunkt `AT SELECTION-SCREEN OUTPUT` ist untrennbar mit der Abarbeitung des Selektionsbilds verbunden. Er wird immer prozessiert, bevor das Selektionsbild zur Bearbeitung bereitsteht. Es ist nicht möglich, das Selektionsbild anzuzeigen, ohne diesen Zeitpunkt auszulösen.

Nachdem der Anwender die Bearbeitung des Selektionsbildschirms beendet hat, arbeitet die Steuerlogik den Zeitpunkt `AT SELECTION-SCREEN` ab. Innerhalb der zugehörigen Anweisungen können beispielsweise Eingabewerte auf Plausibilität getestet werden. Wird innerhalb dieses Zeitpunkts mit der Anweisung `MESSAGE` eine Fehlermeldung ausgelöst, wird der Selektionsbildschirm erneut abgearbeitet, die Selektionen und Parameter sind wieder eingabebereit. Vorher wird natürlich der Zeitpunkt `AT SELECTION-SCREEN OUTPUT` prozessiert.

Die Anweisung AT SELECTION-SCREEN verfügt über eine Reihe von Varianten, über die eine Bezugnahme auf ausgewählte Eingabefelder des Selektionsbildschirmes oder die erwähnten Gruppen möglich ist. Auch Eingabehilfen für Felder oder das Anzeigen von Hilfsinformationen können durch Varianten dieser Anweisung realisiert werden. Die Programmierung dieser kontextsensitiven oder Eingabehilfen erfordert allerdings einige Kenntnisse der Dialogprogrammierung, die erst im nächsten Abschnitt vermittelt werden. In all diesen Fällen wird nach Ausführung der Anweisungen erneut mit der Abarbeitung des Selektionsbildes begonnen. Das folgende Beispiel (YZ434100) baut auf dem Programm zur Definition von Blöcken (YZ434080) auf. Das Programm wird einfach durch folgende Anweisungen ergänzt:

```
AT SELECTION-SCREEN ON p1.
  IF p1 > 'H'.
    MESSAGE ID 'YZ4' TYPE 'E' NUMBER '000'.
  ENDIF.

AT SELECTION-SCREEN ON BLOCK c.
  IF r2 <> 'X'.
    MESSAGE ID 'YZ4' TYPE 'E' NUMBER '000'.
  ENDIF.

START-OF-SELECTION.
  WRITE: / 'Parameter    :', p1.
  WRITE: / 'Radiobutton 1:', r1.
  WRITE: / 'Radiobutton 2:', r2.
```

Die Anweisung MESSAGE löst eine so genannte Nachricht aus. Das Nachrichtenkonzept wird im Kapitel über die Dialogprogrammierung näher erläutert. An dieser Stelle sei nur gesagt, dass die Anweisung MESSAGE die Abarbeitung von Programmen oder Dialogen abbrechen kann. Außerdem stellt sie einen Text in die Statuszeile. Im Beispiel wird eine beliebige Nachricht benutzt, um die Verarbeitung des Selektionsbildschirms abzubrechen. Alle Informationen in der Statuszeile stehen in keinem Zusammenhang mit dem Beispiel, sondern sind rein zufällig. Mit hoher Wahrscheinlichkeit wird lediglich die Ausschrift *E:EYZ000* erscheinen.

Die beiden Prüfungen im Beispiel dienen nur zur Demonstration des Prinzips, sie erfüllen keinen praktischen Zweck. Die erste Zeitpunktanweisung prüft, ob in den Parameter P1 ein Wert eingetragen wurde, der größer als der Buchstabe „H" ist. In diesem Fall wird die MESSAGE-Anweisung ausgeführt, der Selektionsbildschirm wird erneut abgearbeitet. Allerdings ist nur der Parameter eingabebereit, die beiden Auswahlknöpfe sind abgedunkelt, sie können nicht selektiert werden. Abgesehen von der erstmaligen Bearbeitung des Selektionsbildes kann nur dann einer der beiden Auswahlknöpfe selektiert werden, wenn im Parameter P1 ein Zeichen eingetragen wurde, das kleiner als „H" ist. Die zweite Zeitpunktanweisung betrifft den Block C, also den Block mit den beiden Auswahl-

knöpfen. Dort muss zwingend der zweite Knopf markiert werden, ansonsten wird das Programm nicht fortgesetzt. Im Selektionsbildschirm sind, bedingt durch die Verbindung der Zeitpunktanweisung mit dem Block, nur die beiden Auswahlknöpfe eingabebereit, der Parameter ist inaktiv.

Funktionscodes im Selektionsbildschirm

Im Selektionsbildschirm gibt es nur wenige Gründe, einen eigenen Funktionscode auszulösen. Ein Grund könnte sein, per Knopfdruck spezielle Werte für die Parameter zu setzen oder den Selektionsbildschirm zu modifizieren (dynamische Anzeige zusätzlicher Eingabefelder o.Ä.). Ein anderer Grund wäre das Setzen von Optionen zur Gestaltung der Liste oder zur Ausführung des Programms. Viele systemnahe Reports dienen nicht in erster Linie zum Anzeigen von Informationen, sondern zum Ausführen von datenbezogenen Aktionen, z.B. dem Reparieren von Datenschiefständen oder dem Prüfen auf Konsistenz von Datenbeständen. Viele dieser Reports laufen in zwei verschiedenen Modi: einem Prüfmodus, der Fehler nur anzeigt, und einem Reparaturmodus, der Fehler auch behebt. Für solche Reports könnte man zwei Drucktasten PRÜFEN und REPARIEREN definieren und je nach Funktionscode ein Flag setzen, das im Report ausgewertet wird. Es zeugt allerdings nicht von einem guten Programmierstil, vom Selektionsbild aus per individuellem Funktionscode in völlig unterschiedliche Listen zu verzweigen.

Es wurde bereits demonstriert, wie in einem Selektionsbildschirm über zusätzliche Drucktasten eigene Funktionscodes erzeugt und ausgewertet werden können. Eine allgemeinere, aber auch aufwändigere Möglichkeit besteht im Setzen eines eigenen Status für den Selektionsbildschirm. Dies kann zum Zeitpunkt `INITIALIZATION` oder `AT SELECTION-SCREEN OUTPUT` geschehen. Ein individueller Status sollte immer vom Standardstatus für Listen ausgehen, damit alle wichtigen Funktionscodes erhalten bleiben. Sie können einen solchen Status am einfachsten erzeugen, indem Sie im Menu Painter eine entsprechende Vorlage einfügen. Vergleichen Sie dazu Abschnitt 3.3.3 zur Pflege der Oberfläche.

Alle individuellen Funktionscodes müssen natürlich auch selbst verarbeitet werden. Zum Zeitpunkt `AT SELECTION-SCREEN` ist der aktuelle Funktionscode im Feld `SSCRFIELDS-UCOMM` enthalten, unabhängig davon, ob der Funktionscode über ein Menü oder Drucktasten des Selektionsbildes erzeugt wurde.

Das System behandelt alle Funktionscodes, die vom Standardstatus des Selektionsbildschirms erzeugt werden. Nach Bearbeitung des jeweiligen Funktionscodes ist wieder das Selektionsbild aktiv, sofern der Report nicht beendet oder zur Ausführung des Reports verzweigt wurde. Funktionscodes, die nicht vom System verarbeitet werden, führen ebenfalls zur erneuten Verarbeitung des Selektionsbildschirms. Vom programmtechnischen Standpunkt aus ist der Selektionsbildschirm der eigentliche Kern des Programms, alle anderen Funktionen, auch die Erzeugung der eigentlichen Liste, sind letztlich Unterprogramme. Die Erstellung und Ausgabe der Liste wird vom Selektionsbildschirm aus durch den

Funktionscode ONLI eingeleitet. Soll ein individuell erzeugter Funktionscode das Erstellen der Liste einleiten, gelingt dies nur mit einem kleinen Trick, den das folgende Beispiel zeigt:

```
REPORT yz434110.
TABLES sscrfields.
DATA g_ucomm LIKE sscrfields-ucomm.

* PARAMETERS P1. "nur in älteren Versionen erforderlich

SELECTION-SCREEN SKIP 3.

SELECTION-SCREEN FUNCTION KEY 1.
SELECTION-SCREEN FUNCTION KEY 2.

SELECTION-SCREEN PUSHBUTTON  5(15) pb1 USER-COMMAND 0001.
SELECTION-SCREEN PUSHBUTTON 25(15) pb2 USER-COMMAND 0002.
SELECTION-SCREEN PUSHBUTTON 45(15) pb3 USER-COMMAND 0003.

INITIALIZATION.
  sscrfields-functxt_01 = 'Taste 1'.
  sscrfields-functxt_02 = 'Taste 2'.
  pb1 = 'Taste 3'.
  pb2 = 'Taste 4'.
  pb3 = 'Taste 5'.

AT SELECTION-SCREEN.
  g_ucomm = sscrfields-ucomm.
  sscrfields-ucomm = 'ONLI'.

START-OF-SELECTION.
  WRITE: / 'Ausgelöst wurde Taste'.
  CASE g_ucomm.
    WHEN 'FC01'.
      WRITE '1'.
    WHEN 'FC02'.
      WRITE '2'.
    WHEN '0001'.
      WRITE '3'.
    WHEN '0002'.
      WRITE '4'.
    WHEN '0003'.
      WRITE '5'.
  ENDCASE.
```

Der konkret ausgelöste Funktionscode wird in einem programminternen Datenfeld zwischengespeichert und im eigentlichen Report ausgewertet. Die List-Erstellung wird eingeleitet, indem SSCRFIELDS-UCOMM mit dem dazu erforderlichen Funktionscode ONLI belegt wird. Der Parameter P1 ist nur in älteren Versionen der R/3-Software erforderlich, da in diesen ein Selektionsbild nur dann angezeigt wird, wenn es mindestens einen Parameter oder eine Selektion enthält.

Allgemeine Verwendung von Selektionsbildschirmen

Ab Version 4.0 können Selektionsbilder unabhängig vom Programmtyp genutzt werden. Sie können Selektionsbilder mit dem Kommando

```
CALL SELECTION-SCREEN screen.
```

aufrufen. Dabei ist screen ein nummerischer Wert.

Innerhalb einer Anwendung müssen Sie ein derartiges Selektionsbild mit den Anweisungen

```
SELECTION-SCREEN BEGIN OF SCREEN screen.
```

und

```
SELECTION-SCREEN END OF SCREEN screen.
```

deklarieren. Zwischen diesen beiden Anweisungen stehen die üblichen Anweisungen zur Gestaltung von Selektionsbildern. Das nachfolgende Beispiel verfügt über zwei dieser Selektionsbilder sowie einen standardmäßigen Selektionsbildschirm. Nach Betätigen einer der beiden Drucktasten wird eines der beiden zusätzlichen Selektionsbilder aufgerufen. Der Report listet zur Kontrolle die Inhalte der beiden Parameter auf.

```
REPORT yz434120.
TABLES sscrfields.
SELECTION-SCREEN PUSHBUTTON  5(15) pb1 USER-COMMAND 0001.
SELECTION-SCREEN PUSHBUTTON 25(15) pb2 USER-COMMAND 0002.

SELECTION-SCREEN BEGIN OF SCREEN 1.
PARAMETERS a(20).
SELECTION-SCREEN END OF SCREEN 1.

SELECTION-SCREEN BEGIN OF SCREEN 2.
PARAMETERS b(20).
SELECTION-SCREEN END OF SCREEN 2.

INITIALIZATION.
  pb1 = 'Screen 1'.
  pb2 = 'Screen 2'.
```

```
AT SELECTION-SCREEN.
  CASE sscrfields-ucomm.
    WHEN '0001'.
      CLEAR: a, b.
      CALL SELECTION-SCREEN 1.
      sscrfields-ucomm = 'ONLI'.
    WHEN '0002'.
      CLEAR: a, b.
      CALL SELECTION-SCREEN 2.
      sscrfields-ucomm = 'ONLI'.
  ENDCASE.
START-OF-SELECTION.
  WRITE: / 'Parameter A:', a.
  WRITE: / 'Parameter B:', b.
```

Diese allgemein verwendbaren Selektionsbilder verfügen ebenfalls über Varianten. Außerdem fügen sie sich nahtlos in das Programmierkonzept ein. So können Sie diese neuen Selektionsbilder nicht nur innerhalb einer Anwendung aufrufen, sondern auch im Zusammenhang mit dem Kommando `SUBMIT` oder der Gestaltung einer Report-Transaktion benutzen. Dazu wurde der Funktionsumfang einiger bereits existierender Kommandos erweitert.

Um beim Aufruf eines Selektionsbildschirms eine Variante als Vorgabewert zu setzen, verwenden Sie das Kommando

```
CALL SELECTION-SCREEN screen USING SELECTION-SET variant.
```

Beim Aufruf eines Reports mit `SUBMIT` können Sie das Selektionsbild mit der folgenden Anweisung auswählen:

```
SUBMIT report USING SELECTION-SCREEN screen.
```

Tab Strips in Selektionsbildern

Ab Version 4.0 halten so genannte Tab Strips Einzug in Dynpros. Ein Tab Strip besteht aus einem Teilbereich einer Eingabemaske und einigen über diesem Bereich befindlichen Drucktasten. Nach Betätigung einer dieser Tasten wird im Tab-Strip-Bereich eine zu dieser Taste gehörende Eingabemaske eingeblendet. Bei dieser Eingabemaske handelt es sich um ein Dynpro, das als so genannter Subscreen definiert sein muss.

Im Windows-Sprachgebrauch werden die einzelnen Eingabemasken auch als *Registerkarten* und die Drucktasten als *Reiter* bezeichnet. Die entsprechenden Elemente sind nicht nur in Dynpros, sondern auch in Selektionsbildern verfügbar. Bild 3.59 zeigt das Selektionsbild eines Beispielprogramms mit einem Tab Strip.

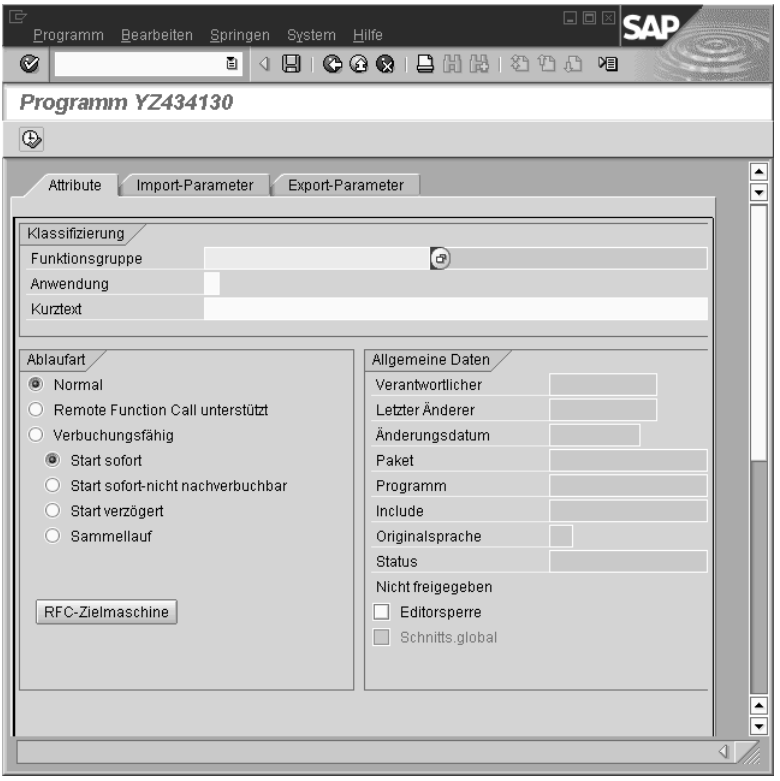

Abbildung 3.59 **© SAP AG**
Selektionsbild mit Tab Strip

Um ein Selektionsbild mit Tab Strips zu versehen, müssen zwei verschiedene Kommandos benutzt werden. Zunächst wird mit

```
SELECTION-SCREEN BEGIN OF TABBED BLOCK block FOR lin LINES.
```

ein `lin` Zeilen langer Bereich für den Tab Strip reserviert. Gleichzeitig legt das System eine Feldleiste an, deren Namen dem Namen des Blocks entspricht. Diese Feldleiste verfügt über die drei Felder `PROG`, `DYNNR` und `ACTIVETAB`. Anschließend definieren Sie mit

```
SELECTION-SCREEN TAB (length) tabstrip_card
  USER-COMMAND fcode
  [ DEFAULT [ PROGRAM program ] SCREEN screen ] .
```

die einzelnen Reiter oder Register. Dabei ist es erforderlich, jedem Register einen beschreibenden Text zuzuordnen. Der Parameter `tabstrip_card` stellt eigentlich den Namen einer der Registerkarten dar. Er kann im Programm wie ein Datenfeld benutzt werden, das zum Zeitpunkt `INITILIZATION` mit der Beschrif-

tung der Reiters der Registerkarte gefüllt wird. Das Verfahren ähnelt dem der Textzuweisung zu Funktions- oder Drucktasten. Damit die einzelnen Reiter in einer definierten Breite erscheinen (z.B. alle gleich breit), geben Sie mit dem Parameter `length` die Breite des Reiters vor. Diese Angabe ist auf jeden Fall erforderlich. Der Wert ist in runde Klammern einzuschließen. Mit dem Parameter `fcode` legt fest, welcher Funktionscode beim Betätigen des Reiters ausgesendet werden soll. Abgeschlossen wird ein Tab-Strip-Block wie ein normaler Block mir der Anweisung

```
SELECTION-SCREEN END OF BLOCK block.
```

Um nach Betätigen eines Reiters die passende Eingabemaske im Tab-Strip-Bereich darstellen zu können, existieren zwei Varianten. Sie können zunächst im oben genannten Kommando durch zusätzliche Parameter den Namen des Programms und des Dynpros angeben. Sie können aber auch zum Zeitpunkt `AT SELECTION-SCREEN` den ausgelösten Funktionscode auswerten und die bereits erwähnten Felder des Tab-Strip-Blocks zur Laufzeit belegen.

Nachfolgend ein kurzes Beispiel. Damit für dieses Beispiel nicht extra eigene Selektionsbilder angelegt werden müssen, finden existierende Dynpros des R/3-Systems Verwendung. Es handelt sich dabei um einige Subscreens, die zur Bearbeitung von Funktionsbausteinen eingesetzt werden. Diese Subscreens führen keinerlei Verarbeitung der eingetragenen Werte aus. Sie müssen also keinerlei Nebenwirkungen befürchten. Hier zunächst das Listing:

```
 1 REPORT yz434130.
 2 SELECTION-SCREEN BEGIN OF TABBED BLOCK tstrip FOR 40 LINES.
 3 SELECTION-SCREEN TAB (79) card_1 USER-COMMAND comm1
 4                      DEFAULT PROGRAM sapms38l SCREEN 3030.
 5 SELECTION-SCREEN TAB (79) card_2 USER-COMMAND comm2
 6                      DEFAULT PROGRAM sapms38l SCREEN 3050.
 7 SELECTION-SCREEN TAB (79) card_3 USER-COMMAND comm3.
 8 SELECTION-SCREEN END OF BLOCK tstrip.
 9
10 INITIALIZATION.
11   card_1 = 'Attribute'.
12   card_2 = 'Import-Parameter'.
13   card_3 = 'Export-Parameter'.
14
15 AT SELECTION-SCREEN.
16   IF sy-ucomm = 'COMM3'.
17     tstrip-prog      = 'SAPMS38L'.
18     tstrip-dynnr     = 3052.
19     tstrip-activetab = 'CARD_3'.
20   ENDIF.
```

Der Selektionsbildschirm enthält einen Tab-Strip-Bereich mit drei verschiedenen Registern. Dazu wird durch die beiden Anweisungen in Zeile 2 und 8 zunächst ein Block definiert. Die Anweisungen in den Zeilen 3 und 5 definieren je-

weils eine Eingabeseite. Dabei werden die einzufügenden Subscreens statisch in der Anweisung definiert. In Zeile 7 wird auf diese Angabe verzichtet. Zum Zeitpunkt AT SELECTION-SCREEN muss daher der Tab Strip dynamisch mit den notwendigen Angaben versorgt werden. Dabei ist natürlich der durch den Reiter ausgelöste Funktionscode auszuwerten. Die durch den Tab Strip ausgelösten Funktionscodes werden im Feld SY-UCOMM abgelegt. Durch die Definition des Tab Strips mit der SELECTION-SCREEN-Anweisung entsteht ein gleichnamiges komplexes Datenobjekt mit den Teilfeldern PROG, DYNNR und ACTIVETAB. Die ersten beiden Felder werden mit dem Programmnamen, aus dem das einzufügende Dynpro stammen soll, und der Dynpronummer gefüllt. In ACTIVETAB müssen Sie den Namen des Tab-Strip-Reiters eintragen, der markiert werden soll. Die Zeilen 11 – 13 enthalten Anweisungen, die einmalig bei der Initialisierung des Selektionsbildschirmes ausgeführt werden. Mit diesen Zuweisungen legen Sie den Text der einzelnen Tab-Strip-Reiter fest. Das sollte in praxiswirksamen Anwendungen aber nicht, wie hier gezeigt, durch hart kodierte Texte, sondern durch leicht zu übersetzende Textsymbole erfolgen.

Übungen

❶ Ändern Sie das Programm YZ434050 bzw. YZ434060 so ab, dass die beiden Prüfungen entweder beide in der SELECT-Anweisung oder beide mit dem Kommando CHECK realisiert werden. Achten Sie auf das Laufzeitverhalten.

❷ Ändern Sie in den Programmen mit Feldgruppenbildung die Reihenfolge der Anweisungen zur Gruppenbildung und analysieren Sie das Ergebnis.

❸ Erfassen Sie eigene Texte für Parameter und Selektionen.

❹ Erstellen Sie zunächst das Programm YZ434100 durch Modifikation von YZ4342080 und führen Sie die im Text beschrieben Änderungen (parameterbezogene AT SELECTION-SCREEN ON-Anweisungen) durch. Testen Sie das Programm. Modifizieren Sie das Programm erneut, indem Sie die parameterbezogenen AT SELECTION-SCREEN ON-Anweisungen durch eine gemeinsame Prüfung zum allgemeinen AT SELECTION-SCREEN-Zeitpunkt ersetzen. Analysieren Sie das Ergebnis.

3.4.4 Ausgabeaufbereitung

In den bisherigen Beispielen wurden bereits einige Daten auf dem Bildschirm ausgegeben und in bescheidenem Maße formatiert. In vollwertigen Reports und deren Ausgabelisten sind weitergehende Varianten der Ausgabeaufbereitung sowohl der einzelnen Zeilen als auch der Seiten möglich. Diese Anweisungen erleichtern einerseits die Programmierung, andererseits kann mit dem Einsatz von farbigen Gestaltungselementen die Übersichtlichkeit der Bildschirmausgabe erhöht werden. Für die Gestaltung der Ausgabezeilen sind die Kommandos WRITE und FORMAT zuständig. Neben der Gestaltung einzelner Zeilen setzen

weitere Kommandos globale Vorgaben wie z.B. Seitenlänge oder -breite sowie diverse Überschriften. All diese Kommandos sind für die reine Funktionalität eines Reports weniger wichtig. Allerdings sind sie unentbehrlich, wenn es darum geht, einer Ausgabeliste ein professionelles Aussehen zu verleihen. Für den Test der nachfolgenden Anweisungen legen Sie bitte ein neues Programm an.

Parameter für Seitengestaltung

Für die Abmessungen einer Liste (Länge und Breite einer Seite) werden vom System zunächst Standardwerte eingesetzt. Die Breite der Liste richtet sich nach der aktuellen Fensterbreite, die Seitenlänge ist unbegrenzt. Die Liste besteht also nur aus einer Seite. Allerdings ist es auch möglich, individuelle Vorgaben zu definieren. Dies erfolgt in der Regel mit zwei Optionen der REPORT-Anweisung.

```
... LINE-SIZE columns ...
... LINE-COUNT lines(footer) ...
```

Die LINE-COUNT-Option kann mit einer optionalen Angabe für freizuhaltende Fußzeilen versehen werden. Fehlt diese Angabe, wird als Standardvorgabe 0 wirksam. Die Zahlenangaben sind als Direktwert, also ohne umschließende Apostrophe, einzutragen. Unabhängig von der eingestellten Seitenlänge werden bei Ausgabe der Liste auf dem Bildschirm oder einem Drucker am oberen Rand der Seite stets der Kopf und die Überschrift der Liste angezeigt bzw. ausgedruckt. Auch bei einer Liste, in der keine individuelle Seitenlänge gesetzt wird, erscheint daher immer ein korrekter Seitenkopf. Die Angabe der Seitenlänge hat aus diesem Grund vor allem Auswirkung auf die korrekte Nummerierung der Seiten. Bei einer unendlichen Seitenlänge werden alle ausgedruckten Blätter mit der gleichen Seitennummer versehen. Dieser Nachteil kann durch manuell ausgelöste Seitenvorschübe umgangen werden. Dies erfolgt mit dem Kommando

```
NEW-PAGE.
```

Einige Optionen zu diesem Kommando erlauben es, für die neue Seite die Länge und Breite zu ändern sowie standardmäßig generierte Titel und Überschriften an- oder abzuschalten. Innerhalb eines Reports können daher die Gestaltungen einzelner Seiten voneinander abweichen.

Individueller Seitenkopf

Neben der Verwendung von Standardelementen kann in einem Report über die Zeitpunktanweisung TOP-OF-PAGE ein individueller Seitenkopf erzeugt werden. In diesem Seitenkopf müssen sowohl Kopfzeile als auch Spaltenüberschriften ausgegeben werden. Wird mittels dieses Zeitpunktes ein eigener Kopf erzeugt, muss natürlich die Ausgabe des Standardseitenkopfes abgeschaltet werden. Dies könnte entweder durch die Anweisung

```
NEW-PAGE NO-TITLE NO-HEADING.
```

oder aber einfacher mit

```
REPORT name NO STANDARD PAGE HEADING.
```

erfolgen. Innerhalb eines Reports können die Vorgaben in der REPORT-Anweisung problemlos durch NEW-PAGE mit entsprechenden Optionen wieder geändert werden. Das Kommando NEW-PAGE führt nur dann einen Seitenvorschub aus, wenn vorher echte Ausgaben auf der Seite erfolgten. Das Erzeugen leerer Seiten ist daher mit NEW-PAGE nicht möglich. Allerdings werden die Optionen (z.B. NO-TITLE) auf jeden Fall wirksam.

Heftrand und Fixierung

In Listen, die auf einem Drucker ausgegeben werden sollen, ist oft ein Heftrand erforderlich. Je nach verwendetem Drucker und dessen konkreter Einstellung erzeugt er den Rand nicht unbedingt selbst. Die Anweisung

```
SET MARGIN column row.
```

setzt daher den Druckbeginn auf die angegebene Position. Die Angabe der Zeile ist optional. Beide Werte können als Direktwert oder Feld angegeben werden.

Listen können bis zu 255 Zeichen breit werden. Um eine solche Liste am Bildschirm betrachten zu können, muss diese durch den Rollbalken des Windows-Fensters horizontal gerollt werden. Um dem Anwender die Orientierung zu erleichtern, ist es sinnvoll, am linken Rand bestimmte Hilfswerte, z.B. eine Zeilen- oder Ordnungsnummer oder einen Schlüsselbegriff, stehen zu lassen. Zur Fixierung von Spalten dient das Kommando

```
SET LEFT SCROLL-BOUNDARY.
```

Es fixiert die Spalten bis zur aktuellen Schreibposition. Diese werden beim horizontalen Rollen des Bildes nicht mitbewegt, bleiben also am linken Bildrand erhalten. Mit einem Zusatz kann die Zahl der zu fixierenden Spalten auch unabhängig von der Schreibposition festgelegt werden:

```
SET LEFT SCROLL-BOUNDARY COLUMN column.
```

Die Fixierung ist auch für einzelne Zeilen, z.B. die Zeilen des Tabellenkopfs, möglich. Dazu wird das Kommando

```
NEW-LINE NO SCROLLING.
```

eingesetzt. Es sorgt dafür, dass die nächste ausgegebene Zeile nicht horizontal gerollt wird. Die Standardlistenüberschrift wird automatisch mit dieser Option ausgegeben.

Positionierung

Bereits in den ersten Demoprogrammen wurde eine Möglichkeit zur gezielten Positionierung von Bildschirmausgaben vorgestellt. Alternativ zur Benutzung von WRITE in Verbindung mit Formatangaben existieren noch einige andere Kommandos. Vor deren Beschreibung soll zunächst noch das Kommando WRITE genauer erläutert werden.

Jedes auszugebende Feld kann eine Formatangabe erhalten, die aus maximal drei Teilen besteht, einem Zeichen für Zeilenvorschub, einer Ausgabeposition und einer Ausgabelänge. Diese Angaben werden in der Form

```
/position(length)
```

notiert und stehen vor dem auszugebenden Wert. Der Schrägstrich ist das Symbol für den Zeilenvorschub. Alle drei Elemente der Formatangabe sind optional. Die Werte für Position und Länge können sowohl als Direktwerte als auch als Felder übergeben werden. Allerdings wird im letzteren Fall der Vorsatz AT vor dem gesamten Konstrukt erforderlich. Folgender Ausschnitt aus einem Programm zeigt einige korrekte Anweisungen:

```
...
DATA: P TYPE I.

WRITE: 'ABC'.
WRITE: / 'DEF'.
WRITE: /4 'GHI'.
WRITE: 8(2) 'JKL'.

P = 8.
WRITE: AT /P 'MNO'.
...
```

Fehlt eine Ausgabelänge, so wird die Standardausgabelänge für den jeweiligen Datentyp benutzt. Die Wirkung des Zeilenvorschubzeichens bleibt aus, wenn das auszugebende Feld leer ist, wie das folgende Beispiel demonstriert.

```
...
DATA: C.

WRITE: / 'ANFANG'.
WRITE: / C.
WRITE: / C.
WRITE: / 'ENDE'.
```

Diese Eigenart tritt aber nur auf, wenn das Zeilenvorschubzeichen in einer Formatangabe für ein Feld steht. Beim Einsatz als eigenständiges Zeichen, also ohne Feldangabe in einer eigenen WRITE-Anweisung oder zumindest als eigen-

ständiges Element in einer solchen Anweisung, wird immer ein Zeilenvorschub erzeugt:

```
DATA: C.

WRITE: / 'ANFANG'.
WRITE: /.
WRITE: /, C.
WRITE: / 'ENDE'.
```

Allerdings kann mit der Anweisung

```
SET BLANK LINES ON.
```

die Ausgabe von Leerzeilen erzwungen werden, auch wenn das auszugebende Feld leer ist. Diese Anweisung bleibt wirksam, bis sie mit

```
SET BLANK LINES OFF.
```

wieder abgeschaltet wurde.

Eine sichere Methode zum Erzeugen von Leerzeilen ist das Kommando

```
SKIP.
```

Es fügt in der eben dargestellten Form eine Leerzeile in den Text ein. Mit einem zusätzlichen Parameter kann die Zahl der einzufügenden Leerzeilen vorgegeben werden:

```
SKIP lines.
```

Eine letzte Variante schließlich positioniert die Schreibmarke auf eine bestimmte Zeile der Seite, wobei innerhalb der aktuellen Seitenlänge vor- und rückwärts gesprungen werden kann. Die Zählung der Zeilen beginnt mit 1.

```
SKIP TO LINE line_number.
```

Innerhalb einer Zeile kann mit

```
POSITION column.
```

die Position für eine nachfolgende Ausgabe mit `WRITE` auf die angegebene Spalte gesetzt werden. In Listen sollen Werte normalerweise exakt nach den Spaltenüberschriften oder in mehrzeiligen Listen nach einem anderen Datenfeld ausgerichtet werden. Änderungen in der Überschrift erfordern so die Überarbeitung aller anderen Positionsangaben. Einfacher ist die Verwendung einer weiteren Option des `WRITE`-Kommandos. Mittels

```
WRITE value UNDER field.
```

wird ein auszugebender Wert exakt in der Spalte positioniert, in der das Feld ausgegeben wurde. Dabei ist der Feldname exakt so zu notieren wie bei dessen

eigener Ausgabe. Dies ist vor allem dann von Bedeutung, wenn auf Textelemente mit Defaultwert Bezug genommen wird. Das nachfolgende Programm, eine Modifikation von YZ432050, zeigt einige Varianten der UNDER-Option.

```
REPORT yz434140 NO STANDARD PAGE HEADING.
TABLES tadir.

PARAMETERS p_author LIKE tadir-author DEFAULT '*'.
SELECT-OPTIONS s_name FOR tadir-obj_name.

WRITE: 'Objecttype', 20 'Name'(001), 40 text-002.
* create text-002 with text 'Author'!

SELECT * FROM tadir
WHERE obj_name IN s_name.
  CHECK tadir-author CP p_author.



  WRITE: / tadir-pgmid    UNDER 'Objecttype',
           tadir-obj_name UNDER 'Name'(001),
           tadir-author   UNDER text-002.

ENDSELECT.
```

Der Zusatz UNDER bewirkt nur die Positionierung innerhalb der Zeile und löst keinen Zeilenvorschub aus. Sollte der Zeilenvorschub fehlen, werden gegebenenfalls die Texte, auf die Bezug genommen wird oder die vorangegangenen Ausgaben durch die neuen Werte überschrieben, da die Ausgabe in derselben Zeile erfolgt.

Indirekt zu den Positionierungsmöglichkeiten gehören auch die Kommandos

```
BACK.
```

und

```
RESERVE n LINES.
```

Die letzte Anweisung führt einen Seitenvorschub aus, wenn auf der aktuellen Seite nicht mindestens n Zeilen frei sind. Diese Anweisung steht üblicherweise vor zusammenhängenden Ausgaben, die mehrere Zeilen umfassen oder vor der Ausgabe einer Zwischenüberschrift. Sie stellt dann indirekt sicher, dass auf der aktuellen Seite nach einer Überschrift noch einige Datenzeilen folgen. Die Anweisung BACK setzt die Ausgabeposition entweder auf die erste Zeile einer Seite oder, nach Verwendung der RESERVE-Anweisung, auf die erste nach RESERVE ausgegebene Zeile.

Formatierung

Das Kommando WRITE übernimmt nicht nur die Ausgabe und Positionierung der Werte, sondern auch eine Formatierung. Dazu stehen einige bisher noch nicht besprochene Optionen zur Verfügung. Diese werden hinter dem auszugebenden Wert notiert. Die WRITE-Anweisung hat damit die Form

```
WRITE format value option.
```

Der größte Teil der Aufbereitungsoptionen (siehe Tabelle 3.33) kann sehr schnell erläutert werden, ausführlichere Beschreibungen werden nur für wenige Optionen erforderlich.

Option	Wirkung
NO-ZERO	Führende Nullen vor Zahlen werden durch Leerzeichen ersetzt, ist der Inhalt gleich 0, werden nur Leerzeichen ausgegeben.
NO-SIGN	Es wird kein Vorzeichen ausgegeben.
NO-GAP	Nach dem Wert wird kein trennendes Leerzeichen ausgegeben.
NO-GROUPING	Tausender-Trennzeichen bei Typ P und I unterdrücken.
DD/MM/YYYY MM/DD/YYYY DD/MM/YY MM/DD/YY DDMMYY MMDDYY YYMMDD	Datumsfelder werden gemäß dem Muster aufbereitet und ausgegeben.
CURRENCY *curr*	Das Feld wird gemäß den Angaben zur Währung aufbereitet. Diese Angaben werden im Customizing eingestellt. Maßgeblich ist vor allem die Tabelle TCURX.
DECIMALS *n*	Ausgabe mit n Dezimalstellen.
ROUND *n*	Nur für Datentyp P. Das Dezimalzeichen wird um n Stellen nach links (r > 0) oder rechts (n < 0) verschoben. Dann wird der Wert ohne Nachkommastellen ausgegeben.

Tabelle 3.33
Aufbereitungsoptionen für WRITE

Option	Wirkung
UNIT	Ähnlich wie bei CURRENCY wird der Wert gemäß den Angaben in einer Einheiten-Tabelle (T006) aufbereitet.
TIME ZONE *timezone*	Aufbereitung einer Zeitangabe in UTC in die der angegebenen Zeitzone.
EXPONENT *n*	Nur für Gleitpunktzahlen. Die Zahl wird in Exponentialdarstellung mit dem angegebenen Exponenten ausgegeben.
USING EDIT MASK *mask*	Der Wert wird gemäß einer Maske aufbereitet.
USING NO EDIT MASK	Die im Dictionary festgelegte Konvertierung wird nicht ausgeführt.
UNDER *field*	Ausgabe exakt unter einem anderen Feld.
LEFT-JUSTIFIED	Wert im Ausgabefeld linksbündig ausrichten.
CENTERED	Wert im Ausgabefeld zentrieren.
RIGHT-JUSTIFIED	Wert im Ausgabefeld rechtsbündig ausrichten.
AS SYMBOL	Interpretation des Feldinhalts als Symbolname und Ausgabe desselben.
AS ICON	Interpretation des Feldinhalts als Icon-Name und Ausgabe desselben.
AS LINE	Darstellung spezieller Zeichen als Rahmensymbol.
QUICKINFO	Angabe eines Tooltips für das auszugebende Feld.
Attribute	Steuerung der Attribute (Farbe, Intensität) für ein Feld mit denselben Anweisungen wie bei FORMAT.

Tabelle 3.33
Aufbereitungsoptionen für WRITE (Fortsetzung)

Erläuterungsbedürftig sind vor allem die beiden USING-Optionen. Hinter der ersten Variante (USING EDIT MASK) wird eine Maske vorgegeben, deren Zeichen mit denen des auszugebenden Wertes gemischt werden. In der Maske dürfen die in Tabelle 3.34 aufgeführten Zeichen enthalten sein. Die Maske selbst ist als Zeichenkette, also in Apostrophe eingeschlossen, anzugeben.

Maskenzeichen	Wirkung
_ (Unterstrich)	Steht für je ein Zeichen des auszugebenden Wertes.
V	Vorzeichen.
LL (am Anfang der Maske)	Linksbündig eintragen.
RR (am Anfang der Maske)	Rechtsbündig übertragen.
==xxxxx	Konvertierung des Wertes mit der Routine CONVERSION_EXIT_xxxxx_OUTPUT.
Alle anderen Zeichen	Trennzeichen, werden mit den Zeichen des Wertes gemischt.

Tabelle 3.34
Zeichen in Ausgabemasken

Die Wirkung der einzelnen Maskenzeichen verdeutlichen einige Beispiele. Im nachfolgenden Programm finden Sie der besseren Übersicht wegen die erzeugte Ausgabe jeweils als Kommentar unter der Ausgabeanweisung.

```
REPORT yz434150 NO STANDARD PAGE HEADING.
DATA: w1(4)        VALUE 'ABCD',
      w2(8)        VALUE '12345678',
      w3    TYPE i VALUE '-12345'.

WRITE /(20) w1 USING EDIT MASK 'LL:______________:'.
* :ABCD          :

WRITE /(20) w1 USING EDIT MASK 'RR:______________:'.
* :          ABCD:

WRITE /(20) w2 USING EDIT MASK ':__-__-__-____:'.
* :12-34-56-78  :

WRITE /(20) w2 USING EDIT MASK 'RR:__-__-__-____:'.
* :  -12-34-5678:

WRITE /(20) w3 USING EDIT MASK 'RR:V_____________:'.
* :-       12345:
```

Für alle Domänen kann im Data Dictionary der fünfstellige Name einer Konvertierungsroutine, auch Konvertierungsexit genannt, festgelegt werden. Aus diesem Bezeichner bildet das System die Namen zweier Funktionsbausteine mit Namen CONVERSION_EXIT_xxxxx_OUTPUT und CONVERSION_EXIT_xxxxx_INPUT. Beim Transport eines Wertes von einem Eingabefeld eines Dynpros in das eigentliche Datenfeld wird automatisch die Routine CONVERSION_EXIT_

xxxxx_INPUT aufgerufen, bei Ausgabe des Feldes CONVERSION_EXIT_xxxxx_OUTPUT. In diesen Routinen kann eine beliebige Transformation des Feldinhalts stattfinden. Mit den Maskenzeichen ==xxxxx wird eine beliebige Konvertierungsfunktion für den Wert aufgerufen. Das kann sinnvoll sein, wenn das auszugebende Feld nicht von einem Dictionary-Feld abgeleitet wurde, aber einen vergleichbaren Inhalt besitzt.

Falls im Dictionary bereits eine Konvertierung für die dem Feld zu Grunde liegende Domäne vorhanden ist, kann mit der Option USING NO EDIT MASK die Konvertierung auch abgeschaltet werden.

Für Währungen und Einheiten existieren im R/3-System einige Tabellen, in denen im Rahmen des Customizing einige Informationen abgelegt werden, z.B. die Zahl der Nachkommastellen, Umrechnungsfaktoren usw. Mittels der Optionen CURRENCY und UNIT werden die auszugebenden Werte entsprechend der Angaben in diesen Tabellen aufbereitet.

Bildschirmattribute

Listen können durch optische Hervorhebung wichtiger Elemente wesentlich übersichtlicher gestaltet werden. Dies kann im R/3-System durch eine unterschiedliche Helligkeit oder durch Farbe erfolgen. In den Listen können maximal acht Farben verwendet werden. Diese werden über eine Nummer oder vordefinierte Bezeichner angesprochen. Im SAP-Style Guide werden für bestimmte Elemente einer Liste bereits Farben empfohlen. An dieser Empfehlung für den Einsatzzweck orientieren sich auch die Namen der vordefinierten Bezeichner (siehe Tabelle 3.35). Es ist ratsam, sich an diese Empfehlungen zu halten. Sie erfüllen ihren Zweck nur, wenn alle Listen im System nach möglichst einheitlichen Richtlinien gestaltet werden.

Nummer	Bezeichner	Einsatz für	Farbe
OFF	COL_BACKGROUND	Hintergrund	GUI-abhängig
1	COL_HEADING	Überschriften	Graublau
2	COL_NORMAL	Listenkörper	Hellgrau
3	COL_TOTAL	Summen	Gelb
4	COL_KEY	Schlüsselspalten	Blaugrün
5	COL_POSITIVE	Positiver Schwellwert	Grün
6	COL_NEGATIVE	Negativer Schwellwert	Rot
7	COL_GROUP	Gruppenstufen	Violett

Tabelle 3.35
Farben in Listen (Zeilenhintergrund)

Die Farben sowie einige andere Attribute können mit dem Kommando

```
FORMAT attribut_1 attribut_2 ... attribut_n.
```

gesetzt werden. Sie gelten dann ab der nächsten WRITE-Anweisung. Durch ein Ereignis, also Ansprung einer Zeitpunktanweisung, werden die Attribute wieder auf ihren Standardwert zurückgesetzt, sind nach dem Austritt aus dem Ereignis aber wieder gültig. Attribute, die beispielsweise in einem TOP-OF-PAGE-Block gesetzt werden, gelten damit nur für diesen. Die Farben werden mit der Anweisung

```
FORMAT COLOR color.
```

gesetzt. Dabei ist *color* entweder eine der in Tabelle 3.35 aufgeführten Ziffern oder einer der vordefinierten Bezeichner. Die Farbnummer kann auch dynamisch übergeben werden, also als Wert eines Feldes:

```
FORMAT COLOR = field.
```

In diesem Fall ist *field* ein Datenfeld vom Typ I. Das Feld enthält eine der Farbnummern. Neben den Farben kann noch die Intensität mittels des Zusatzes INTENSIFIED bzw. INTENSIFIED OFF eingestellt werden. Der Zusatz INVERSE bzw. INVERSE OFF vertauscht Vorder- und Hintergrundfarbe. Auch hier kann die jeweilige Option (ON oder OFF) dynamisch angegeben werden. Dabei werden ein Feldinhalt von 0 als OFF, alle anderen Werte als ON interpretiert.

Die Optionen des Kommandos FORMAT können auch in der WRITE-Anweisung verwendet werden. Sie setzen dort die Attribute für die Ausgabe eines einzelnen Feldes. Das folgende Programm demonstriert einige Varianten des Kommandos.

```
REPORT yz434160 NO STANDARD PAGE HEADING.
DATA n TYPE i VALUE 0.
DO 8 TIMES.
  WRITE: / 'Color: ' NO-GAP COLOR = n ,
           n COLOR = n,
           'Color: ' NO-GAP COLOR = n INTENSIFIED OFF,
           n COLOR = n INTENSIFIED OFF,
           'Color: ' NO-GAP COLOR = n INVERSE ON,
           n COLOR = n INVERSE ON.
  ADD 1 TO n.
ENDDO.
```

Übungen

❶ Ergänzen Sie einige Programme (z.B. YZ434010) mit einigen Anweisungen zur Ausgabeformatierung, z.B. zur farbigen Hervorhebung von Zwischenüberschriften oder Summenzeilen.

❷ Setzen Sie in den bisher erzeugten Reports eine eigene Überschrift.

❸ Schreiben Sie einen Report, der ausreichend viele Ausgabezeilen mit beliebigem Inhalt erzeugt. Testen Sie die Kommandos zur expliziten Abgabe von Seitenlänge und Seitenbreite.

❹ Ergänzen Sie den Report `YZ43020` mit Spaltenüberschriften. Testen Sie die Kommandos zur Fixierung von Zeilen bzw. Spalten.

3.5 Interaktive Reports

Die im Abschnitt 3.4 besprochenen Reports ermöglichen es, einige Standardfunktionen über das Menü aufzurufen, z.B. das Suchen von Zeichenketten oder das Drucken der Liste. Echte Interaktivität bietet diese einfache Form der List-Verarbeitung nicht. Einige einfache Ergänzungen, die sich nahtlos in das bisher besprochene Konzept einreihen, ermöglichen die Erweiterung der einfachen Reports zu einer interaktiven List-Verarbeitung, die auf spezielle, vom Anwender ausgelöste Ereignisse reagieren kann. Mit Hilfe dieser zusätzlichen Sprachelemente kann beispielsweise folgende Funktionalität programmiert werden:

- Auswahl eines Datensatzes einer Liste und Rückgabe der Information an ein rufendes Programm (Beispiel: Suchhilfe).
- Verzweigen in untergeordnete Listen mit zusätzlichen Informationen.
- Flexible Darstellung von Datenstrukturen oder hierarchischen Listen (Beispiel: Object Browser).
- Eingabe von Werten und Weiterverarbeitung im Report (Beispiel: Auflösestufe in diversen Fehlerlisten, z.B. des Korrektur- und Transportwesens).
- Realisierung von Hypertextanwendungen (Beispiel: Dynpro-basierte Online-Hilfe).

Um einen Report mit interaktiven Elementen zu ergänzen, sind bis zu vier Arbeitsschritte erforderlich:

- Erzeugen einer Oberfläche mit eigenen Funktionscodes.
- Verbinden dieser Oberfläche mit dem Report.
- Abfangen der eigenen Funktionscodes mit Zeitpunktanweisungen.
- Programmieren der individuellen Funktionalität.

Der folgende Abschnitt soll diese Arbeitsschritte näher erläutern und Beispiele zur praktischen Anwendung vorstellen.

3.5.1 Ereignisse in interaktiven Reports

Zusätzlich zu den bereits beschriebenen Ereignissen sind für das interaktive Reporting drei Gruppen von Ereignissen von Bedeutung, um Aktivitäten des Anwenders zu erkennen und zu verarbeiten. Diese Ereignisse werden jeweils über eine eigene Zeitpunktanweisung verarbeitet.

Ein Doppelklick mit der Maus oder die F2-Taste löst ein Ereignis aus, das durch den Zeitpunkt

```
AT LINE-SELECTION.
```

erkannt wird. Voraussetzung dafür ist, dass im aktuellen Status des Reports der Funktionscode PICK existiert und der Funktionstaste F2 zugewiesen wurde. Das ist beim automatisch gesetzten Standardstatus für Reports automatisch der Fall. Mittels einiger Systemfelder oder der noch zu beschreibenden Anweisung HIDE können Informationen über die selektierte Zeile ermittelt werden.

Alle Funktionscodes, die nicht automatisch durch das System verarbeitet werden, behandelt die Anweisung

```
AT USER-COMMAND.
```

Dieser Zeitpunkt ist eine verallgemeinerte Form von AT LINE-SELECTION. Zu diesem Zeitpunkt werden alle Systemfelder wie beim Selektieren einer List-Zeile mit Doppelklick oder der Taste F2 gefüllt. Außerdem wird der Funktionscode der gewählten Funktion in das Feld SY-UCOMM gestellt. Er wird innerhalb des Anweisungsblocks der Zeitpunktanweisung auf ähnliche Weise weiterverarbeitet wie der Funktionscode einer Dialoganwendung im OK-CODE-Modul. Als aktuelle Zeile wird diejenige benutzt, auf die der Cursor platziert wurde.

Der Vorteil des Zeitpunkts AT USER-COMMAND gegenüber AT LINE-SELECTION liegt darin, dass für die aktuelle Zeile mehrere unterschiedliche Aktionen ausgeführt werden können. Individuelle Funktionscodes, die über AT USER-COMMAND ausgewertet werden können, stehen nur zur Verfügung, wenn mit dem Menu Painter eine eigene Oberfläche erzeugt und im Report aufgerufen wird.

Eine Sonderrolle nimmt der Zeitpunkt

```
AT PFxx.
```

ein. Falls eine der Funktionstasten mit dem Funktionscode PFxx. belegt wurde, wobei xx der Nummer der Funktionstaste entspricht, wird nicht AT USER-COMMAND prozessiert, sondern AT PFxx. Da Funktionscodes eigentlich nicht mit dem Buchstaben „P“ beginnen sollen, eignet sich diese Anweisung vor allem für Testzwecke. Auch zur Nutzung dieser Zeitpunktanweisung muss die Oberfläche modifiziert werden.

3.5.2 Verzweigungslisten

Eine der wichtigsten Aufgaben der interaktiven Reports ist die Reaktion auf Auswahl einer Zeile der Liste per Doppelklick oder Funktionstaste F2. Der folgende kurze Report demonstriert zunächst das Prinzip.

```
REPORT yz434170 NO STANDARD PAGE HEADING.
DO 30 TIMES.
  WRITE: / 'Line', sy-index.
  WRITE /.
ENDDO.

AT LINE-SELECTION.
  WRITE: / 'List-Level            :', sy-lsind COLOR 4.
  WRITE: / 'Line rel.             :', sy-curow COLOR 4.
  WRITE: / 'Column rel.           :', sy-cucol COLOR 4.
  WRITE: / 'Linenumber            :', sy-lilli COLOR 4.
  WRITE: / 'Content               :', (40)sy-lisel COLOR 4.
```

Der Report erzeugt zunächst eine Liste, in der 30 Textzeilen mit jeweils einer nachfolgenden Leerzeile ausgegeben werden. Ein Doppelklick auf eine dieser Zeilen wechselt zu einem neuen Bildschirm, auf dem einige Werte erscheinen. Bei diesen Werten handelt es sich um die Nummer der Liste, die Position des Mauszeigers bezogen auf das Bildschirmfenster sowie die absolute Zeilennummer und den Inhalt der selektierten Zeile. Alle Informationen werden durch automatisch gefüllte Systemfelder zur Verfügung gestellt. Im allgemeinen Sprachgebrauch wird die erste Liste als Grundliste, alle weiteren als Verzweigungsliste bezeichnet.

Selbst dieses kurze Programm lässt einige aufschlussreiche Experimente zu. Ein Doppelklick in der Verzweigungsliste führt zu einer erneuten Listen-Erstellung. Dies ist vor allem an der Erhöhung des Wertes für `SY-LSIND` sichtbar. Bei jedem Doppelklick in der Liste erhöht sich der angezeigte Wert um eins, jede Betätigung der Funktionstaste F3 führt zur vorhergehenden Liste zurück. Die Zeilenselektion ist nur für Zeilen möglich, die wirklich in die Liste geschrieben wurden. Ein Doppelklick nach der letzten Zeile führt zu keiner Reaktion des Reports.

Bei Ausführung der Zeilenselektion setzt das System einige Systemfelder. Diese können, wie im Beispiel dargestellt, abgefragt und im weiteren Verlauf des Programms ausgewertet werden. Tabelle 3.36 zeigt die im interaktiven Reporting wichtigen Systemfelder.

Nach der Anweisung `AT LINE-SELECTION` beginnt die Erstellung einer neuen Liste. Nach dieser Anweisung können daher Anweisungen stehen, die Grundeinstellungen für eine neue Liste setzen. So kann mit `SET PF-STATUS` ein neuer Status gesetzt werden, die Abmessungen der Liste können verändert werden usw. Derartige Einstellungen gelten nur für die Verzweigungsliste, nicht aber für die Grundliste.

Systemfeld	Bedeutung
SY-LSIND	Nummer der Liste (Grundliste = 0).
SY-CUROW	Zeilenposition des Cursors in der letzten angezeigten Liste.
SY-CUCOL	Spaltenposition des Cursors in der letzten angezeigten Liste.
SY-LISEL	Inhalt der selektierten Zeile.
SY-LILLI	Absolute Zeilennummer der selektierten Zeile.
SY-LISTI	Nummer der Liste der selektierten Zeile.
SY-PFKEY	Name des aktuellen Status der Oberfläche.
SY-CPAGE	Erste angezeigte Seite.
SY-STARO	Erste angezeigte Zeile.
SY-STACO	Erste angezeigte Spalte.

Tabelle 3.36
Systemfelder für das interaktive Reporting

Wird beispielsweise in einer Verzweigungsliste ein neuer Status gesetzt, ist nach der Rückkehr zur Grundliste wieder der alte Status aktiv. Es existieren, abgesehen von einer Ausnahme, keine speziellen Anweisungen für Verzweigungslisten. Diese Ausnahme stellt die Anweisung `TOP OF PAGE` dar. Sie sorgt innerhalb von Grundlisten für die Erzeugung eines Seitenkopfs. Für Verzweigungslisten wird dieser Zeitpunkt nicht prozessiert. Stattdessen wird eine ähnliche Anweisung

```
TOP OF PAGE DURING LINE-SELECTION.
```

verwendet. Innerhalb dieses Zeitpunktes werden alle Verzweigungslisten behandelt. Wenn für Verzweigungslisten unterschiedlicher Ebenen auch unterschiedliche Überschriften gesetzt werden sollen, ist innerhalb der `TOP OF PAGE`-Verarbeitung das Systemfeld für die Listen-Stufe auszuwerten.

Im obigen Beispiel können beliebig viele Verzweigungen erzeugt werden, was aber in der Praxis weder notwendig noch sinnvoll ist. Soll die weitere Verzweigung unterbunden werden, ist in einer `IF`-Anweisung die Listen-Stufe abzufragen. Dabei ist zu beachten, dass unmittelbar nach `AT LINE-SELECTION` der Wert für `SY-LSIND` bereits der neuen Listen-Ebene entspricht. Falls keine Ausgaben erfolgen, wird auch keine neue Listen-Ebene auf dem Bildschirm erzeugt. Um die Verzweigung nach der ersten Verzweigungsliste zu beenden, ist folgende Modifikation erforderlich:

```
AT LINE-SELECTION.
  IF SY-LSIND < 2.
    WRITE: / 'List-Level           :', SY-LSIND COLOR 4.
    WRITE: / 'Line rel.            :', SY-CUROW COLOR 4.
    WRITE: / 'Column rel.          :', SY-CUCOL COLOR 4.
    WRITE: / 'Linenumber           :', SY-LILLI COLOR 4.
    WRITE: / 'Content              :', (40)SY-LISEL COLOR 4.
  ENDIF.
```

Eine selektierte Zeile wird so, wie sie in die Liste geschrieben wurde, in SY-LISEL bereitgestellt. Das bedeutet, dass bei eventuellen Auswertungen nur die Informationen verfügbar wären, die in der Liste erscheinen. Für Hypertextanwendungen (z.B. die Online-Hilfe) ist dies durchaus nützlich, da mit SY-CUCOL aus SY-LISEL der Begriff an der Cursorposition ermittelt werden kann. Für andere Anwendungen hingegen ist dieses Verhalten nachteilig. Sollen beispielsweise anhand der Informationen der Grundliste ein Datensatz selektiert und zu diesem weitere Informationen in der Verzweigungsliste dargestellt werden, wäre dies nur möglich, wenn in der Grundliste der komplette Schlüssel erscheint und aus dem Wert SY-LISEL herausgefiltert werden kann. Dies ist recht aufwändig, würde zu unübersichtlichen Listen führen und vor allem bei mehrzeiligen Einträgen in der Liste versagen. Mittels einer speziellen Anweisung können daher zu jeder Zeile der Liste zusätzliche Daten in einen so genannten verborgenen Datenbereich gestellt werden. Diese Daten werden aufbewahrt und bei Selektion der Zeile auch wieder zur Verfügung gestellt. Sie erscheinen nicht auf dem Bildschirm. Die Nutzung des verborgenen Datenbereiches erfordert den Einsatz eines speziellen Kommandos und eine darauf zugeschnittene Programmstruktur. Das Kommando lautet

```
HIDE field.
```

Die Anweisung stellt den aktuellen Inhalt des Feldes in den verborgenen Datenbereich der aktuellen Zeile. Beim Feld kann es sich auch um ein einfaches Feld, eine Feldleiste oder die Kopfzeile einer Tabelle handeln. Mit HIDE können beliebige Felder behandelt werden. Mehrere Felder in einer HIDE-Anweisung müssen mit Hilfe der Doppelpunktvariante des Kommandos notiert werden:

```
HIDE: field_1, field_2, ... field_n.
```

Durch einige Anweisungen, die auf eine Zeile der Liste Bezug nehmen, werden die mit HIDE verborgenen Daten wieder in das jeweilige Feld zurückgelesen. Zu den Kommandos, nach denen ein Zurücklesen erfolgt, gehört z.B. die Zeitpunktanweisung AT LINE-SELECTION. Das folgende Programmbeispiel demonstriert die Anwendung des HIDE-Kommandos:

```
REPORT yz434180 NO STANDARD PAGE HEADING.
DATA: line TYPE i VALUE 1.

FORMAT COLOR 4.
```

```
DO 30 TIMES.
  WRITE: / 'Line', sy-index, AT sy-linsz ' '.
  HIDE line.
  WRITE: / 'Next line', AT sy-linsz ' '.
  HIDE line.
  WRITE: /.
  ADD 1 TO line.
ENDDO.

AT LINE-SELECTION.
  IF sy-lsind < 2.
    WRITE: / 'Selected line:', line COLOR 4.
  ENDIF.
```

Auch mit diesem Report sind einige Experimente möglich. In der Liste werden jeweils zwei zusammengehörige Zeilen ausgegeben und zur besseren Orientierung farbig hervorgehoben. Für jede dieser Zeilen wird mit `HIDE` der Wert des Feldes `line` in den verborgenen Datenbereich gestellt. Die Zeilengruppen werden durch eine Leerzeile getrennt, für die keine `HIDE`-Anweisung erfolgt.

Nach Selektion einer Zeile per Doppelklick wird der Wert für das Feld `line` wieder aus dem verborgenen Datenbereich geholt und angezeigt. Dazu sind keinerlei Programmanweisungen notwendig, der Vorgang wird automatisch durch das System gesteuert. Die Inhalte von Feldern, die nicht mit `HIDE` verborgen wurden, bleiben nach einer Zeilenselektion unverändert. Sofern eine Zeile selektiert wird, für die keine Informationen verborgen wurden, ändern sich daher die aktuellen Feldinhalte nicht. Diese Eigenschaft führt zu einem unerwünschten Verhalten des Reports. Obwohl für die Trennzeilen keine `HIDE`-Anweisung programmiert wurde, erstellt das System bei einem Doppelklick auf eine der Leerzeilen eine Verzweigungsliste. Der für das Feld `line` angezeigte Wert entspricht dabei dem zuvor aktuellen Wert. Er ändert sich auch nach Selektieren unterschiedlicher Leerzeilen nicht.

Die Unterdrückung dieser Eigenart erfordert eine kleine Modifikation im Programm. Die Verzweigung in neue Listen ist nun davon abhängig, ob für die selektierte Zeile Daten im verborgenen Datenbereich existieren. Die Auswertung von `SY-LSIND` wird dadurch in vielen Fällen überflüssig. Zum Test wird das vorangegangene Programm kopiert und die Kopie modifiziert.

```
REPORT yz434190 NO STANDARD PAGE HEADING.
DATA: line TYPE i VALUE 1.
FORMAT COLOR 4.

DO 30 TIMES.
  WRITE: / 'Line', sy-index, AT sy-linsz ' '.
  HIDE line.
  WRITE: / 'Next line', AT sy-linsz ' '.
```

```
  HIDE line.
  WRITE: /.
  ADD 1 TO line.
ENDDO.

CLEAR line.

AT LINE-SELECTION.
  IF NOT line IS INITIAL.
    WRITE: / 'Selected line:', line COLOR 4.
    CLEAR line.
  ENDIF.
```

Der Report entspricht im Wesentlichen der vorangegangenen Version. Neu ist lediglich, dass nach Erzeugen der Liste das Feld `line` mit der Anweisung `CLEAR` auf Initialwert gesetzt wird. Innerhalb des Anweisungsblocks für die Zeilenselektion erfolgt nun keine Abfrage auf die List-Stufe, sondern ein Test des Feldes `line` auf Initialwert. Die Verzweigungsliste wird nur dann erzeugt, wenn das Feld `line` mit einem Wert ungleich dem Initialwert belegt ist. Dies ist nur dann der Fall, wenn eine Zeile selektiert wurde, für die mit `HIDE` für das Feld `line` eine Information verborgen wurde. Nach der Erstellung der Verzweigungsliste muss `line` natürlich wieder auf Initialwert gesetzt werden.

Beim Auswerten von Datenfeldern, die aus dem verborgenen Datenbereich einer Zeile stammen, müssen die Feldnamen unbedingt übereinstimmen. Es führt zu Fehlern, wenn in der `HIDE`-Anweisung mehrere Einzelfelder einer Feldleiste verborgen werden, später aber die gesamte Feldleiste auf Initialwert getestet wird.

Da in der Verzweigungsliste keine `HIDE`-Anweisungen ausgeführt werden, verhindert die `IF`-Anweisung gleichzeitig die Erstellung weiterer Verzweigungslisten. Die verschiedenen Verzweigungslisten oder die unterschiedlichen Zeilen einer Liste können unterschiedliche Datenfelder in den verborgenen Datenbereich stellen. Zum Zeitpunkt `AT LINE-SELECTION` kann dann entweder anhand der aktuellen Feldbelegung oder aber mittels des Systemfeldes `SY-LSIND` die konkrete Aktion der Anwendung bestimmt werden.

In den bisherigen Beispielen nimmt die Verzweigungsliste stets den kompletten Bildschirm ein. Eine ABAP-Anweisung ermöglicht die Definition von Fenstern (Popups) beliebiger Größe, in denen eine Verzweigungsliste angezeigt werden kann. Die Anweisung lautet

```
WINDOW STARTING AT x1 y1 ENDING AT x2 y2.
```

Die Koordinaten bestimmen die linke obere (`x1, y1`) bzw. rechte untere (`x2, y2`) Ecke. Der Zusatz `ENDING AT` mit den beiden Parametern `x2` und `y2` ist optional. Fehlt dieser Zusatz, werden die Abmessungen des Fensters entsprechend den aktuellen Abmessungen der Liste automatisch bestimmt. Auch zu dieser

Anweisung ein kurzes Beispiel (die Änderungen betreffen nur die Erstellung der Verzweigungsliste):

```
AT LINE-SELECTION.
  IF NOT ZEILE IS INITIAL.
    WINDOW STARTING AT 5 5 ENDING AT 50 10.
    WRITE: / 'List-Level            :', SY-LSIND COLOR 4.
    WRITE: / 'Line rel.             :', SY-CUROW COLOR 4.
    WRITE: / 'Column rel.           :', SY-CUCOL COLOR 4.
    WRITE: / 'Line number           :', SY-LILLI COLOR 4.
    CLEAR ZEILE.
  ENDIF.
```

Zum Zeitpunkt `AT LINE-SELECTION` muss nicht zwangsläufig eine Liste erstellt werden. Auch andere Aktivitäten wie beispielsweise der Aufruf einer Dialoganwendung sind möglich. Das nachfolgende Beispiel demonstriert den Aufruf eines Funktionsbausteins:

```
REPORT yz434200 NO STANDARD PAGE HEADING.
DATA: trdir    TYPE trdir,       " Table with ABAP-Programs
      textpool TYPE textpool.    " Structure for texts

* internal table for texts
DATA: BEGIN OF itext OCCURS 20.
        INCLUDE STRUCTURE textpool.
DATA: END OF itext.

FORMAT COLOR 4.

SELECT * FROM trdir
  WHERE cnam = sy-uname           " User
    AND appl <> 'S'               " no generated programs
    AND subc IN ('1', 'M', 'F') . " no includes

* write program name
  WRITE: /(60) trdir-name.
  HIDE trdir.

* read texts
  CLEAR itext.
  REFRESH itext.
  READ TEXTPOOL trdir-name INTO itext LANGUAGE sy-langu.

* find description
  CLEAR itext.
  itext-id = 'R'.
  READ TABLE itext.

* write description
  IF sy-subrc = 0.
```

```
    WRITE: /(60) itext-entry.
    HIDE trdir.
  ENDIF.
  WRITE /.
ENDSELECT.

CLEAR trdir.

AT LINE-SELECTION.
  IF NOT trdir IS INITIAL.

* call editor for program
    CALL FUNCTION 'EDITOR_PROGRAM'
      EXPORTING
        program   = trdir-name
        MESSAGE   = ' '
        display   = 'X'          " SPACE for edit,
                                   X for show only
        trdir_inf = trdir.
    CLEAR trdir.
  ENDIF.
```

Aus der Tabelle TRDIR werden die Attribute aller Programme gelesen, die der aktuelle Benutzer erstellt hat. Der Test einiger der Attribute sorgt dafür, dass keine Include-Dateien und keine vom System generierten Programme in der Liste erscheinen. In der Tabelle TRDIR sind nur einige allgemeine Verwaltungsinformationen enthalten. Die Kurzbeschreibung muss separat gelesen werden. Alle sprachabhängigen Elemente einer Anwendung, z.B. Textelemente, Überschriften oder auch die Kurzbeschreibung, werden getrennt von diesen Verwaltungsdaten abgespeichert und müssen bei Bedarf durch zusätzliche Anweisungen gelesen werden. Dazu steht eine spezielle Variante des Kommandos READ zur Verfügung. Es ist nicht erforderlich, direkt mit dem Kommando SELECT auf Tabellen zuzugreifen. Das READ-Kommando füllt eine interne Tabelle mit allen Texten der verschiedenen Arten. Der Kurztext zum Programm muss mittels eines Kennbuchstabens aus der internen Tabelle gelesen werden. Existiert eine Kurzbeschreibung, wird sie ebenfalls auf der Liste ausgegeben. Für beide Zeilen wird die gesamte Kopfzeile der Tabelle TRDIR in den verborgenen Datenbereich gestellt. Bei Auswahl einer Zeile wird der Programmeditor für das Programm aufgerufen. Das Flag DISPLAY sorgt dafür, dass der Editor nach dem Aufruf zunächst im Anzeigemodus arbeitet.

Neben dem Aufruf eines Funktionsbausteins werden oft folgende Funktionen mit dem interaktiven Reporting ausgelöst:

- Bereitstellen von Informationen zum selektierten Satz in globalen Feldern oder anwendungsübergreifenden Speicherbereichen und Verlassen des Reports mit LEAVE.
- CALL TRANSACTION zum Aufruf einer Transaktion.

- `SUBMIT` zum Aufruf eines anderen Reports.
- `CALL FUNCTION` zum Aufruf von Funktionsbausteinen.
- `CALL DIALOG` zum Aufruf von Dialogbausteinen.
- `CALL SCREEN` zum Aufruf von Dynpros, die zum aktuellen Programm gehören.

Besonders interessant ist die letzte Variante wegen der Tatsache, dass auch ein Report über eigene Dynpros verfügen kann. Innerhalb einer Dialoganwendung kann analog dazu mit der Anweisung `LEAVE TO LIST-PROCESSING` der Listen-Modus eingeschaltet werden. Die Grenzen zwischen Report und Dialoganwendung werden damit etwas durchlässiger. Die intensive Vermengung von Listen- und Dialogverarbeitung in einer Anwendung führt allerdings schnell zu unübersichtlichen Programmen.

Falls in einem Report ein Dynpro angelegt werden soll, kann dies entweder über die Objektliste des Object Browsers oder mittels des Navigationsmechanismus geschehen. Im Quelltext wird die `CALL SCREEN`-Anweisung notiert. Nach einem Doppelklick auf die Nummer des Dynpros erzeugt das System dieses Dynpro, wobei der Programmierer einige zusätzliche Angaben in diversen Dynpros eintragen muss.

Alle der oben beschriebenen Beispiele funktionieren prinzipiell auch für den Zeitpunkt `AT USER-COMMAND`, falls dieser Zeitpunkt durch Auslösen eines eigenen Funktionscodes prozessiert wird. Auch bei diesem Zeitpunkt wird für die durch die aktuelle Cursorposition bestimmte Zeile eine Zeilenselektion durchgeführt. Falls für eine Zeile allerdings nur eine Aktion ausgeführt werden soll, z.B. immer das Verzweigen in eine zweite Liste, dann ist es für den Anwender einfacher, diese Zeile per Doppelklick zu selektieren, als zunächst den Cursor auf eine Zeile zu stellen und dann eine Drucktaste zu betätigen oder gar eine Menüfunktion zu suchen. Analog dazu können auch zum Zeitpunkt `AT LINE-SELECTION` die nachfolgend beschriebenen Modifikationskommandos stehen.

3.5.3 Modifikationen der Liste

Nicht alle Aktionen innerhalb einer Liste erfordern den Aufbau einer Verzweigungsliste. Oft reicht es aus, Zeilen der aktuellen Liste zu be- oder verarbeiten. Ein Beispiel sind diverse Auswahllisten, z.B. die in Matchcodes. Je nach Art der Liste können ein oder mehrere Datensätze markiert und in das rufende Programm übernommen werden. Diese Auswahl erfordert zum Teil die Modifikation der Einträge in einer existierenden Liste.

Das wichtigste Kommando zum Modifizieren einer Liste ist das Kommando `MODIFY`. Es existiert in mehreren Varianten, wobei für jede Variante wiederum einige Zusätze möglich sind. Für die Demonstration des Prinzips soll zunächst das Kommando

```
MODIFY CURRENT LINE.
```

ausreichen. Dieses Kommando bezieht sich stets auf die letzte mit einer Zeilenselektion ausgewählte Zeile. In der oben notierten Grundform wird der Inhalt des Systemfeldes SY-LISEL an der Position der Zeile in der Liste ausgegeben. Das Kommando MODIFY kann nur existierende Zeilen ändern, aber keine neuen anlegen. Bei erfolgreicher Ausführung belegt das Kommando das Systemfeld SY-SUBRC mit dem Wert 0. Konnte MODIFY die zu ändernde Zeile nicht finden, wird SY-SUBRC mit einem Wert ungleich 0 gefüllt. Der Fehlercode ist immer nur davon abhängig, ob die zu ändernde Zeile gefunden wurde oder nicht. Die noch zu beschreibenden Zusätze haben keinen Einfluss auf den Inhalt von SY-SUBRC.

Zur Abarbeitung der nachfolgenden Beispiele wird jeweils ein Status benötigt, der neben den für die Listenverarbeitung erforderlichen Standardfunktionscodes vier zusätzliche Funktionscodes der Form FNCx liefern kann. Legen Sie einen Status an und weisen Sie ihm innerhalb des Menu Painter über die Menüfunktion ZUSÄTZE | VORLAGEN ABGLEICHEN | LISTSTATUS zunächst die Vorgabewerte für einen Report zu.

Die individuellen Funktionscodes des Status sollten über die Funktionstasten ab F5 und über Drucktasten verfügbar sein. Für die verschiedenen Beispiele können Sie das erste Programm dann immer wieder kopieren und die Kopie entsprechend modifizieren. Dabei müssen Sie im Popup, in dem die zu kopierenden Elemente ausgewählt werden können, den Punkt GUI-Status explizit markieren. Das Programm selbst ist etwas umfangreicher, da zur Demonstration aller Möglichkeiten eine interne Tabelle benötigt wird.

```
REPORT yz434210 NO STANDARD PAGE HEADING.
DATA: BEGIN OF itab OCCURS 5,
  select,
  name(20),
END OF itab.

itab-select = ' '.
itab-name = 'Line 1'.
APPEND itab.

itab-name = 'Line 2'.
APPEND itab.

itab-name = 'Line 3'.
APPEND itab.

itab-name = 'Line 4'.
APPEND itab.

itab-name = 'Line 5'.
APPEND itab.

SET PF-STATUS 'STAT1'.
```

```
LOOP AT itab.
  WRITE: / itab-select, itab-name.
ENDLOOP.

AT USER-COMMAND.
  CASE sy-ucomm.
    WHEN 'FNC1'.
      sy-lisel+30 = 'modified'.
      MODIFY CURRENT LINE.
  ENDCASE.
```

In diesem Programm wird per Funktionscode `FNC1` (der Doppelklick funktioniert hier nicht!) ein zusätzlicher Text an die Zeile angefügt, in welcher sich der Cursor befindet. Während die ursprünglichen Zeilen der Liste in einem hervorgehobenen Farbton erscheinen (meist in Blau, je nach Systemeinstellung), werden die hinzugefügten Zeichenketten in normaler Farbe (für gewöhnlich in Schwarz) dargestellt.

Nachträgliche Änderungen des Ausgabeformats sind mühelos möglich. Dazu wird der `AT USER-COMMAND`-Teil des ersten Beispiels durch die Auswertung eines zweiten Funktionscodes etwas erweitert:

```
AT USER-COMMAND.
...
    WHEN 'FNC2'.
      MODIFY CURRENT LINE LINE FORMAT INTENSIFIED OFF.
```

Der Funktionscode `FNC2` schaltet die Intensivdarstellung für die aktuelle Zeile ab. Nach `FORMAT` können alle Optionen dieses Kommandos verwendet werden. Sie beziehen sich in diesem Beispiel, bedingt durch das zweite `LINE`, auf die gesamte Zeile. Die mehrfache Verwendung des Begriffes `LINE` in der Anweisung ist zwar nicht besonders elegant, syntaktisch aber völlig korrekt. Das erste `LINE` gehört zu `CURRENT`, das zweite legt fest, dass `FORMAT` für die ganze Zeile gültig sein soll. Die ausdrückliche Verwendung von `LINE` lässt die Vermutung zu, dass auch einzelne Felder einer Zeile modifiziert werden können. Dies erfolgt für das Format mit dem Zusatz

```
... FIELD FORMAT field_1 format_1 ... field_n format_n.
```

Die Namen der zu modifizierenden Felder und die jeweiligen Formate werden hintereinander notiert. Auch dieses Kommando kann recht einfach durch einen dritten Funktionscode demonstriert werden, der für die ersten beiden Felder der Zeile die Farbdarstellung ändert:

```
AT USER-COMMAND.
...
    WHEN 'FNC3'.
      MODIFY CURRENT LINE FIELD FORMAT itab-select COLOR 7
                          INVERSE ON itab-name COLOR 5.
```

Die feldbezogene Modifikation funktioniert nur dann, wenn Datenfelder in die Liste ausgegeben wurden. Die Eigenschaften von Festwerten können auf diese Weise nachträglich nicht geändert werden. Textelemente sind für ABAP auch eine Art Datenfeld, sie können daher ebenfalls mit neuen Formaten versehen werden.

Um die Modifikation einzelner Felder zu ermöglichen, muss die interne Steuerlogik für jede Zeile die ausgegebenen Felder mit den Ausgabepositionen aufbewahren. Mit diesen Informationen ist es auch möglich, den Feldern einer Zeile neue Inhalte zuzuordnen. Das Kommando dazu ähnelt dem zum Zuweisen neuer Formate:

```
... FIELD VALUE field_1 FROM value_1 ... field_n
                        FROM value_n.
```

Die folgenden Zeilen demonstrieren die Verwendung dieses Kommandos. Sie setzen das Feld `itab-select` der aktuellen Zeile auf den Wert „X“ und markieren den Satz dadurch für eventuelle spätere Auswertungen.

```
AT USER-COMMAND.
...
    WHEN 'FNC4'.
      MODIFY CURRENT LINE FIELD VALUE itab-select FROM 'X'.
```

Das Kommando kann etwas vereinfacht werden, indem die neuen Werte direkt in die zu verändernden Felder eingetragen werden. In der `MODIFY`-Anweisung kann die `FROM`-Klausel dann entfallen. Bei einer einzigen Anweisung wie in diesem Beispiel bringt diese Version des Kommandos keinen Vorteil, sollten aber mehrere Felder geändert werden, wird die `MODIFY`-Anweisung wesentlich übersichtlicher.

```
WHEN 'FNC4'.
  ITAB-SELECT = 'X'.
  MODIFY CURRENT LINE FIELD VALUE ITAB-SELECT.
```

Falls die modifizierten Felder bei der Erstellung der Liste in einer `HIDE`-Anweisung benutzt wurden, wird der verborgene Datenbereich ebenfalls geändert. Von dieser Feldänderung sind natürlich nur die Werte in der Liste bzw. im verborgenen Datenbereich betroffen. Die zur Erstellung der Liste verwendete interne Tabelle oder gar eine Datenbanktabelle wird durch diese Form der Zuweisung nicht geändert. Das wirft ein neues Problem auf. Der zuletzt verwendete Funktionscode markiert eine Zeile. Diese Information ist in praxisrelevanten Anwendungen natürlich auszuwerten. Es muss also eine Möglichkeit geben, die aktuellen Feldwerte aus der Liste zu lesen. Der Weg über `SY-LISEL` scheidet aus, weil Einzelwerte von dort nur sehr umständlich selektiert werden können. Mit einer speziellen Form der `READ`-Anweisung können aktuelle Werte auch direkt aus der Liste gelesen werden. Sie stellt das Gegenstück zur `MODIFY VALUE`-Anweisung dar. Die `MODIFY`-Anweisung in der vorgestellten Form arbeitet,

bedingt durch den Zusatz CURRENT LINE, mit der aktuellen Zeile zusammen. Anstelle dieses Zusatzes sind noch einige weitere möglich, die den Wirkungsbereich auf beliebige Zeilen aller im Programm existierenden Listen ausdehnen. Diese Zusätze sind sowohl für READ als auch für MODIFY möglich. Die READ-Anweisung soll daher mit einem dieser alternativen Zusätze demonstriert werden, der auch im Zusammenhang mit der MODIFY-Anweisung benutzt werden kann. Im praktischen Einsatz ist das wahlfreie Lesen in einer Liste häufiger notwendig als das wahlfreie Schreiben. Die Syntax des READ-Kommandos lautet:

```
READ LINE index FIELD VALUE
  listfield_1 INTO progfield_1 ...
  listfield_n INTO progfield_n.
```

Die namentlich anzugebenden Felder der Liste werden aus der per Zeilennummer identifizierten Zeile gelesen und in programminterne Felder geschrieben. Falls der Zusatz INTO progfield fehlt, werden die Inhalte der Listen-Felder in den ursprünglichen Feldern bereitgestellt. Alle nicht benannten Felder bleiben unverändert. Zusätzlich zum Lesen der ausdrücklich angegebenen Felder werden alle mit HIDE verborgenen Feldinhalte für diese Zeile ebenfalls zurückgelesen. Falls keine Werte im verborgenen Datenbereich existieren, bleiben die entsprechenden Felder unverändert. Das Ergebnis des Kommandos READ ist daher nur dann wirklich aussagekräftig, wenn alle per READ zu lesenden Felder unmittelbar vor Ausführung dieses Kommandos initialisiert werden. Für die Auswertung der HIDE-Felder wurde dies ja schon demonstriert. Das nachfolgende Beispiel (YZ434220) entspricht im Wesentlichen dem bereits benutzten Programm YZ434210, allerdings sind auch im Hauptprogramm einige Änderungen erforderlich. Zunächst sind ein zusätzliches Datenfeld und zwei Zeitpunktanweisungen zu notieren. Bedingt durch die Zeitpunktanweisungen für den Titel ist auch eine Anweisung notwendig, um den Beginn des „Hauptprogramms“ zu markieren. Folgende Anweisungen sind daher zwischen die Deklaration der internen Tabelle und die erste Zuweisung zur Kopfzeile der Tabelle einzufügen:

```
REPORT yz434220 NO STANDARD PAGE HEADING.
DATA: BEGIN OF itab OCCURS 5,
  select,
  name(20),
END OF itab.
DATA i TYPE i.

* Header main list
TOP-OF-PAGE.
  WRITE: / 'Please select'.
  SKIP.

* Header sub list
TOP-OF-PAGE DURING LINE-SELECTION.
  WRITE: / 'Selected lines:'.
  SKIP.
```

```
* create main list
START-OF-SELECTION.
  itab-select = ' '.
  itab-name = 'Line 1'.
  APPEND itab.

  itab-name = 'Line 2'.
  APPEND itab.

  itab-name = 'Line 3'.
  APPEND itab.

  itab-name = 'Line 4'.
  APPEND itab.

  itab-name = 'Line 5'.
  APPEND itab.

  SET PF-STATUS 'STAT1'.

  LOOP AT itab.
    WRITE: / itab-select INPUT, "AS CHECKBOX,
             itab-name.
  ENDLOOP.
  CLEAR itab.

* push buttons
AT USER-COMMAND.
  CASE sy-ucomm.

* set flag
    WHEN 'FNC1'.
      MODIFY CURRENT LINE FIELD VALUE itab-select FROM 'X'.

* reset flag
    WHEN 'FNC2'.
      MODIFY CURRENT LINE FIELD VALUE itab-select FROM ' '.

* show selected lines
    WHEN 'FNC3'.
      i = 1.
      CLEAR itab.

* read first line  from screen
      READ LINE i FIELD VALUE itab-select itab-name.

* SY-SUBRC <> 0  -> line i dosn't exist
      WHILE sy-subrc = 0.
```

```
        IF itab-select <> ' '.
          WRITE: / itab-name.
        ENDIF.

* read next line
        i = i + 1.
        CLEAR itab.
        READ LINE i FIELD VALUE itab-select itab-name.
      ENDWHILE.
  ENDCASE.

* select line with double click
AT LINE-SELECTION.
  CLEAR itab.

* read flag
  READ CURRENT LINE FIELD VALUE itab-select.

* modify flag
  IF itab-select = ' '.
    MODIFY CURRENT LINE FIELD VALUE itab-select FROM 'X'.
  ELSE.
    MODIFY CURRENT LINE FIELD VALUE itab-select FROM ' '.
  ENDIF.
```

Bis zum Setzen des Status entspricht dieses Programm dem vorangegangenen Beispiel. Sie können ihn daher relativ einfach mit der Funktion ZUSÄTZE | VORLAGE ABGLEICHEN aus dem Programm `YZ434210` übernehmen (siehe Abbildung 3.60).

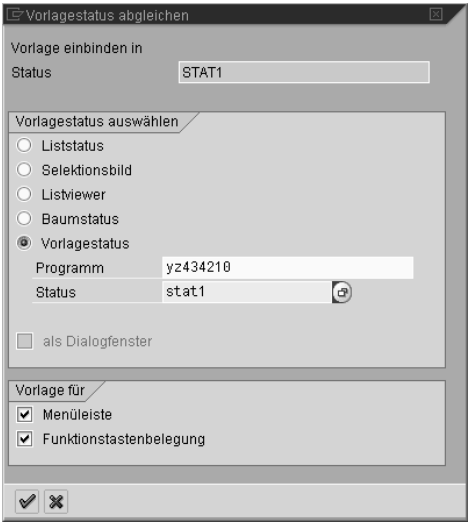

Abbildung 3.60 **© SAP AG**
Existierenden Status eines anderen Programms als Vorlage übernehmen

Das Feld I übernimmt später beim Lesen der Liste die Funktion eines Zeilenzählers. Die Schleife zur Erstellung der Grundliste weist ebenfalls eine Modifikation auf. Bei der Ausgabe des Feldes SELECT wird diesem das Format INPUT zugewiesen. Ein derartiges Feld ist in der Liste eingabebereit, es kann durch den Anwender manuell mit einem Wert gefüllt werden. Dieser Wert ist nur innerhalb der Liste vorhanden, die interne Tabelle wird durch diese Eingabe nicht geändert. Dieses Verhalten entspricht Feldänderungen mit MODIFY. Die Eingaben in ein solches Feld werden später mittels der READ-Anweisung gelesen.

Zum Zeitpunkt AT USER-COMMAND werden mit den Funktionscodes FNC1 und FNC2 die Flags entweder gesetzt oder zurückgesetzt. Diese Kommandos bieten nichts Neues, sie wurden bereits im vorangegangenen Beispiel benutzt. Interessant ist die Verarbeitung zum Funktionscode FNC3. Hier werden alle Sätze der Grundliste gelesen. Alle markierten Datensätze werden in eine Verzweigungsliste geschrieben.

Zum Auslesen wird das Kommando READ in der bereits erwähnten Form benutzt. Zum Lesen ist ein Satzzähler erforderlich, der vor dem Zugriff auf die erste Zeile auf einen Anfangswert gesetzt werden muss. Die Feldleiste, der die durch READ zu lesenden Felder angehören, wird ebenfalls vor jedem READ initialisiert. Der Rückgabecode von READ entspricht dem von MODIFY. Er gibt also nur Aufschluss darüber, ob eine zu lesende Zeile existiert oder nicht. Ob in dieser Zeile das zu lesende Feld vorhanden ist, kann aus dem Fehlercode von READ nicht ermittelt werden. Der Inhalt von itab-select ist nach READ nur dann ungleich dem Initialwert, wenn des Feld wirklich gefunden und in der Liste mit einem Wert ungleich dem Leerzeichen belegt wurde. Nur in diesem Fall wird das Feld name der jeweiligen Zeile in die Verzweigungsliste ausgegeben. Die Verzweigungsliste wird erst nach der Bearbeitung aller Anweisungen des AT USER-COMMAND-Blocks angezeigt. Daher lesen alle weiteren READ-Anweisungen immer noch aus der Grundliste. In komplexeren Programmen muss stets darauf geachtet werden, welche der Listen gerade aktiv ist, damit nicht in der falschen Liste gelesen wird.

Nach der Auswertung einer Zeile der Liste wird der Zeilenzähler erhöht und die nächste Zeile gelesen. Falls eine Zeile nach dem Ende der Liste gelesen werden soll, liefert READ einen Rückkehrcode ungleich Null. Dies kann als Abbruchkriterium dienen.

In solchen Auswahllisten kann natürlich auch das Mittel der Zeilenselektion benutzt werden, um eine Markierung ein- und auszuschalten. Der Zeitpunkt AT LINE-SELECTION zeigt die dazu notwendigen Anweisungen. Zunächst wird die aktuelle Zeile gelesen. Je nach Zustand des Flags wird es dann entweder gesetzt oder zurückgesetzt. Dies ist einer der Fälle, in dem das Lesen der aktuellen Zeile mit dem Zusatz CURRENT LINE sinnvoll ist.

Die Felder für das Selektionsflag jedes Datensatzes können auch manuell mit einem Wert belegt werden. Das Programmbeispiel wertet alle Inhalte außer dem Leerzeichen als gesetztes Flag. Innerhalb der Dynpros, in denen ja echte Ankreuz- oder Auswahlfelder definiert werden können, sorgt die interne Steuer-

logik dafür, dass markierte Felder immer durch den Buchstaben „X“ symbolisiert werden. Möglicherweise sind andere Anwendungen daher so programmiert, dass nicht auf Ungleichheit zum Leerzeichen, sondern auf Gleichheit mit dem Buchstaben „X“ geprüft wird. Fehleingaben des Anwenders könnten so zum unerwünschten Verhalten der Anwendung führen. Zur Gestaltung von Ankreuzfeldern stellt die WRITE-Anweisung daher eine zusätzliche Option bereit. Sie kann durch Austausch einer einzigen Zeile im Beispiel getestet werden:

```
WRITE: / itab-select AS CHECKBOX,
         itab-name.
```

Anstelle des Eingabefelds für beliebige alphanummerische Daten erscheint in der Liste nun ein Ankreuzfeld. Ein solches Feld kann per Mausklick direkt in das Feld umgeschaltet werden. Dazu reicht ein einzelner Mausklick! Ein markiertes Feld wird intern durch ein „X“, ein nicht markiertes durch ein Leerzeichen abgebildet. Das Umschalten der Markierung funktioniert auch ohne die Anweisungen nach AT LINE-SELECTION. Die beiden MODIFY-Kommandos dieses Zeitpunkts können vorübergehend auskommentiert werden. Allerdings sorgt die Zeilenselektion dafür, dass die Markierung durch einen Doppelklick an einer beliebigen Stelle der Zeile gesetzt werden kann, während die Checkbox-Eingabefelder nur auf einen Mausklick innerhalb des Feldes reagieren.

Die bisherigen Beispiele ermöglichen nur Modifikationen innerhalb einer Zeile, ohne dabei die Struktur oder Länge der Liste zu verändern. Einige der anfangs erwähnten Einsatzmöglichkeiten für das interaktive Reporting gehen aber davon aus, dass bei Selektion einer Zeile die Liste neu gestaltet wird. Ein Doppelklick auf einen Knoten in der Objektliste der Workbench zeigt keine Verzweigungsliste an, sondern baut die aktuelle (die Grundliste) neu auf. Eine derartige Funktionalität kann am einfachsten durch Verbindung eines Dynpros mit der eingebetteten List-Verarbeitung erreicht werden. Das Dynpro selbst ist dabei inaktiv, es wird mit SUPPRESS DIALOG dunkel prozessiert. Im PAI-Teil des Dynpros wird mittels LEAVE TO LIST-PROCESSING eine Liste aufgebaut. Ein Anwenderkommando oder eine Zeilenselektion in dieser Liste lösen eine Zeitpunktanweisung aus, die mittels LEAVE LIST-PROCESSING die List-Verarbeitung zunächst beendet. In diesem Fall wird das Dynpro aber erneut ausgeführt, die Liste also erneut aufgebaut. Am Ende des Abschnitts 3.8 finden Sie ein umfangreicheres Beispiel, das die Verknüpfung von dialogorientierter Anwendung und List-Verarbeitung demonstriert.

3.6 Logische Datenbanken

In den vorangegangenen Abschnitten wurde angedeutet, dass die Selektion der für einen Report benötigten Daten recht aufwändig werden kann. Oft sind mehrere Tabellen logisch voneinander abhängig und bilden eine Hierarchie. Jeder Report muss diese hierarchischen Abhängigkeiten durch entsprechend verschachtelte SELECT-Anweisungen oder ähnliche Konstruktionen bei der Selektion von

Daten berücksichtigen. Wenn mehrere Reports für ähnliche Auswertungen auf dieselben Dateien zugreifen und immer wieder dieselben Abhängigkeiten beachten müssen, werden immer wieder identische oder nahezu identische Abfragen erforderlich. Der damit verbundene Programmieraufwand kann mit so genannten logischen Datenbanken erheblich reduziert werden.

Logische Datenbanken sind keine eigenständig nutzbaren Objekte. Sie können sie nur in Verbindung mit einem Report sinnvoll einsetzen. Beim Erzeugen eines neuen Reports können Sie im Attributebildschirm den Namen einer logischen Datenbank eintragen. Der Report benutzt diese dann automatisch zur Selektion von Datensätzen.

3.6.1 Aufgabe und Bestandteile

Eine logische Datenbank ist ein speziell aufgebautes Programm. Sie übernimmt die Selektionen von Datensätzen in mehreren logisch miteinander verbundenen Tabellen und stellt die gelesenen Datensätze dem eigentlichen Report zur Verfügung. Dieser bekommt die Datensätze in der korrekten Reihenfolge geliefert und muss sie nur noch auswerten und anzeigen. Das Bereitstellen von Datensätzen durch die logische Datenbank sowie das Lesen der Datensätze im eigentlichen Report erfolgen über das Kommandopaar `PUT` und `GET`, wobei `GET` eine Zeitpunktanweisung ist.

Eine logische Datenbank besteht aus drei wesentlichen Elementen:

- Struktur (Tabellenhierarchie),
- Selektionen,
- Datenbankprogramm.

Struktur

Die durch die logische Datenbank zu verarbeitenden Tabellen müssen in einer Baumstruktur angeordnet werden. In dieser Struktur darf jede Tabelle nur einmal vorkommen. Die Zweige der Struktur legen die Abhängigkeiten bzw. die Rangordnung der Tabellen untereinander fest. Die Struktur wird mit einem pseudografischen Editor gepflegt. Ausgehend von einer Wurzel werden die untergeordneten Tabellen schrittweise zugeordnet. Die Struktur ist kein direkt nutzbares Objekt. Sie stellt lediglich die Tabellen, die in die logische Datenbank aufgenommen werden, und deren Abhängigkeiten untereinander dar. Aus diesen Angaben generiert das System zwei Programmdateien, das Datenbankprogramm und das Selektions-Include. Dabei besteht das Datenbankprogramm in den neueren Versionen der R/3-Software selbst aus mehreren Includes.

Selektionen

Die Selektionen sind, worauf der Name bereits hindeutet, eine Reihe von SELECT-OPTIONS-Anweisungen. Für die Schlüsselfelder aller Tabellen der logischen Datenbank wird jeweils eine eigene SELECT-OPTIONS-Anweisung generiert. Diese Anweisungen legt das System in einer Include-Datei ab, die wiederum in das Datenbankprogramm eingebunden wird. Diese Datei mit den Selektionen muss durch den Programmierer gemäß seiner speziellen Wünsche weiterbearbeitet werden. Es können Selektionen oder Parameter für weitere Felder eingefügt oder vorhandene Selektionen gelöscht werden. Der nachfolgende Programmausschnitt zeigt ein rudimentäres Selektionsprogramm. Der Name des Selektions-Includes wird nach dem Muster DBldb_nameSEL gebildet.

```
*---------------------------------------------------------------*
* INCLUDE DBYLB1SEL
* It will be automatically included into the
* database program.
*---------------------------------------------------------------*
...
SELECT-OPTIONS :
*                   ?  FOR TADIR-PGMID,
*                   ?  FOR TADIR-OBJECT,
*                   ?  FOR TADIR-OBJ_NAME.
  s_author for tadir-author.
* Parameter for search pattern selection (Type SY-LDB_SP):
* PARAMETERS p_sp AS SEARCH PATTERN FOR TABLE TADIR.

* SELECT-OPTIONS :
*                   ?  FOR E071-TRKORR,
*                   ?  FOR E071-AS4POS.

* SELECT-OPTIONS :  ?  FOR E070-TRKORR.
...
```

Datenbankprogramm

Das Datenbankprogramm ist das eigentliche Hauptprogramm der logischen Datenbank. Es enthält nach der Generierung einige leere oder nur aus Kommentaren bestehende und damit funktionslose Unterprogramme sowie SELECT-Anweisungen für jede Tabelle der logischen Datenbank. Je nach R/3-Version kann dieses Programm selbst aus mehrfach verschachtelten Includes bestehen. Der Name des Datenbankprogramms wird durch das Muster SAPDBldb_name beschrieben.

Für die SELECT-Anweisungen wird eine WHERE-Klausel generiert, die Abfragen für alle Schlüsselfelder der jeweiligen Tabelle enthält. In diese Abfragen trägt das System als Vorschlag automatisch Vergleiche mit Schlüsselfeldern der übergeordneten Tabelle ein, falls die beteiligten Tabellenfelder auf dem selben Da-

tenelement beruhen. Damit wird erreicht, dass zu einem Datensatz einer übergeordneten Tabelle die zugehörigen Datensätze der untergeordneten Tabelle(n) gelesen werden. Sie müssen die WHERE-Klausel entsprechend Ihrer Vorstellung anpassen. Sie können dort beispielsweise die Selektionen aus dem oben gezeigten Selektions-Include einfügen, sofern dort welche definiert wurden. Die aktuellen Werte der Schlüsselfelder der übergeordneten Tabelle(n) sollten Sie auf jeden Fall berücksichtigen, um brauchbare Ergebnisse zu erlangen. Innerhalb der SELECT-Anweisung wird der aktuelle Datensatz mit dem Kommando PUT an den auswertenden Report gesendet.

In den generierten WHERE-Klauseln berücksichtigt das System die in der Struktur der logischen Datenbank vorgegebenen Abhängigkeiten. Beim Lesen einer untergeordneten Tabelle wird auf Schlüsselfelder der übergeordneten Tabelle Bezug genommen, wie der folgende Ausschnitt aus einem Datenbankprogramm zeigt. Falls die Verknüpfung zwischen den Tabellen über Felder erfolgen soll, die nicht in beiden Tabellen Schlüsselfelder sind, müssen Sie diese Abfragen selbst programmieren.

```
FORM PUT_TADIR.
  SELECT * FROM TADIR
*   INTO TADIR
*   INTO TABLE ? (choose one!)
*   WHERE PGMID = ?
*     AND OBJECT = ?
*     AND OBJ_NAME = ?.
    where author in p_author.
    PUT TADIR.
  ENDSELECT.
ENDFORM.                                "PUT_
...
FORM PUT_E071.
* SELECT * FROM E071
*   INTO E071
*   INTO TABLE ? (choose one!)
*   WHERE TRKORR = ?
*     AND AS4POS = ?.
    PUT E071.
* ENDSELECT.
ENDFORM.                                "PUT_
...
FORM PUT_E070.
* SELECT * FROM E070
*   INTO E070
*   INTO TABLE ? (choose one!)
*   WHERE TRKORR = E071-TRKORR.
    PUT E070.
* ENDSELECT.
ENDFORM.                                "PUT_
```

Neben den Unterprogrammen für die SELECT-Anweisungen existieren weitere Unterprogramme, die vom System automatisch zu bestimmten Zeitpunkten der Verarbeitung des Selektionsbildschirms aufgerufen werden. Die Namen der Unterprogramme und die entsprechenden Zeitpunkte zeigt die Tabelle 3.37.

Unterprogramm	Bemerkung
LDB_PROCESS_INIT	Mehrfachverarbeitung einer logischen Datenbank initialisieren. Wird unmittelbar beim Start aufgerufen.
LDB_PROCESS_CHECK_SELECTIONS	Nach Eingabe der Selektionswerte. Prüfung der Selektionen.
INIT	Einmalige Ausführung vor erstmaliger Anzeige des Selektionsbildes.
PBO	Ausführung vor jeder Anzeige des Selektionsbildes.
PAI	Ausführung nach jeder Anzeige des Selektionsbildes.
BEFORE_EVENT	Ausführung vor verschiedenen Zeitpunktanweisungen.
AFTER_EVENT	Ausführung nach verschiedenen Zeitpunktanweisungen.

Tabelle 3.37
Unterprogramme und Zeitpunkte im Selektionsbildschirm einer logischen Datenbank

3.6.2 Freie Abgrenzung

Logische Datenbanken erleichtern die Programmierung von Reports, schränken wegen der vorgegebenen Selektionsmöglichkeiten aber auch deren Flexibilität ein. Einen Ausweg bietet die so genannte *Freie Abgrenzung*. Sie ermöglicht es dem Anwender eines Reports, zur Laufzeit der Anwendung beliebige Selektionskriterien für eine oder mehrere Tabellen zu formulieren. Dies erspart die Programmierung einer Vielzahl von einzelnen Selektionen oder Parametern.

Praktische Anwendung

Falls für eine Tabelle eine freie Abgrenzung definiert wurde, erscheint in der Drucktastenzeile des Selektionsbildes die Drucktaste FREIE ABGRENZUNGEN (siehe Abbildung 3.61).

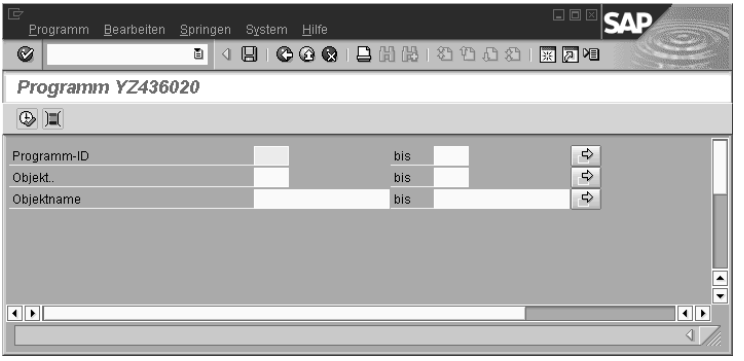

Abbildung 3.61 **© SAP AG**
Selektionsbild einer logischen Datenbank mit freier Abgrenzung

Nach Betätigen dieser Drucktaste bietet das System ein zweigeteiltes Dynpro an. Im oberen Teil erscheint eine Liste mit allen Tabellen, für die freie Abgrenzungen definiert wurden (siehe Abbildung 3.62). In dieser Übersicht erschienen allerdings nicht die Tabellennamen, sondern deren Kurzbeschreibung. Das ist für den Endanwender hilfreich, für den Programmierer aber etwas gewöhnungsbedürftig.

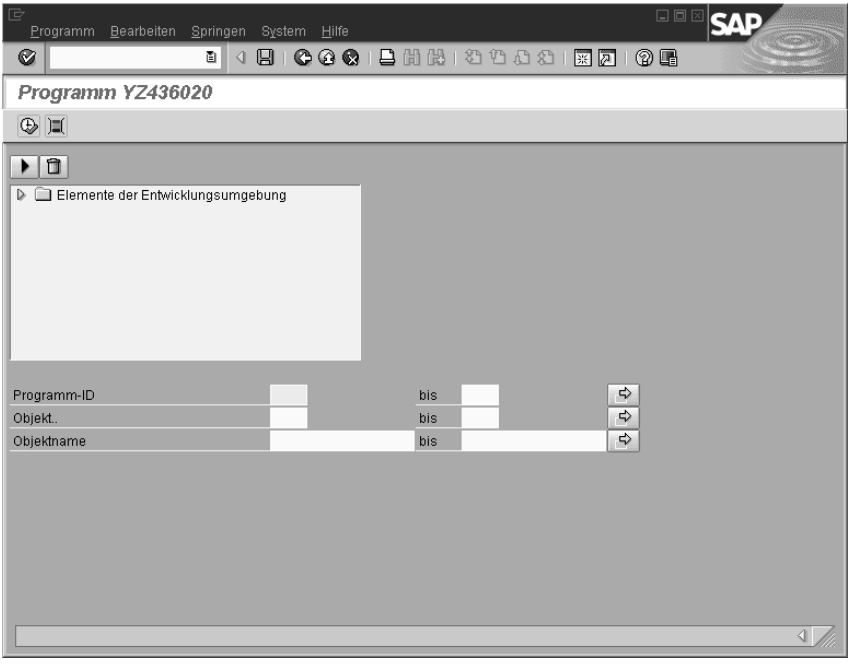

Abbildung 3.62 **© SAP AG**
Auswahl der Tabellen für eine freie Abgrenzung

Ein Mausklick auf das Symbol vor dem Tabellennamen blendet die Liste der Tabellenfelder ein. In dieser Liste können Sie Felder durch einen Mausklick markieren. Ein Druck auf die Schaltfläche mit dem Pfeilsymbol bewirkt die Übernahme einer Selektion für das selektierte Feld in das rechte Bildschirmfenster (siehe Abbildung 3.63). Der Anwender des Reports kann nun alle interessierenden Felder markieren.

Sie können auf diese Weise alle interessierenden Felder in die Liste der Selektionen übernehmen. Die von Ihnen gewünschten Selektionswerte tragen Sie auf die selbe Art und Weise ein wie im Selektionsbild eines Reports. Nach Abschluss der Eingabe müssen Sie die soeben erfassten Werte der freien Abgrenzung mit der Menüfunktion ABGRENZUNGEN | SICHERN speichern.

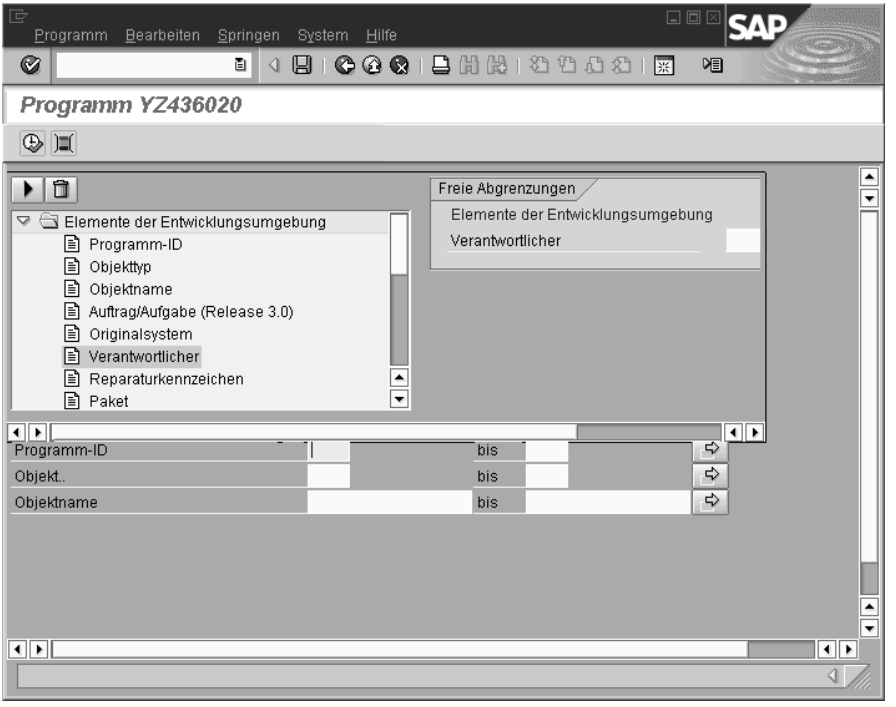

Abbildung 3.63 **© SAP AG**
Auswahl der Felder für eine freie Abgrenzung

Wenn Sie zum Startbild des Reports zurückkehren, werden Sie durch eine Ausschrift innerhalb des Symbols für die freien Abgrenzungen auf die zusätzlichen Selektionskriterien hingewiesen. Sie können den Report nun auf die gewohnte Art und Weise ausführen, wobei die Selektionswerte der freien Abgrenzung ebenfalls ausgewertet werden.

Programmierung freier Abgrenzungen

Um eine freie Abgrenzung in einer logischen Datenbank zu realisieren, sind zwei Dinge notwendig. Eine spezielle Anweisung im Selektions-Include (DBxxxSEL) des Datenbankprogramms sorgt dafür, dass der Anwender das entsprechende Selektionskriterium erfassen kann. Das Selektionskriterium für eine freie Abgrenzung wird auf andere Weise erfasst und im Programm ausgewertet als einfache Parameter oder Selektionstabellen. Um dieses Selektionskriterium auswerten zu können, muss daher die Selektionsanweisung für die jeweilige Tabelle angepasst werden. Dieser Abschnitt soll Ihnen die notwendigen Kommandos näher vorstellen.

Im Selektions-Include DBxxxSEL wird durch die Anweisung

```
SELECTION-SCREEN DYNAMIC SELECTIONS
  FOR TABLE table
  [ID ident].
```

festgelegt, für welche Datenbanktabelle eine freie Abgrenzung realisiert werden soll. Falls für mehrere Tabellen freie Abgrenzungen unterstützt werden sollen, ist für jede Tabelle eine Anweisung erforderlich. Falls der Report eine Tabelle benutzt, die für eine freie Abgrenzung vorgesehen ist, erscheint im Selektionsbildschirm die Drucktaste FREIE ABGRENZUNGEN. Alle zur Laufzeit erfassten Angaben zu allen freien Abgrenzungen werden in einem komplexen Datenobjekt mit Namen DYN_SEL gespeichert. In diesem Datenobjekt existiert eine interne Tabelle mit Namen CLAUSES. Jeder Datensatz dieser Tabelle enthält in einer weiteren internen Tabelle mit Namen WHERE_TAB die Selektionskriterien für alle Datenbanktabellen. Diese Selektionskriterien müssen durch eine spezielle Variante der SELECT-Anweisung ausgewertet werden.

```
SELECT * FROM table
  WHERE (itab).
```

Diese Form der SELECT-Anweisung liest die Selektionskriterien aus einer internen Tabelle. In der WHERE-Klausel können auch noch andere Bedingungen notiert werden. Die in der WHERE-Klausel benutzte interne Tabelle muss zuvor aus dem Datenbestand von DYN_SEL mit Daten gefüllt werden.

Das Datenobjekt DYN_SEL besitzt einen sehr komplexen Aufbau. Die Definition ist im Type-Pool RSDS abgelegt. Wichtigster Bestandteil ist die Typdefinition für eine interne Tabelle RSDS_WHERE_TAB. Diese interne Tabelle wird von einer im Dictionary abgelegten Struktur abgeleitet. Sie besteht lediglich aus einem Feld des Typs CHAR mit einer Länge von 72 Zeichen.

```
TYPES: RSDS_WHERE_TAB LIKE RSDSWHERE OCCURS 5.
```

Diese Definition fließt in die Definition der Feldleiste RSDS_WHERE ein. Diese Feldleiste besteht aus einem Feld TABLENAME zur Aufnahme eines Tabellennamens und einer internen Tabelle WHERE_TAB, in der die Selektionskriterien für die in TABLENAME benannte Tabelle aufbewahrt werden.

```
TYPES:
  BEGIN OF RSDS_WHERE,
    TABLENAME LIKE RSDSTABS-PRIM_TAB,
    WHERE_TAB WHERE_TAB TYPE RSDS_WHERE_TAB,
  END OF RSDS_WHERE.
```

Mit Hilfe dieses Feldleisten-Typs wird eine weitere interne Tabelle definiert. Diese interne Tabelle dient dazu, die Selektionskriterien für mehrere Dictionary-Tabellen aufzubewahren.

```
TYPES: RSDS_TWHERE TYPE RSDS_WHERE OCCURS 5.
```

Schließlich geht die eben vorgestellte Tabellendefinition in die Typbeschreibung `RSDS_TYPE` ein. Dieser Typ enthält nun das Feld `CLAUSES`, welches eine Tabelle des Typs `RSDS_TWHERE` aufnimmt.

```
TYPES:
  BEGIN OF RSDS_TYPE,
    CLAUSES TYPE RSDS_TWHERE,
    TEXPR   TYPE RSDS_TEXPR,
    TRANGE  TYPE RSDS_TRANGE,
  END OF RSDS_TYPE.
```

Dieser Datentyp enthält zwei weitere Felder, die ebenfalls sehr komplex aufgebaute Objekte enthalten. Das Feld `TEXPR` enthält die Selektionskriterien in einer speicherbaren Form, während `TRANGE` diese Kriterien in einigen `RANGES`-Tabellen ablegt. Diese `RANGES`-Tabellen können bei Bedarf vom Kommando `CHECK` benutzt werden. Letztlich kann mit dem Datentyp `RSDS_TYPE` noch ein reales Datenobjekt erzeugt werden. Dies erfolgt mit der Anweisung

```
DATA DYN_SEL TYPE RSDS_TYPE.
```

Nähere Angaben zum Aufbau der anderen Bestandteile von `DYN_SEL` können Sie jederzeit dem oben erwähnten Type-Pool entnehmen.

Aus den vorgestellten Datentypen können Sie entnehmen, dass in `DYN_SEL` stets Angaben zu allen Tabellen abgelegt sind, für die über die freien Abgrenzungen Selektionskriterien erfasst wurden. Zugriffe auf Werte in `DYN_SEL` erfolgen daher üblicherweise immer unter Verwendung des Namens der Datenbanktabelle als einschränkendes Kriterium.

3.6.3 Verwendung einer logischen Datenbank

Um eine logische Datenbank in einem Report benutzen zu können, ist deren Name im Attributebildschirm des Reports einzutragen. Der Name einer logischen Datenbank besteht aus drei Zeichen. Im Attributebildschirm wird dieser Name allerdings zweigeteilt erfasst. Ein Zeichen soll laut SAP-Empfehlungen benutzt werden, um die Zugehörigkeit der logischen Datenbank zu einer Anwendungsgruppe zu kennzeichnen. Die beiden anderen Zeichen stellen den frei verwendbaren Teil des Namens dar. Weitere direkte Verweise auf die logische Datenbank sind im Report nicht notwendig.

Im Report selbst müssen alle Tabellen, aus denen Daten gelesen werden sollen, mittels der TABLES-Anweisung deklariert werden. Anschließend kann dann im Report mit der Zeitpunktanweisung

```
GET table.
```

die Verarbeitung eines Datensatzes definiert werden. Die PUT- und die GET-Anweisung arbeiten auf eine relativ komplizierte Weise zusammen. Die eigentliche Steuerung übernimmt das System selbst. Zunächst sorgt es dafür, dass die SELECT-Schleife für die in der Hierarchie am weitesten oben stehende Tabelle ausgeführt wird. Nachdem ein Datensatz gelesen wurde, wird dieser Datensatz mit dem Kommando PUT an den Report übergeben. Der Aufruf von PUT löst ein Ereignis aus, in dessen Folge im Report der Zeitpunkt GET für die gerade bearbeitete Tabelle prozessiert wird. Dort könnte der Datensatz beispielsweise ausgegeben werden. Nach dem GET-Ereignis werden durch das Datenbankprogramm alle PUT-Unterprogramme der in der Hierarchie nachfolgenden Tabellen ausgeführt, sofern für diese Tabellen im Report eine GET-Verarbeitung programmiert wurde. Beim Aufruf der Unterprogramme werden auch Abhängigkeiten über mehrere Ebenen hinweg berücksichtigt. Wertet ein Report nur eine mehrere Hierarchieebenen unter der Wurzel stehende Tabelle aus, werden trotzdem alle Tabellen zwischen der Wurzel und der auszuwertenden Tabelle gelesen.

In der beschriebenen Form wird der Zeitpunkt GET für eine übergeordnete Tabelle immer vor dem GET-Zeitpunkt der untergeordneten Tabellen aufgerufen. Mit dem Zusatz

```
GET table LATE.
```

werden GET-Zeitpunkte gekennzeichnet, die erst nach Bearbeitung aller untergeordneten Tabellen prozessiert werden. Damit können Auswertungen bezüglich der untergeordneten Tabelle (z.B. Aufsummieren von Feldwerten) in die Liste geschrieben werden. Für jede Tabelle darf es in einem Report nur einen GET- und einen GET LATE-Zeitpunkt geben.

3.6.4 Ein praktisches Beispiel

Das folgende Beispiel orientiert sich an den praktischen Erfordernissen der Programmentwicklung. Mitunter ist es wichtig zu wissen, wer wann welches Entwicklungsobjekt bearbeitet hat. Unter der Voraussetzung, dass es sich nicht um lokale private Objekte handelt, gibt die Tabelle E071 Auskunft darüber, in welchen Aufgaben und Aufträgen ein Objekt enthalten ist. Den Bearbeiter eines Auftrages oder einer Aufgabe finden Sie in der Tabelle E070. Dieses Beispiel soll daher zu einem in der Tabelle TADIR enthaltenen Objekt alle Aufgaben und Aufträge sowie deren Bearbeiter auflisten, indem es neben der Tabelle TADIR die Tabellen E070 und E071 auswertet. Dazu soll eine logische Datenbank geschaffen werden.

Die Bearbeitung einer logischen Datenbank wird vom CASE-Menü aus über die Menüfunktion ENTWICKLUNG | PROGRAMMIERUMFELD | LOGICAL DATABASE BUILDER aufgerufen. Die Funktion im Kontextmenü des Object Navigator lautet ANLEGEN | WEITERE | LOGISCHE DATENBANK. Alternativ kann natürlich auch der Transaktionscode SE36 benutzt werden.

Im ersten Dynpro dieser Transaktion (siehe Abbildung 3.64) ist der Name der logischen Datenbank anzugeben. In älteren Versionen der R/3-Software standen dazu nur drei Stellen zur Verfügung. Diese Beschränkung ist inzwischen aufgehoben.

Abbildung 3.64 **© SAP AG**
Startbild der Transaktion SE36

Nach Eintragen des Namens betätigen Sie die ANLEGEN-Schaltfläche. Wie für alle anderen Elemente der Entwicklungsumgebung ist auch für die logische Datenbank ein Kurztext zu erfassen. Dies erfolgt in einem Popup gemäß Abbildung 3.65. Auch dieses Popup wird mit der ANLEGEN-Drucktaste beendet.

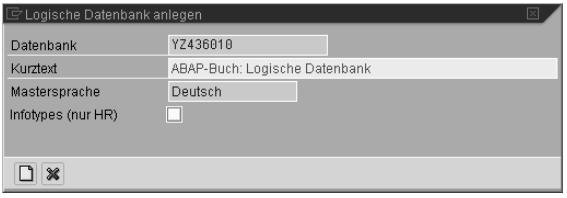

Abbildung 3.65 **© SAP AG**
Kurztext für logische Datenbank festlegen

Zu jeder logischen Datenbank gehört eine Tabellen-Struktur, die genau einen Wurzelknoten besitzt. Die Bearbeitung einer existierenden Struktur erfolgt mit einem dafür vorgesehenen Editor. Der Wurzelknoten für eine neuen Datenbank wird allerdings in einem weiteren Popup (Abbildung 3.66) definiert.

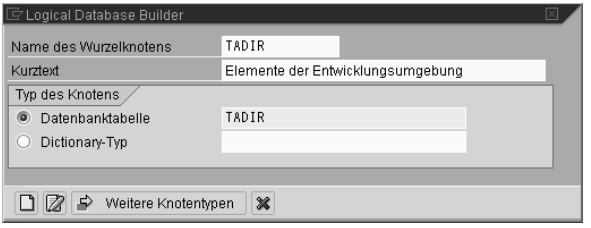

Abbildung 3.66 **© SAP AG**
Wurzelknoten für Tabellenhierarchie definieren

In diesem Popup tragen Sie den Namen des Knotens und einen Kurztext zu dessen Beschreibung ein. Es empfiehlt sich, hier die Namen der beteiligten Tabellen zu benutzen. Außerdem ist natürlich die auszuwertende Tabelle anzugeben.

Nach Bestätigung der Eingabe mit der ANLEGEN-Schaltfläche gelangen Sie schließlich zur eigentlichen Pflegeoberfläche für logische Datenbanken (Abbildung 3.67). Zu diesem Dynpro gelangen Sie auch, wenn Sie im Startbild der Transaktion SE36 den Namen einer existierenden logischen Datenbank eintragen, im Rahmen TEILOBJEKTE das Auswahlfeld für STRUKTUR markieren und die ÄNDERN-Drucktaste betätigen.

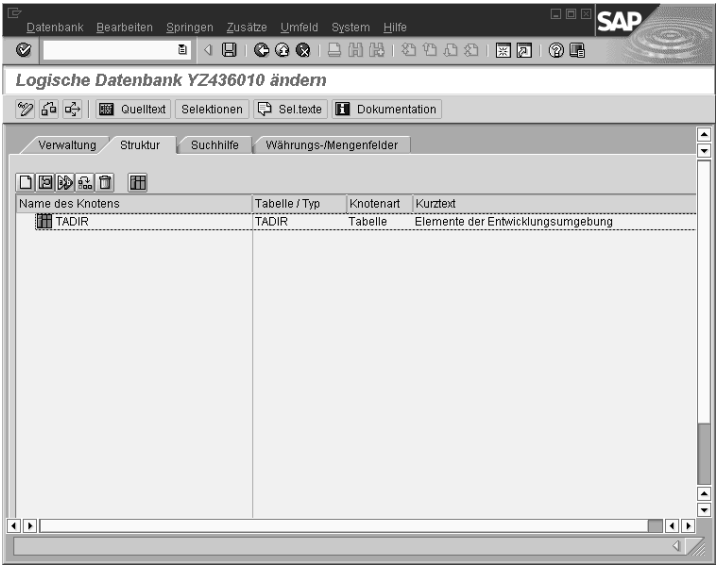

Abbildung 3.67 **© SAP AG**
Pflegeoberfläche für die Struktur einer logischen Datenbank

Innerhalb des Strukturpflegebildes legen Sie nun zwei weitere Knoten in der Tabellenhierarchie an. Dazu stellen Sie den Cursor auf den Knoten, unter dem ein neuer Knoten eingefügt werden soll, und betätigen das Anlegen-Symbol, das Sie unmittelbar über der eigentlichen Arbeitsfläche im Tab-Strip-Bereich finden. Alle Angaben für den neuen Knoten erfassen Sie in einem Popup, das dem zum Anlegen des Wurzelknotens ähnelt (siehe Abbildung 3.68).

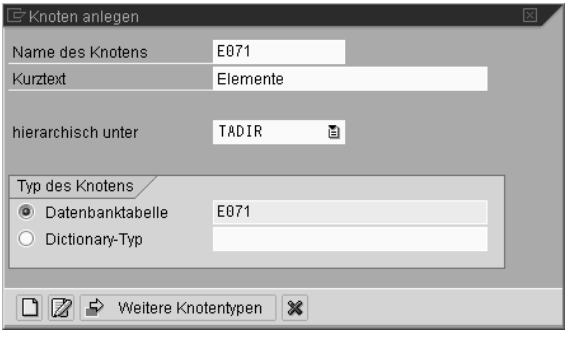

Abbildung 3.68 **© SAP AG**
Anlegen eines neuen Knotens

Neu in diesem Popup ist ein zusätzliches Eingabefeld, in dem Sie den Knoten angeben können, unter dem der neue Knoten eingefügt werden soll. Dieses Feld wird anhand der Cursorposition vorbelegt. Sie können den Inhalt aber jederzeit verändern, wobei sogar eine Eingabehilfe zur Verfügung steht, in der alle momentan existierenden Knoten angeboten werden.

Legen Sie die beiden zusätzlichen Knoten gemäß der in Bild 3.69 ersichtlichen Struktur an.

Nach der Definition der Tabellenstruktur werden die Selektionen angelegt. Sie können dies über eine Menüfunktion oder die Schaltfläche SELEKTIONEN erreichen, die Sie in der Drucktastenzeile finden.

Als Ausgangspunkt für die zu generierenden Programme kann die soeben definierte Tabellenhierarchie dienen. In einem Popup (Abbildung 3.70) müssen Sie durch die Drucktaste JA bestätigen, dass im generierten Programm Selektionen gemäß der Tabellenstruktur vorgesehen werden sollen.

Ein darauf folgendes Popup (Abbildung 3.71) ermöglicht Ihnen, zu den Selektionsfeldern auch noch Suchhilfen vorzusehen. Dies ist für das aktuelle Beispiel nicht notwendig, Sie können die Anfrage daher durch die NEIN-Taste beenden.

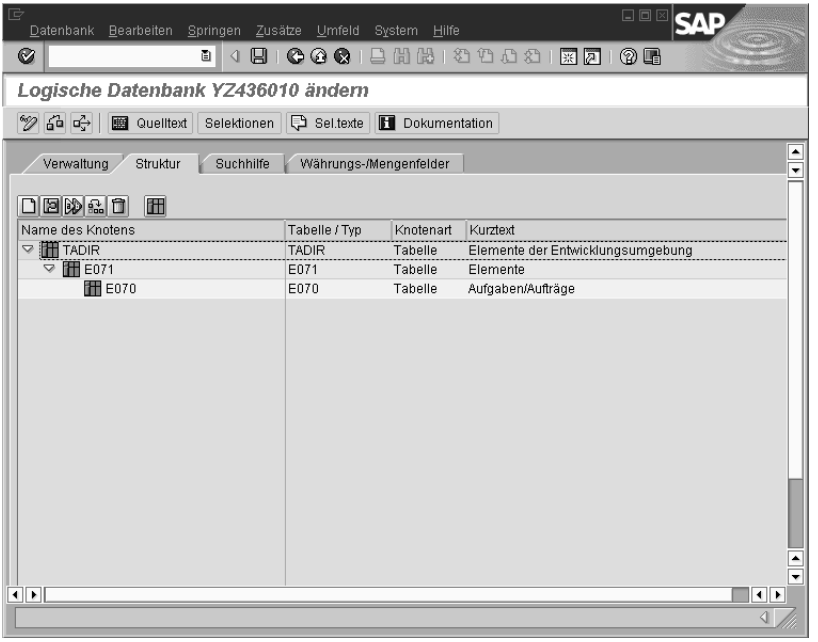

Abbildung 3.69 **© SAP AG**
Endgültige Struktur der logischen Datenbank

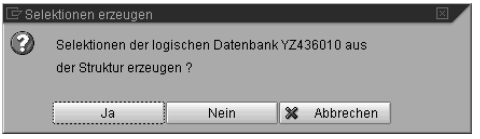

Abbildung 3.70 **© SAP AG**
Sicherheitsabfrage zum Generieren der Selektionen

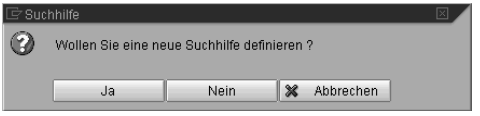

Abbildung 3.71 **© SAP AG**
Entscheidung zum Anlegen von Suchhilfen

Nach der Abfrage zur Suchhilfe erscheint ein letztes Popup (Abbildung 3.72), in dem Sie die Tabellen auswählen können, für die freie Abgrenzungen oder Feldselektionen generiert werden sollen. Auf diese Elemente soll zunächst verzichtet werden. Sie können das Popup ohne Veränderung durch betätigen der ÜBERNEHMEN-Drucktaste beenden.

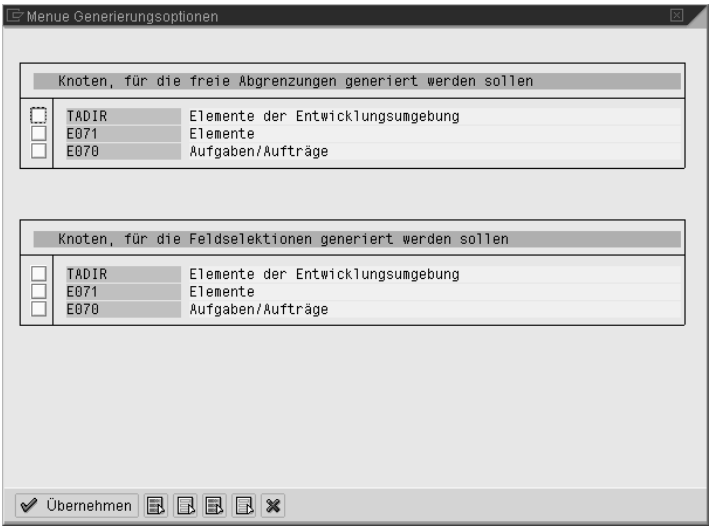

Abbildung 3.72 **© SAP AG**
Auswahl der Tabellen für freie Abgrenzungen

Nun erscheint der Editor mit dem generierten Quelltext. Im folgenden Listing wurden einige Kommentarzeilen entfernt.

```
*---------------------------------------------------------------------*
* INCLUDE DBYZ436010SEL
* It will be automatically included into the
* database program.
*---------------------------------------------------------------------*
* SELECT-OPTIONS :
*                  ?  FOR TADIR-PGMID,
*                  ?  FOR TADIR-OBJECT,
*                  ?  FOR TADIR-OBJ_NAME.


* Parameter for search pattern selection (Type SY-LDB_SP):
* PARAMETERS p_sp AS SEARCH PATTERN FOR TABLE TADIR.

* SELECT-OPTIONS :
*                  ?  FOR E071-TRKORR,
*                  ?  FOR E071-AS4POS.

* SELECT-OPTIONS :  ?  FOR E070-TRKORR.


* Enable DYNAMIC SELECTIONS for selected nodes :

* Enable FIELD SELECTION for selected nodes
```

Der Quelltext enthält für alle Schlüsselfelder der beteiligten Tabellen eine rudimentäre und auskommentierte SELECT-OPTIONS-Anweisung. Für die Tabelle TADIR sollen nun drei Selektionsoptionen programmiert werden. Ändern Sie dazu das Selektions-Include (erste SELECT-OPTIONS-Anweisung) wie folgt:

```
SELECT-OPTIONS :
   s_pgm  FOR TADIR-PGMID,
   s_obj  FOR TADIR-OBJECT,
   s_obn  FOR TADIR-OBJ_NAME.
```

Da es sich bei diesem Programm um ein Include handelt, können Sie weder eine Programmprüfung noch eine Generierung durchführen. Sie würden dabei immer eine Fehlermeldung erhalten. Speichern Sie das Programm ab und kehren Sie zum Struktur-Pflegebild zurück. Rufen Sie dort die Menüfunktion BEARBEITEN | ERZEUGEN | PROGRAMM auf, um das eigentliche Datenbankprogramm zu erzeugen. Diese Funktion ist nur verfügbar, wenn der Ändern-Modus aktiv ist. Bevor das Programm wirklich erzeugt wird, ist eine Bestätigung erforderlich.

Sie gelangen in den Quelltext, der in den jüngeren Versionen der R/3-Software nur INCLUDE-Anweisungen enthält. Im nachfolgenden Listing wurden der besseren Übersicht wegen wieder alle nicht unbedingt notwendigen Kommentare entfernt.

```
include DBYZ436010TOP . " header
include DBYZ436010NXXX . " all system routines
* include DBYZ436010F001 . " user defined include
```

Die wichtigen Elemente sind im zweiten Include enthalten, dessen Name auf NXXX endet. Ein erneuter Doppelklick auf diesen Programmnamen führt zu folgendem, wiederum nur aus Includes bestehendem Quelltext:

```
include DBYZ436010N001 . " Node TADIR
include DBYZ436010N002 . " Node E071
include DBYZ436010N003 . " Node E070
include DBYZ436010FXXX . " init, PBO, PAI
include DBYZ436010SXXX . " search help
```

Im Gegensatz zu früheren Versionen der SAP-Software, bei denen alle SELECT-Anweisungen in einer Datei enthalten waren, wird für jede beteiligte Tabelle eine eigene Include-Datei generiert. Welche Datei in welchem Include behandelt wird, erschließt sich nicht aus dem Dateinamen, sondern nur aus dem Kommentar am Ende jeder Zeile. Wechseln Sie mit einem Doppelklick auf den Programmnamen DBYZ436010N001 in diese Datei. Sie finden dort unter anderem ein Unterprogramm mit einer auskommentierten SELECT-Anweisung:

```
FORM PUT_TADIR.

* SELECT * FROM TADIR
*   INTO TADIR
```

```
*   INTO TABLE ? (choose one!)
*   WHERE PGMID = ?
*     AND OBJECT = ?
*     AND OBJ_NAME = ?.

    PUT TADIR.

* ENDSELECT.
ENDFORM.                                "PUT_TADIR
```

Diese Anweisung ist syntaktisch noch nicht korrekt. Sie kann durch Entfernen einiger Kommentarzeichen und Modifizieren einiger Zeilen aktiviert werden:

```
SELECT * FROM TADIR
*   INTO TADIR
*   INTO TABLE ? (choose one!)
   WHERE PGMID IN s_pgm
     AND OBJECT IN s_obj
     AND OBJ_NAME IN s_obn.

    PUT TADIR.

 ENDSELECT.
```

Die anderen beiden Includes müssen ebenfalls bearbeitet werden. Da die Tabellen `TADIR` und `E071` nicht über gemeinsame Schlüsselfelder verknüpft sind, müssen Sie die für die Tabelle `E071` generierte `WHERE`-Klausel verwerfen und dafür die nachfolgend aufgeführte `SELECT`-Anweisung einfügen.

```
FORM put_e071.
  SELECT * FROM e071
*   INTO E071
*   INTO TABLE ? (choose one!)
    WHERE pgmid = tadir-pgmid
      AND object = tadir-object
      AND obj_name = tadir-obj_name.
    PUT e071.
  ENDSELECT.
ENDFORM.                                „PUT_
```

Die Verknüpfung von `E070` und `E071` erfolgt über ein gemeinsames Schlüsselfeld, so dass die generierte `WHERE`-Klausel übernommen werden kann.

```
FORM put_e070.
  SELECT * FROM e070
*   INTO E070
*   INTO TABLE ? (choose one!)
    WHERE trkorr = e071-trkorr.
    PUT e070.
  ENDSELECT.
ENDFORM.                                "PUT_
```

Nach der Überarbeitung der Selektions-Unterprogramme kehren Sie zur ersten Programmdatei des Datenbankprogramms zurück und aktivieren von dort aus alle bearbeiteten Programme. Es handelt sich dabei um die drei NXXX-Includes, das Selektions-Include und das Rahmenprogramm SAPDBYZ334010. Nach erfolgreicher Aktivierung können Sie wieder zum Strukturpflegebild zurückkehren. Prinzipiell ist die logische Datenbank in diesem Zustand bereits verwendbar. Allerdings sollten noch die Selektionstexte gepflegt werden, um eine einfachere Bedienbarkeit zu gewährleisten. Betätigen Sie dazu die Schaltfläche SELEKTIONSTEXTE. In der bereits bekannten Pflegetransaktion für die Textsymbole (Abbildung 3.73) können Sie nun Selektionstexte nach Ihrer Wahl eintragen. Auch diese Texte müssen aktiviert werden, bevor sie wirksam werden können.

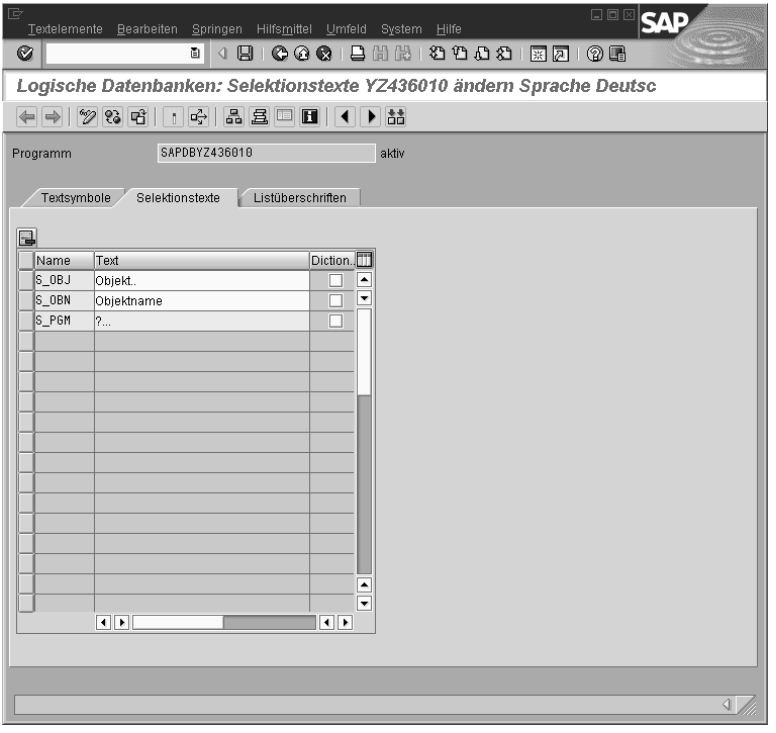

Abbildung 3.73 **© SAP AG**
Selektionstexte pflegen

Zur Sicherheit können Sie abschließend noch eine Prüfung der logischen Datenbank durchführen. Rufen Sie dazu im Strukturpflegebild die Menüfunktion DATENBANK | PRÜFEN auf oder betätigen Sie die DRUCKTASTE PRÜFEN. Ein Popup (Abbildung 3.74) informiert Sie über den Zustand der logischen Datenbank.

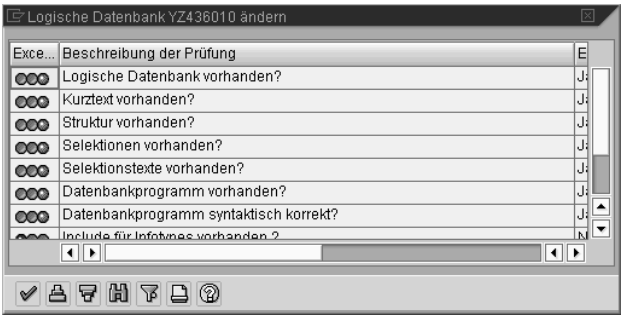

Abbildung 3.74 **© SAP AG**
Prüfergebnis

Nun kann der eigentliche Auswertungsreport erstellt werden. Erzeugen Sie dazu ein neues Programm (`YZ436020`). Tragen Sie in den Attributen des Programms den Namen der logischen Datenbank im entsprechenden Eingabefeld ein. Abbildung 3.75 zeigt das Pflegebild. Beachten Sie dabei, dass nicht der Name des Datenbankprogramms, sondern der Name der logischen Datenbank einzutragen ist.

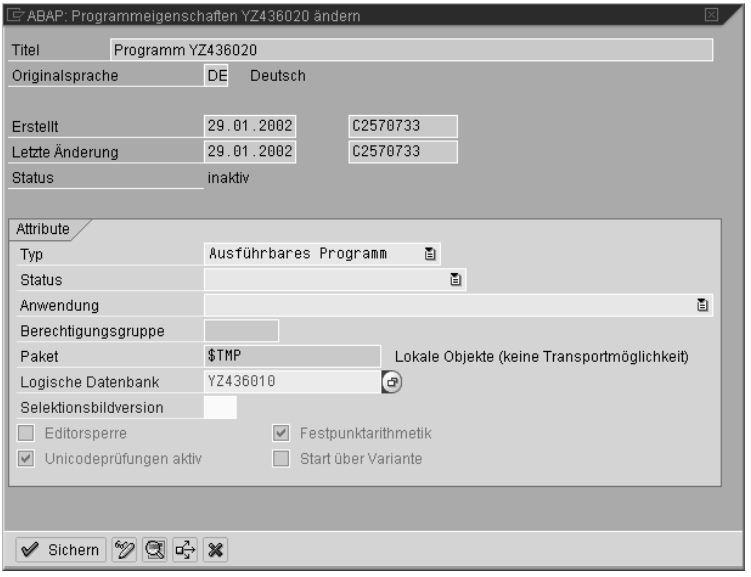

Abbildung 3.75 **© SAP AG**
Attribute des Auswertungsreports

Der Auswertungsreport selbst ist sehr kurz. Das folgende Listing zeigt den Quelltext.

```
REPORT yz436020.
TABLES: tadir,
        e070.

START-OF-SELECTION.

  GET tadir.
    FORMAT COLOR 1.
    WRITE: / tadir-pgmid, tadir-object, tadir-obj_name.

  GET e070.
    WRITE: / e070-trkorr, e070-as4user, e070-as4date.
```

Im Programm existieren zwei GET-Anweisungen. Die erste wird immer dann ausgelöst, wenn von der logischen Datenbank ein Satz in der Tabelle TADIR gelesen wurde. Innerhalb des folgenden Anweisungsblocks wird dieser Datensatz benutzt, um eine Zwischenüberschrift auszugeben. Die zweite GET-Anweisung gehört zum letzten Knoten der Tabellenhierarchie. Der zugehörigen Tabelle können Sie den Namen des Anwenders entnehmen, der dieses Objekt modifiziert hat.

Bei allem Komfort besitzt die Anwendung natürlich eine wesentliche Einschränkung. Die zur Verfügung stehenden Selektionskriterien und die Verknüpfungen der einzelnen Tabellen sind fest vorgegeben. Aber gerade bei der Auswahl der zu analysierenden Objekte aus der Tabelle TADIR sind noch mehr Möglichkeiten denkbar. Statt jedes Selektionskriterium separat zu programmieren, können Sie die bereits erwähnten freien Abgrenzungen verwenden. Dadurch stehen auf einen Schlag und mit relativ wenig Programmierarbeit alle Tabellenfelder einer oder mehrerer Tabellen als potenzielle Selektionsfelder zur Verfügung. Sofern Sie freie Abgrenzungen nicht schon beim Anlegen der logischen Datenbank berücksichtigen, müssen Sie die erforderlichen Anweisungen manuell nachtragen.

In diesem Teil des Beispiels soll eine freie Abgrenzung für die Tabelle TADIR definiert werden. Editieren Sie dazu zunächst das Selektions-Include. Fügen Sie am Ende des Quelltextes die folgende Anweisung ein:

```
* Enable DYNAMIC SELECTIONS for selected nodes :
 SELECTION-SCREEN DYNAMIC SELECTIONS
   FOR TABLE tadir.
```

Mit dieser Anweisung legen Sie fest, dass die Tabelle TADIR für freie Abgrenzungen vorgesehen ist. Infolge dieser Anweisung legt das System zur Laufzeit automatisch eine interne Tabelle dyn_sel_clauses an, die in einem ebenfalls automatisch generierten Dynpro mit den Selektionsanweisungen gefüllt werden kann. Der Inhalt dieser Tabelle muss im Datenbankprogramm ausgewertet werden. Dazu muss das Unterprogramm PUT_TADIR ebenfalls modifiziert werden.

```
FORM put_tadir.
  DATA il_cl TYPE rsds_where.
  READ TABLE dyn_sel-clauses
    WITH KEY 'TADIR'
    INTO il_cl.

  SELECT * FROM tadir
*   INTO TADIR
*   INTO TABLE ? (choose one!)
    WHERE pgmid IN s_pgm
      AND object IN s_obj
      AND obj_name IN s_obn
      AND (il_cl-where_tab).
    PUT tadir.

  ENDSELECT.
ENDFORM.                                    "PUT_
```

Zunächst wird aus der internen Tabelle mit den Selektionsanweisungen der Teil mit den Anweisungen für die Tabelle `TADIR` herausgelöst und in eine zweite interne Tabelle `il_cl` übertragen. Diese Tabelle wird dann zusätzlich zu den bereits vorhandenen Selektionskriterien als weiterer Bestandteil in die `WHERE`-Klausel übernommen. Damit sind die Programmänderungen bereits abgeschlossen. Sie können das Programm nun testen.

Aufgaben

❶ Erstellen und testen Sie das Programm.

❷ Fügen Sie eine neue Selektion für den Datumsbereich hinzu und werten Sie diese Selektion beim Zugriff auf die Tabelle `E070` aus.

❸ Erstellen Sie eine freie Abgrenzung für die Tabelle `E070`.

3.7 Dialoganwendungen

Die Funktionalität der Reports reicht nicht aus, um vollwertige interaktive Anwendungen zu erstellen. Für diesen Zweck stellt das SAP-System die dialogorientierten Anwendungen zur Verfügung. Ebenso wie bei Reports ist hier eine Grundstruktur der Anwendung vorgegeben, der sich die verfügbaren ABAP-Kommandos unterordnen. Dialogorientierte Anwendungen beruhen im Wesentlichen auf dem Einsatz von Bildschirmmasken, die im R/3-System *Dynpros* genannt werden. Das Grundkonzept für Dialoganwendungen wird daher auch als *Dynpro-Konzept* bezeichnet.

Einige Grundzüge dieses Konzepts wurden bereits im Abschnitt über den Selektionsbildschirm der Reports angedeutet. Selektionsbildschirme greifen verschiedene Mechanismen der Dynpros auf, allerdings ist die programmtechni-

sche Realisierung im Falle der echten Dialoganwendungen eine völlig andere. Sie ist komplizierter zu programmieren, ermöglicht dafür aber auch eine wesentlich umfangreichere Funktionalität der Anwendung.

3.7.1 Ein einfacher Einstieg

Dialoganwendungen stellen dem Anwender die unter modernen grafischen Oberflächen verfügbaren Eingabe- und Bedienelemente zur Verfügung. Diese Elemente können in zwei große Gruppen eingeteilt werden. Zur ersten gehören Elemente, die eine Bearbeitung von Daten ermöglichen, also Editierfelder und Ähnliches. Diese Elemente werden in das Dynpro aufgenommen und in dessen Arbeitsfläche dargestellt. Die zweite Gruppe bilden alle Elemente, mit denen der Anwender eine Funktion auslösen kann. Dies sind z. B. Menüs, Drucktasten oder Symbole. Innerhalb von ABAP-Anwendungen werden diese Elemente unter dem leider etwas missverständlichen Begriff *Oberfläche* zusammengefasst. Dieser Begriff darf nicht mit der gesamten Bedienoberfläche, also dem Bildschirmfenster, gleichgesetzt werden. Er umfasst lediglich die genannten Elemente. Die Objekte der Arbeitsfläche des Dynpros gehören in der SAP-Begriffswelt nicht zur Oberfläche. Eine Oberfläche besteht aus einem oder mehreren so genannten *Status*. Da der Begriff Status dazu beiträgt, Missverständnisse zu vermeiden, soll er im Folgenden vorrangig benutzt werden. Für die Bildschirmmaske, die eigentliche Arbeitsfläche des Dynpros, hat sich auch der Begriff *Fullscreen* eingebürgert. Die Bearbeitung der Oberfläche bzw. eines Status wurde bereits im Abschnitt 3.3.3 erläutert.

Alle Dialogelemente müssen mit Elementen des Programms verbunden werden, um die gewünschte Funktionalität der Anwendung herzustellen. Bei Auswahl eines Menüeintrags ist die gewünschte Funktionalität auszuführen, die Werte aus Eingabeelementen sind in Datenfelder zu übertragen usw. Neben dem Dynpro und der Oberfläche ist also zusätzlich noch Programmcode erforderlich. Während ein Report im einfachsten Fall nur aus einer einzigen Programmdatei besteht, die zudem direkt abgearbeitet werden kann, umfasst eine Dialoganwendung folgende Elemente:

- eine oder mehrere Eingabemasken (Dynpros);
- einen oder mehrere Status (Oberfläche);
- ABAP-Anweisungen.

Zwischen der Ausführung von Reports und von Dialoganwendungen besteht ein wesentlicher Unterschied, der Umsteigern von herkömmlichen prozeduralen Programmiersprachen die Einarbeitung erschwert. Ein Report verfügt über einen Programmabschnitt, der entfernt mit dem Hauptprogramm herkömmlicher Sprachen verglichen werden kann. Üblicherweise ist dies der Abschnitt, in dem Daten gelesen und ausgegeben werden, also der Abschnitt nach dem Zeitpunkt `START-OF-SELECTION`. In diesem Abschnitt finden die eigentlichen Programmaktivitäten statt. Alle anderen Zeitpunkte ähneln Unterprogrammen,

auch wenn sie nicht direkt aufgerufen, sondern durch Ereignisse und Zeitpunktanweisungen ausgeführt werden. Sie erfüllen Hilfsaufgaben. Es bestehen also gewisse Ähnlichkeiten zu Programmen, die in prozeduralen Sprachen geschrieben wurden. Diese Ähnlichkeiten wurden im vorangegangenen Abschnitt bewusst ausgenutzt, um den Einstieg in die Programmiersprache ABAP zu erleichtern.

Eine Dialoganwendung verfügt nicht über einen solchen Programmkern. Im Quelltext ist nur eine Anzahl so genannter Module, eine Art Unterprogramm, zu finden. Die eigentliche Programmdatei trägt daher auch den Namen *Modul-Pool*. Der Modul-Pool kann entfernt mit einer Bibliothek verglichen werden, die eine Reihe von Hilfsroutinen zur Verfügung stellt. Die sequenzielle Abarbeitung der Anweisungen im Modul-Pool ist weder möglich noch sinnvoll. Trotzdem ist der Name des Modul-Pools auch der Name der Anwendung.

Die entscheidende Rolle bei der Steuerung des Programmablaufs in Dialoganwendungen spielen die Dynpros. Bei der Abarbeitung einer dialogorientierten Anwendung werden nicht Programme oder Unterprogramme, sondern Dynpros aufgerufen. Erst diese Dynpros führen zu exakt bestimmten Zeitpunkten einzelne Module aus dem Modul-Pool aus. Der Aufruf von Modulen erfolgt in der so genannten *Ablauflogik*, einem speziellen Quelltext, über den jedes Dynpro verfügt und der jeweils individuell programmiert werden kann. Die Ablauflogik wird zusammen mit dem Dynpro abgespeichert, sie ist nicht direkt im Modul-Pool zu finden.

Dynpro und Modul-Pool sind untrennbar miteinander verbunden. Der Modul-Pool wird von mehreren Dynpros einer Anwendung benutzt. Dynpros werden deshalb durch den Namen des Modul-Pools und durch ihre Nummer identifiziert.

Dialoganwendungen müssen mit einer Transaktion verbunden werden, über deren Namen die Anwendung aufgerufen wird. Dieser Transaktion wird der Name des Modul-Pools und die Nummer des ersten auszuführenden Dynpros übergeben.

Der folgende Abschnitt soll Kenntnisse zu dialogorientierten Anwendungen vermitteln. Die ersten beiden Unterabschnitte werden daher die Bearbeitung einer sehr einfachen Dialoganwendung beschreiben. Der nächste Unterabschnitt erläutert die Eigenschaften und die verschiedenen Bestandteile eines Dynpros detaillierter. Anschließend werden spezielle Programmiertechniken vorgestellt, die im Zusammenhang mit Dialoganwendungen von Interesse sind.

Die Erstellung einer einfachen dialogorientierten Anwendung ist erheblich aufwändiger als das Schreiben des „Hello-World"-Programms. Ein Test der Anwendung ist erst möglich, nachdem mehrere Bestandteile der Anwendung angelegt wurden. Zusammenhänge erschließen sich erst, wenn die Menge der vermittelten Fakten ein bestimmtes Maß überschreitet. Nachfolgend sollen daher zunächst die wichtigsten Elemente einer Dialoganwendung erzeugt werden, wobei deren Funktionalität auf das unbedingt notwendige Maß beschränkt wird. Beachten Sie beim Nachvollziehen der Beispiele an Ihrem System bitte, dass alle Elemente als lokale private Objekte angelegt werden sollten.

Die erste Dialoganwendung soll einen einfachen Rechner realisieren. Im ersten Dynpro erfassen Sie zwei Operanden und wählen die auszuführende Rechenoperation. Das zweite Dynpro zeigt Ihnen das Ergebnis an.

Da eine Dialoganwendung aus verschiedenen Objekten besteht, die wiederum mehrere Bestandteile haben können, sind zur Erstellung und zum Test einer Dialoganwendung folgende Arbeitsschritte erforderlich:

- Anlegen eines Modul-Pools;
- Deklaration der globalen Daten;
- Erzeugen der Status;
- Erzeugen der Dynpros (Maske und Ablauflogik);
- Programmieren der Module;
- Erzeugen einer Transaktion.

Anlegen eines Modul-Pools

Neue Dynpros müssen einem Modul-Pool zugeordnet werden. Die Programmierung einer Dialoganwendung beginnt daher mit dem Anlegen eines Modul-Pools. Starten Sie dazu zunächst die Entwicklungsumgebung.

Im Object Navigator rufen Sie wieder die Funktion zum Anlegen eines neuen Programms auf. Im Popup tragen Sie nun aber den Namen `SAPMY410` ein. Die Zeichenfolge `SAPM` am Beginn des Programmnamens ist wichtig, da das System an dieser Zeichenfolge erkennt, dass es sich um einen Modul-Pool handelt. Außerdem bleibt das Auswahlfeld zum Anlegen eines Top Includes diesmal aktiviert (siehe Abbildung 3.76).

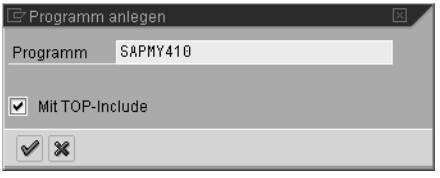

Abbildung 3.76 **© SAP AG**
Abfrage Programmname

Im nächsten Schritt erfragt das System den Namen des Top Includes. Da dieser üblicherweise den SAP-internen Namenskonventionen genügt, kann das System einen Vorschlag erstellen. Dieser Vorschlag sollte immer übernommen werden (Abbildung 3.77).

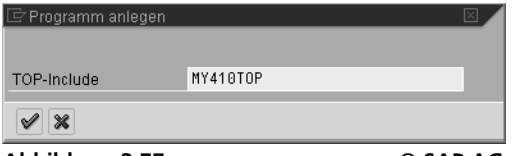

Abbildung 3.77 **© SAP AG**
Abfrage Include-Name

Nun erscheint ein drittes Popup (Abbildung 3.78) zur Pflege der Eigenschaften des Programms. Auf Grund des gewählten Programmnamens wird als Programmtyp bereits Modul-Pool vorgeschlagen.

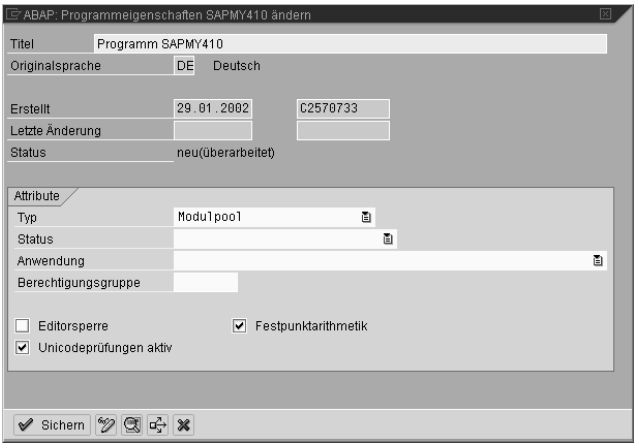

Abbildung 3.78 **© SAP AG**
Programmattribute festlegen

Ebenso wie bei Reports sind nun noch eine Kurzbeschreibung und das Anwendungskürzel zu erfassen. Die Attribute werden abgespeichert und der Programmeditor aufgerufen. Es bietet sich das in Abbildung 3.79 sichtbare Bild.

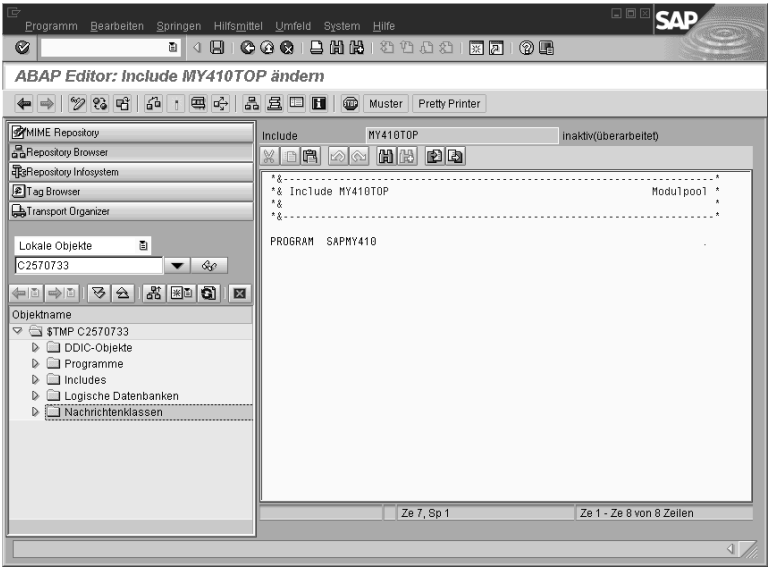

Abbildung 3.79 **© SAP AG**
Editor mit Top Include

Der Editor springt sofort in das Top Include und nicht in den eigentlichen Modul-Pool. Es fällt auf, dass die PROGRAM-Anweisung nicht im Hauptprogramm, sondern im Include steht. Der eigentliche Modul-Pool besteht nur aus einigen INCLUDE-Anweisungen. An dieser Stelle können Sie im Quelltext die PROGRAM-Anweisung ergänzen und folgende Felddeklarationen eintragen:

```
PROGRAM sapmy410.
DATA:
  fcode LIKE sy-ucomm,
  value1 TYPE p,
  value2 TYPE p,
  result TYPE p,

  op.
```

Speichern Sie die Änderungen ab. Sie können in der Objektliste des Object Navigator nun den Eintrag für die soeben erzeugte Anwendung suchen und deren Sub-Elemente durch Öffnen des Knotens einblenden. Abbildung 3.80 zeigt die noch recht bescheidene Objektliste nach Erstellen des Modul-Pools. Die Elemente werden entsprechend ihres Typs (Programme, Module, Unterprogramme, Dictionary-Objekte usw.) zu Gruppen zusammengefasst.

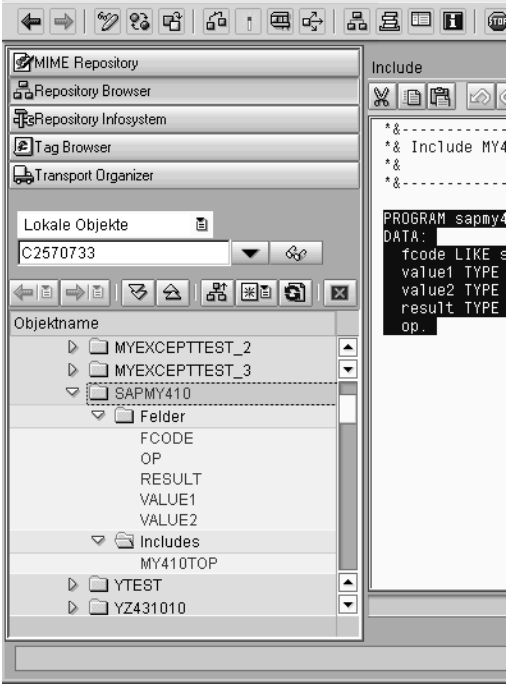

Abbildung 3.80 **© SAP AG**
Objektliste des Browsers

Der Klick auf den Namen einer Gruppe bzw. ein einzelner Mausklick auf das Symbol vor dem Gruppennamen blendet die Elementliste für diese Gruppe ein oder aus. Der Doppelklick auf ein Element startet das Werkzeug für die Bearbeitung dieses Elements und lädt das zu bearbeitende Element in das Werkzeug.

An dieser Stelle sollten Sie den Cursor auf den Programmnamen platzieren und die existierende Anwendung aktivieren (Menüfunktion ENTWICKLUNGSOBJEKT | AKTIVIEREN). Damit erleichtern Sie sich die Bearbeitung des nun anzulegenden Dynpros.

Das erste Dynpro

Nachdem der Modul-Pool erstellt wurde, kann eine erste Version des Dynpros erzeugt werden. Dynpros werden mit dem Screen Painter bearbeitet. Dieses Werkzeug ist von der Workbench aus (nicht vom Object Navigator) mit der Menüfunktion ENTWICKLUNG | OBERFLÄCHE | SCREEN PAINTER erreichbar. Der Transaktionscode für den Direktaufruf lautet SE51. Neben dem direkten Aufruf des Screen Painter kann ein Dynpro auch direkt aus der Objektliste des Object Navigator heraus erzeugt werden.

In der Objektliste sind alle Elemente des aktuellen Programms enthalten. Da bis jetzt nur der Modul-Pool und ein Include erzeugt wurden, ist die Liste relativ kurz. Der Cursor wird auf das Namensfeld des Modul-Pools platziert und die Menüfunktion ANLEGEN | DYNPRO aufgerufen. Es erscheint ein Popup (Abbildung 3.81), in dem die Nummer des anzulegenden Dynpros eingetragen wird. In diesem Beispiel soll dem Dynpro die Nummer 100 zugeordnet werden.

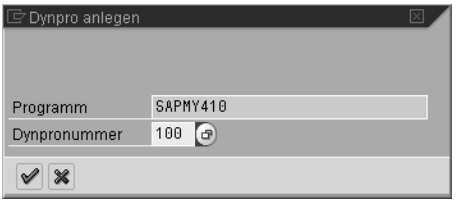

Abbildung 3.81 **© SAP AG**
Anlegen eines neuen Dynpros

Nach Bestätigen des Namens erscheint eine Maske, in der wichtige Attribute für das Dynpro erfasst werden. Abbildung 3.82 zeigt diese Eingabemaske. Wie bei allen anderen bisher behandelten Elementen ist die Erfassung einer Kurzbeschreibung notwendig. Falls nicht automatisch vom System vorgegeben, ist das Auswahlfeld NORMAL für den Dynpro-Typ zu markieren. Alle anderen Felder bleiben unverändert.

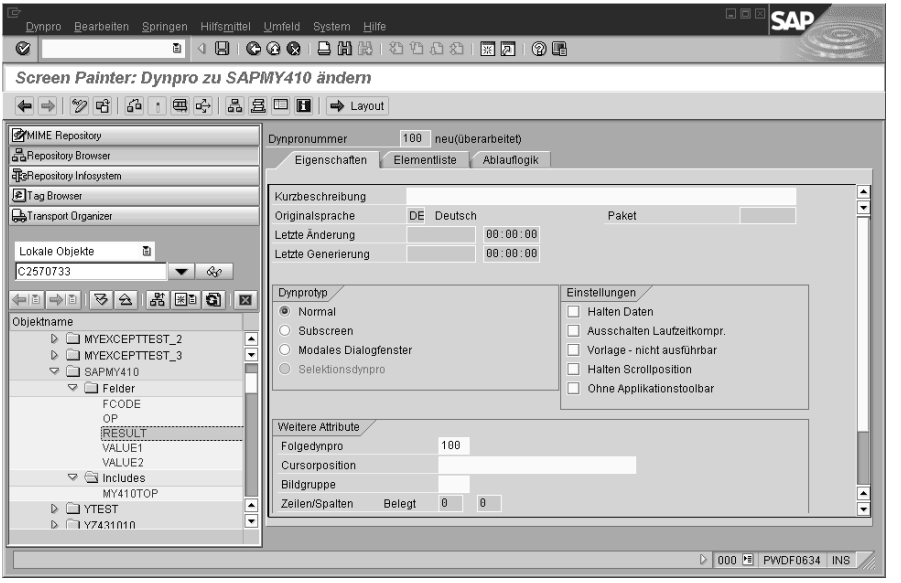

Abbildung 3.82 **© SAP AG**
Attribute-Bildschirm für Dynpros

Die Werte in diesem Dynpro werden abgespeichert. Mit der Drucktaste LAYOUT oder der Menüfunktion SPRINGEN | LAYOUT gelangen Sie nun zur Bearbeitung der visuellen Oberfläche des Dynpros. Für dieses Werkzeug existieren zwei unterschiedliche Implementierungen. Zunächst ist seit jeher ein Layout-Editor verfügbar, der innerhalb des SAPGUI mit Mitteln des interaktiven Reportings realisiert wird. Er wird daher auch als alphanummerischer Editor bezeichnet. Da dieser Editor immer verfügbar ist, soll er in den folgenden Beispielen auch benutzt werden. Die zweite Variante wird durch ein externes, betriebssystembasiertes Programm gebildet. Es nutzt die grafischen Möglichkeiten einer solchen Anwendung aus und trägt daher auch die Bezeichnung *Grafischer Layout-Editor*. Diese Anwendung muss bei der Installation des SAPGUI zusätzlich lokal installiert werden.

Unabhängig von der Art des Editors besteht ein neues Dynpro zunächst aus einem völlig leeren Arbeitsbereich (siehe Abbildung 3.83).

In der hier zur Verfügung gestellten Arbeitsfläche werden nun alle Dynpro-Elemente platziert. Für das Dynpro ist jede lückenlose Zeichenkette im Fullscreen, unabhängig davon, welche Zeichen sie enthält, ein Feld. Ein Feld wird automatisch in die so genannte *Feldliste* des Dynpros aufgenommen. Durch Einträge in der Feldliste werden programminterne Datenfelder mit den Dynpro-Feldern verbunden und damit der Datenaustausch zwischen Programm und Dynpro ermöglicht.

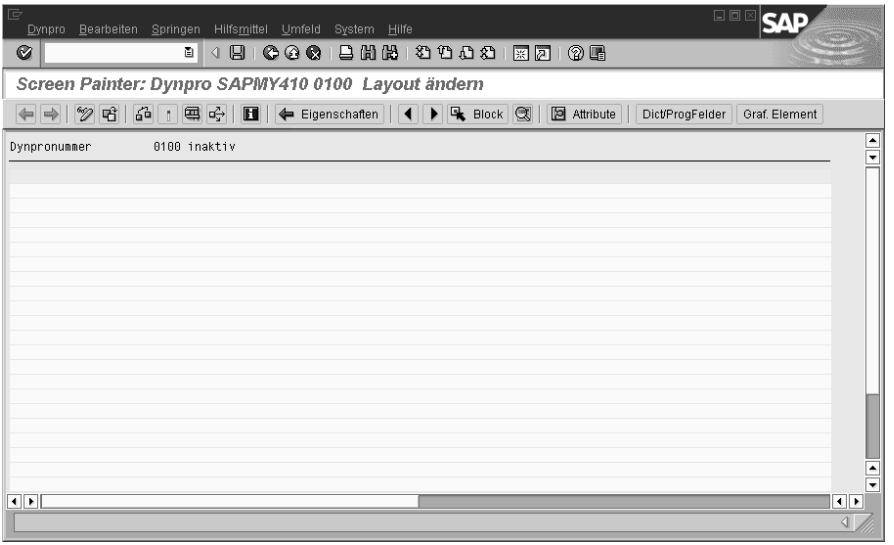

Abbildung 3.83 **© SAP AG**
Fullscreen-Editor

Diverse Attribute bestimmen, wie das Feld später im Dynpro erscheint. Möglich ist beispielsweise die Darstellung als Editierfeld, als reines Anzeigefeld ohne Editierfunktion, als Ankreuz- oder Auswahlfeld oder als einfacher Text. Die verschiedenen Attribute bzw. Erscheinungsformen von Feldern stehen im Mittelpunkt des Abschnitts 3.7.2.

Die Attribute eines Feldes können auf verschiedene Weise gesetzt bzw. verändert werden. Eine der Möglichkeiten besteht in der manuellen Bearbeitung der Feldliste. Ebenfalls möglich ist die Zuweisung von Attributen über die Menüfunktion BEARBEITEN | ATTRIBUTE. Diese ruft für das durch die Cursorposition identifizierte Feld ein Popup auf, in dem alle Attribute des Feldes dargestellt und zum großen Teil auch verändert werden können. Spezielle Bearbeitungsfunktionen, z B. das Umwandeln eines Feldes in ein grafisches Element, wirken sich ebenfalls auf einige Attribute aus. Diese speziellen Attribute können nur durch die jeweilige Bearbeitungsfunktion gesetzt werden. Die letzte Möglichkeit besteht im Zuweisen von Attributen durch ein spezielles Werkzeug, welches das Anlegen von Dynpro-Feldern unterstützt.

Dieses eben erwähnte Werkzeug kann in einem Zug Dynpro-Felder anlegen und diese mit Datenfeldern, die im Modul-Pool bereits deklariert wurden, bzw. mit Tabellenfeldern aus dem Data Dictionary verbinden. Außerdem sorgt es dafür, dass für Eingabefelder mit Dictionary-Bezug weitere Felder mit den zugehörigen Beschreibungen erzeugt werden. Es ist daher nicht üblich, Dynpro-Felder ohne zwingenden Grund durch manuelle Bearbeitung im Fullscreen anzulegen und ihnen dann einzeln Attribute zuzuweisen. Im Allgemeinen wird

das im Folgenden beschriebenen Einfügewerkzeug verwendet. Beachten Sie bitte, dass dieses Werkzeug nur die programminternen Felder in der generierten Fassung eines Programms findet.

Dieses Werkzeug wird mit SPRINGEN | DICT/PROGRAMMFELDER aufgerufen. Bei dieser handelt es sich um ein Popup (Abbildung 3.84), in dem zunächst einige Angaben erfolgen müssen, um die einzufügenden Felder näher zu spezifizieren.

Für die Übernahme programminterner Felder tragen Sie ein Muster, das auf die Namen der einzufügenden Felder passt, in das Eingabefeld TABELLEN-/FELDNAME ein. In diesem Fall kann dies der Stern sein, um alle Felder zu selektieren.

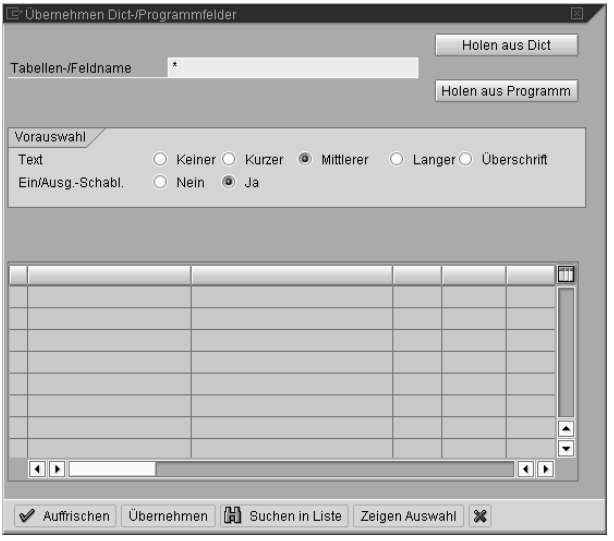

Abbildung 3.84 **© SAP AG**
Übernehmen von Feldern aus dem Programm in das Dynpro

Nach Auslösen der Drucktaste HOLEN AUS PROGRAMM werden alle verfügbaren Felder aufgelistet. Ein Ankreuzfeld vor jedem Feldnamen ermöglicht die Markierung der zu übernehmenden Felder. Die Auswahlliste kann mittels eines Rollbalkens bewegt werden. In der Liste werden die Felder `value1` *und* `value2` durch einen Mausklick im Ankreuzfeld markiert (Abbildung 3.85).

Mit der Drucktaste ÜBERNEHMEN übernehmen Sie die ausgewählten Tabellenfelder ins Dynpro. Das Popup verschwindet, die Arbeitsoberfläche des Fullscreen-Editors wird wieder sichtbar. Dort werden durch einen Doppelklick mit der Maus oder der Funktionstaste F2 die vorher gewählten Felder eingefügt. Der Layout-Editor ermöglicht das Einfügen leider nur linksbündig, unabhängig davon, wo Sie die Schreibmarke platzieren. Abbildung 3.86 zeigt den Fullscreen zu diesem Zeitpunkt. Die Eingabefelder werden symbolisiert durch eine Folge von Unterstrichen.

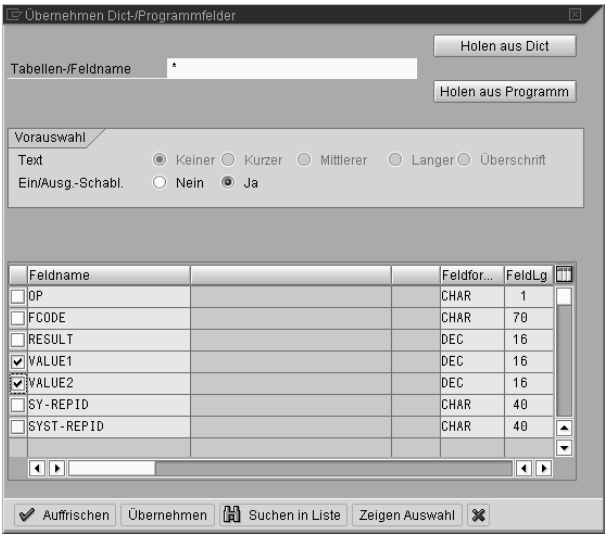

Abbildung 3.85 **© SAP AG**
Auswahl von programminternen Feldern

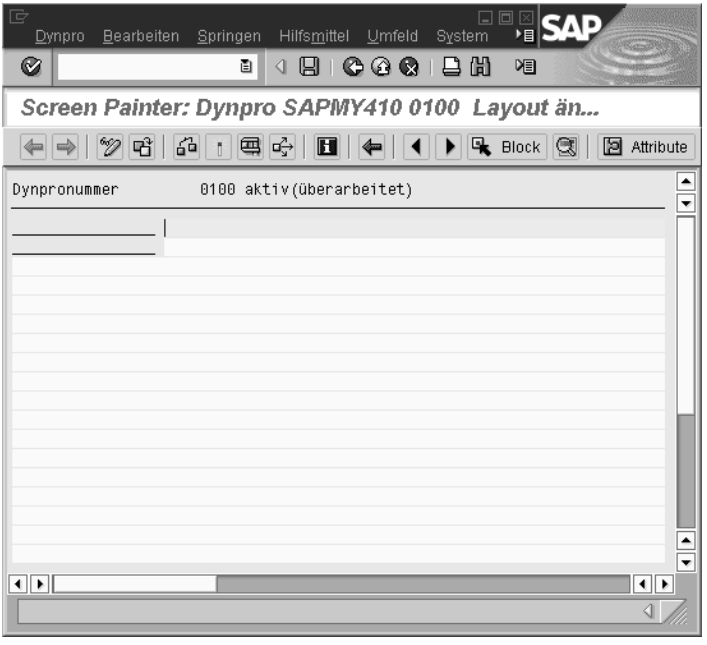

Abbildung 3.86 **© SAP AG**
Fullscreen mit eingefügten Feldern

Um Platz für die Feldbezeichner zu schaffen, müssen Sie die Eingabefelder nach dem Einfügen noch verschieben. Markieren Sie dazu ein Feld durch einen Doppelklick, platzieren Sie den Cursor an der gewünschten Stelle und betätigen Sie die VERSCHIEBEN-Drucktaste in der Drucktastenzeile (alternativ Funktionstaste ⇧ F4 oder Menüfunktion BEARBEITEN | BLOCK VERSCHIEBEN).

Beim Einfügen von programminternen Feldern werden nur die eigentlichen Felder, aber keine Bezeichnungen übernommen. Aus diesem Grund werden derartige programminterne Felder nur selten benutzt. Praktischer ist die Verwendung von Dictionary-Feldern, da diese über eine Bezeichnung verfügen, die automatisch mit ins Dynpro übernommen werden kann. Da für dieses Beispiel keine geeignete Dictionary-Struktur existiert, wird der Einfachheit halber auf programminterne Felder zurückgegriffen. In diesem Fall müssen Sie die Bezeichnungen manuell als Zeichenkette im Dynpro eintragen. Benutzen Sie dabei den Unterstrich statt eines Leerzeichens zum Trennen der Wörter. Bild 3.87 zeigt das entsprechend überarbeitete Dynpro. Speichern Sie das Dynpro vor dem nächsten Bearbeitungsschritt ab.

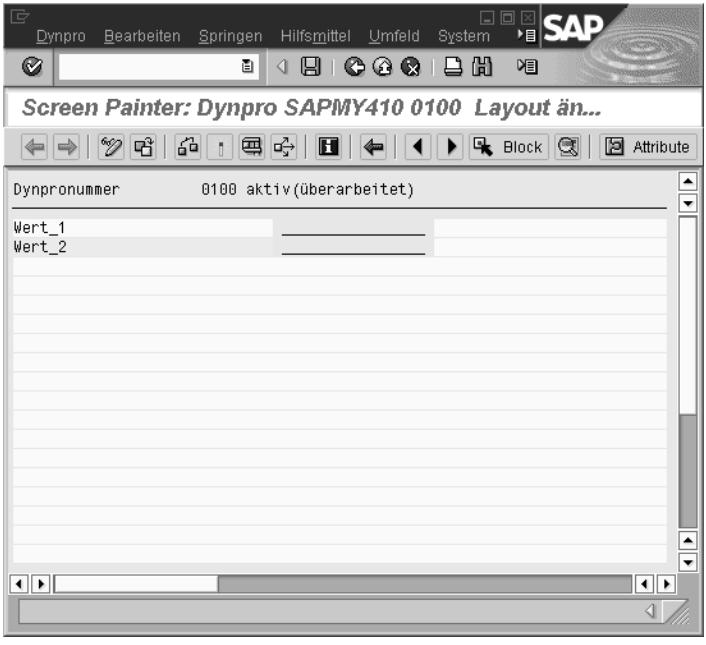

Abbildung 3.87 **© SAP AG**
Dynpro mit Bezeichnungen für Eingabefelder

Nähere Angaben zu den Feldern sind aus der *Elementliste* ersichtlich. In älteren Versionen der R/3-Software wurde noch die Bezeichnung *Feldliste* verwendet. Diese Liste ist von verschiedenen Stellen der Dynpro-Bearbeitung aus mit der Menüfunktion SPRINGEN | ELEMENTLISTE oder einer Drucktaste erreichbar. Ab-

bildung 3.88 zeigt die Elementliste. Für die eben eingefügten Felder (sowohl für die Eingabefelder als auch für die Texte) ist jeweils ein Eintrag vorhanden. Lediglich der letzte Eintrag der Liste ist noch leer. Dieser wird automatisch angelegt. Er ist in jedem Dynpro vorhanden. Der Eintrag verweist auf ein verborgenes Feld des Dynpros, in dem der durch den Anwender ausgelöste Funktionscode bereitgestellt wird. Wenn Sie diesen Funktionscode in Ihrer Anwendung auswerten möchten, so müssen Sie ein programminternes Feld bereitstellen, das zur Aufnahme des Funktionscodes dient. Das ist in diesem Fall das bereits deklarierte Feld `fcode`. Tragen Sie diesen Feldnamen in die letzte Zeile der Elementliste (Abbildung 3.88) ein und speichern Sie nochmals ab.

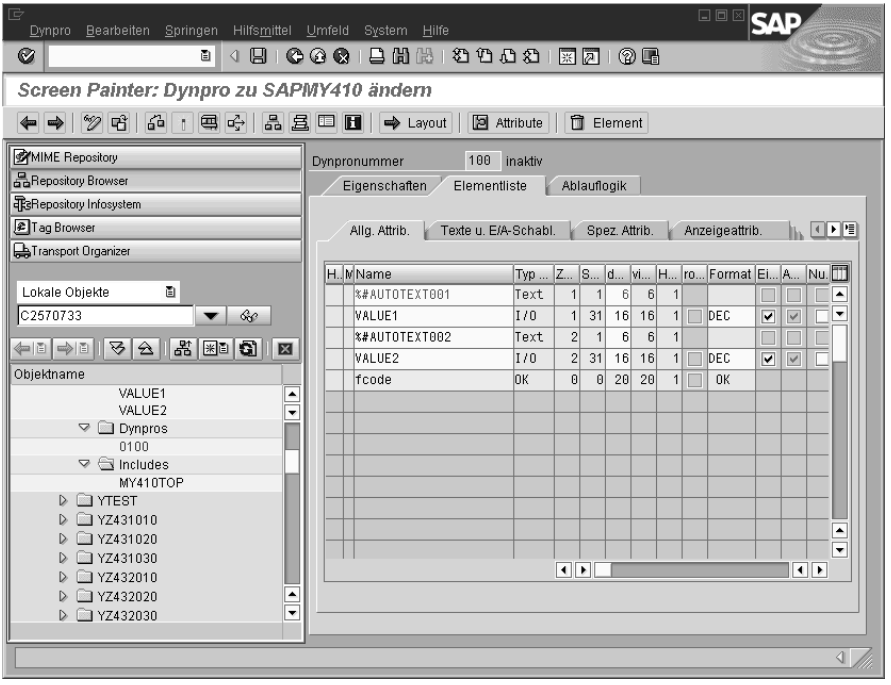

Abbildung 3.88 **© SAP AG**
Feldliste des Dynpros

Nun kann die Programmierung der Module der Ablauflogik erfolgen. Erst dadurch erhält die Anwendung die gewünschte Funktionalität. Um das Dynpro mit Programmcode zu verbinden, werden in der Ablauflogik des Dynpros Module aufgerufen. Diese Module sind im Modul-Pool anzulegen und mit Quelltext zu füllen.

Zur Ablauflogik gelangen Sie durch Betätigen der Registerkarte ABLAUFLOGIK. Die Ablauflogik für ein neues Dynpro besteht zunächst nur aus wenigen Zeilen, die vom System automatisch generiert werden. Der genaue Inhalt der Ablauf-

logik ist von der SAP-Version abhängig. Die beiden zunächst auskommentierten Modulaufrufe werden erst ab Version 3.0 automatisch eingefügt.

```
PROCESS BEFORE OUTPUT.
* MODULE STATUS_0100.
*
PROCESS AFTER INPUT.
* MODULE USER_COMMAND_0100.
```

Bei den beiden Anweisungen `PROCESS BEFORE OUTPUT` und `PROCESS AFTER INPUT` handelt es sich um Zeitpunktanweisungen. Sie werden vor bzw. nach Aufruf des Dynpros aufgerufen. Die Ablauflogik scheint auf den ersten Blick ein normales ABAP-Programm zu sein. Dieser Eindruck täuscht allerdings. Es existieren nur wenige spezielle Anweisungen, die in der Ablauflogik benutzt werden dürfen. Sie unterscheiden sich erheblich von denen normaler ABAP-Programme.

In der Ablauflogik werden nun die Kommentarzeichen für die beiden Modulaufrufe entfernt, so dass sich der folgende Quelltext ergibt:

```
PROCESS BEFORE OUTPUT.
  MODULE STATUS_0100.
*
PROCESS AFTER INPUT.
  MODULE USER_COMMAND_0100.
```

An dieser Stelle wird die Ablauflogik und damit das Dynpro gespeichert und aktiviert. Ohne Aktivierung würden die Modulaufrufe zwar in der Ablauflogik des Dynpros stehen, sie würden bei dessen Abarbeitung allerdings nicht erkannt und daher auch nicht ausgeführt werden. Dieser Fehler ist relativ schwer zu entdecken, da die Anweisungen im Quelltext völlig korrekt erscheinen.

Die beiden Module erfüllen grundlegende Aufgaben. Sie kommen daher in nahezu allen Dynpros vor, mitunter sogar mit fast identischen Namen. Im Modul `STATUS_0100` wird ein Status geladen. Erst dadurch stehen die dort angelegten Menüfunktionen und Funktionstasten im Dynpro zur Verfügung. Das Modul `USER_COMMAND_0100` hingegen wertet die eingegebenen Daten und den Funktionscode aus, um Folgeaktivitäten anzusteuern.

Die Namen von Modulen können bis zu 20 Zeichen lang werden, ansonsten gelten dieselben Empfehlungen wie bei den Namen von Unterprogrammen oder Feldern. Allerdings existieren einige (ungeschriebene) Empfehlungen für die Namensgebung. In einem Modul-Pool befinden sich sehr oft Module für mehrere Dynpros. Zwecks besserer Übersicht und einheitlicher Namensbildung der Module beginnen viele Entwickler den Namen eines Moduls mit einem Begriff, der die Aufgabe des Moduls andeutet, und fügen zum Schluss die Nummer des Dynpros an. Sollten beispielsweise mehrere Dynpros existieren, in denen jeweils ein anderer Status gesetzt wird, werden mehrere `STATUS`-Module erforderlich,

die alle einen eindeutigen Namen erhalten müssen. Durch Verwendung der Dynpro-Nummern werden die Namen der Module eindeutig, ohne an Aussagekraft zu verlieren. Mitunter ist es üblich, ähnliche Funktionen für mehrere Dynpros in einem Modul zusammenzufassen und innerhalb dieses Moduls anhand der Nummer des aktuellen Dynpros, die in der Systemvariablen `SY-DYNNR` enthalten ist, die konkret auszuführenden Funktionen auszuwählen. Die gemeinsame Nutzung von Modulen kann allerdings zu recht komplexen und unübersichtlichen Programmen führen. Auch die dynprobezogene Dokumentation oder die selektive Nutzung von Dynpros aus anderen Programmen heraus wird durch derartige komplexe Module erschwert.

Nachdem in der Ablauflogik die Modulaufrufe eingebaut wurden, muss natürlich noch der dazugehörige Quelltext im Modul-Pool angelegt werden. Da der Editor der Ablauflogik ebenso wie der allgemeine ABAP-Editor über einen recht komfortablen Navigationsmechanismus verfügt, ist es nicht notwendig, den Screen Painter zu verlassen und in den Programmeditor zu wechseln. Ein Doppelklick auf den Namen des Moduls reicht aus, um den ABAP-Editor für den Modul-Pool zu starten. Das System erkennt dabei, dass das zu bearbeitende Modul noch nicht existiert und fragt, ob es angelegt werden soll (Abbildung 3.89).

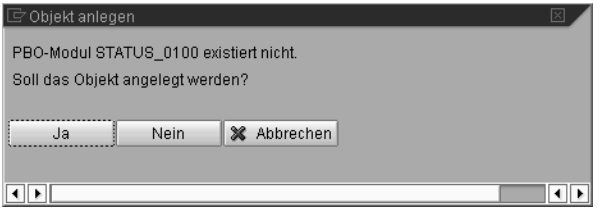

Abbildung 3.89 **© SAP AG**
Abfrage vor Anlegen eines neuen Moduls

In einem weiteren Popup ist der Name der Programmdatei anzugeben, in der das Modul angelegt werden soll. In diesem Popup werden zunächst alle bereits existierenden Include-Dateien des Modul-Pools in einer Liste zur Auswahl angeboten. Außerdem kann in einem Eingabefeld ein neuer Name eingetragen werden. In diesem Feld stellt das System einen Vorschlagswert bereit, der überschrieben werden kann. Die so benannte Include-Datei wird vom System dann automatisch erzeugt und per `INCLUDE`-Anweisung im Modul-Pool eingebunden.

Die Namen der Include-Dateien für Module werden ebenso wie der Name des Top Includes anhand einiger Konventionen vom System gebildet. Sie unterscheiden sich je nach R/3-Version voneinander. Während ältere Versionen für Input- und Output-Module jeweils eine gemeinsame Include-Datei erzeugen, wird in der neuen Version zunächst für jedes Modul eine eigene Datei angelegt. Sie können sich dieser Vorgabe anschließen, wenn Sie bei der Bearbeitung der Anwendung vorwiegend die bereitgestellten Navigationsmechanismen des Systems nutzen möchten. Es ist aber auch möglich, die Vorschläge des Systems

zu überschreiben und dadurch mehrere Module in einer Datei abzulegen. In den folgenden Beispielen wird der Einfachheit halber die Systemvorgabe übernommen. Abbildung 3.90 zeigt das Popup zur Auswahl der Include-Datei.

Nach der Bestätigung des Namens wird die Include-Datei durch das System angelegt. Das System startet dann den ABAP-Editor für die Include-Datei und übergibt dem Programmierer den automatisch generierten Quelltext zur weiteren Bearbeitung. Das neu angelegte Include enthält bereits einen Rumpf des Moduls (Anweisungen `MODULE` und `ENDMODULE`) und einige Kommentare.

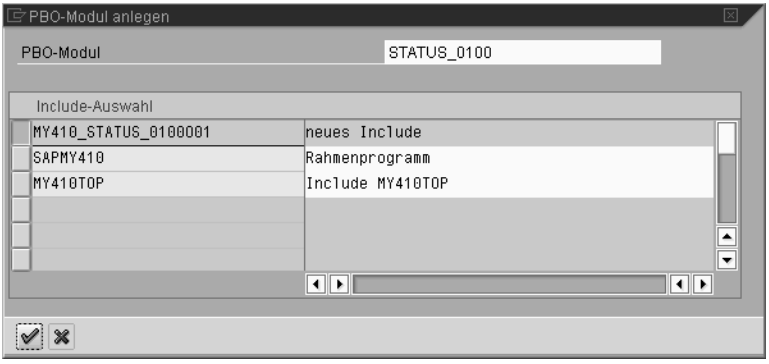

Abbildung 3.90 **© SAP AG**
Auswahl einer Include-Datei für PBO-Modul

Im Modul fügen Sie die im folgenden Listing sichtbaren Zeilen hinzu.

```
*---------------------------------------------------------------*
***INCLUDE MY410_STATUS_0100001 .
*---------------------------------------------------------------*
*&---------------------------------------------------------------*
*&      Module  STATUS_0100  OUTPUT
*&---------------------------------------------------------------*
*       text
*---------------------------------------------------------------*
MODULE status_0100 OUTPUT.
  CLEAR: value1, value2, result, op.
  SET TITLEBAR '001'.
  SET PF-STATUS 'STAT1'.
ENDMODULE.                 " STATUS_0100  OUTPUT
```

Diese Anweisung initialisiert die globalen Felder und lädt den Status `STAT1`, wodurch die dort definierten Menüs und andere Elemente für das Dynpro verfügbar werden. Die Anweisung `SET PF-STATUS` erwartet als Parameter den Namen des Status, der als Zeichenkettenkonstante, also in einfache Anführungszeichen eingeschlossen, notiert werden muss. Außerdem muss der Name des Status in Großbuchstaben angegeben werden.

Hinter den Namen des Moduls fügt das System automatisch den Begriff OUTPUT an. Module müssen bei ihrer Deklaration eindeutig einem Abschnitt der Dynpro-Verarbeitung zugeordnet werden. Das System sucht bei der Ausführung eines Moduls nur unter denen des jeweils aktuellen Verarbeitungsabschnitts. Die Kennzeichnung für PBO-Module erfolgt durch den Zusatz OUTPUT. Die PAI-Module werden bei ihrer Deklaration entweder gar nicht oder durch den Zusatz INPUT gekennzeichnet. In einem Modul-Pool können daher zwei Module mit identischem Namen existieren, eines davon mit dem Zusatz OUTPUT. Im Interesse eines übersichtlichen Programmcodes sollte auf diese Möglichkeit allerdings verzichtet werden.

Die durch den Status bereitgestellten Funktionscodes müssen natürlich nach der Dynpro-Bearbeitung im PAI-Abschnitt der Ablauflogik auch ausgewertet werden. Dies erfolgt im zweiten Modul, das auf die eben beschriebene Art und Weise angelegt wird. Das folgende Listing zeigt dessen Quelltext.

```
MODULE user_command_0100 INPUT.
  CASE fcode.
    WHEN 'OPCA'.
      result = value1 + value2.
      op     = '+'.
      LEAVE TO SCREEN 200.
    WHEN 'OPCS'.
      result = value1 - value2.
      op     = '-'.
      LEAVE TO SCREEN 200.
    WHEN 'OPCM'.
      result = value1 * value2.
      op     = '*'.
      LEAVE TO SCREEN 200.
    WHEN 'OPCD'.
      result = value1 / value2.
      op     = '/'.
      LEAVE TO SCREEN 200.
    WHEN 'FT03'.
      LEAVE PROGRAM.
  ENDCASE.
  CLEAR fcode.
ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Die Funktion des Moduls ist relativ einfach. Eine CASE-Anweisung wertet den vom Anwender ausgelöste Funktionscode aus. Je nach Funktionscode wird aus dem Inhalt der beiden Felder value1 und value2 das passende Ergebnis berechnet. Der eigentlich notwendige Test auf eine Division durch Null erfolgt in dieser Phase des Beispiels bewusst nicht. Er bleibt einer Erweiterung vorbehalten. Nach der Berechnung wird durch das Kommando LEAVE TO SCREEN 200 ein noch anzulegendes zweites Dynpro aufgerufen, in dem das Ergebnis ange-

zeigt wird. Damit das Programm ordnungsgemäß beendet werden kann, wertet das Modul außerdem den Funktionscode FT03 aus. Es reagiert auf diesen Funktionscode mit der Anweisung LEAVE PROGRAM, die eine Beendigung des aktuell laufenden Programms bewirkt. Nach Auswertung des Funktionscodes wird der Inhalt des Feldes fcode gelöscht. Dies sollte immer erfolgen. Der Grund liegt in der Behandlung der ↵-Taste durch das System. Diese Taste steht in Dynpros immer bereit, auch ohne dass ihr ein Funktionscode zugewiesen wurde. Sie bewirkt normalerweise die Abarbeitung des PAI-Teils eines Dynpros, um die dort enthaltenen Prüfroutinen auszuführen. Details dazu folgen. Allerdings wird der existierende Inhalt des Funktionscode-Felds des Dynpros durch Betätigen der ↵-Taste nicht geändert, wenn diese Taste keinen Funktionscode besitzt. Dies könnte zu unerwünschten Nebenwirkungen führen, da das User-Command-Modul einen Funktionscode erhält, der im Dynpro überhaupt nicht ausgelöst wurde.

Die Programmierung des ersten Dynpros ist damit abgeschlossen. Sie können die Anwendung nun auf korrekte Syntax prüfen und generieren. Ein Testlauf ist im Moment allerdings weder sinnvoll noch möglich.

Das zweite Dynpro

In der Ablauflogik des ersten Dynpros wird zur Anzeige des Resultats das Dynpro 200 aufgerufen. Dieses muss ebenfalls erstellt werden. Es ist einfacher aufgebaut als das erste. Auch die Erstellung des neuen Dynpros kann erheblich vereinfacht werden. Sie müssen, um das neue Dynpro anzulegen, nicht unbedingt zur Objektliste des Repository Browser zurückkehren. Innerhalb des User-Command-Moduls des ersten Dynpros reicht ein Doppelklick auf die Zahl 200 innerhalb der LEAVE TO SCREEN-Anweisung aus. Der Navigationsmechanismus der Entwicklungsumgebung erkennt, dass ein Sprung zum Dynpro 200 erfolgen soll, das im Moment noch nicht existiert, und legt dieses Dynpro an. Die Vorgehensweise entspricht der bereits beschriebenen, auf eine Wiederholung soll daher verzichtet werden.

Innerhalb des neuen Dynpros soll nur die Ausgabe des Resultats erfolgen. Sie können daher, wie in Bild 3.91 dargestellt, zunächst einen erläuternden Text direkt im Dynpro eingeben und dann hinter diesem Text mit dem Einfügewerkzeug das programminterne Feld result einfügen.

Standardmäßig wird das neue Feld als Ein-/Ausgabefeld erzeugt. Für den beschriebenen Einsatzfall ist dies nicht sinnvoll. Sie sollten daher dem Feld die Eingabe-Funktion entziehen. Stellen Sie dazu den Cursor auf das Feld und betätigen Sie die Drucktaste ATTRIBUTE. Es erscheint ein Popup (siehe Abbildung 3.92), in dem Sie die Eigenschaften des Dynpro-Felds pflegen können.

In diesem Popup ist das Flag EINGABEFELD zu deaktivieren und dafür das Flag für die zweidimensionale Anzeige (2-DIMENS.) zu aktivieren. Die zuletzt genannte Einstellung bewirkt, dass das Feld ohne den dreidimensional wirkenden Rahmen angezeigt wird. Es erscheint somit als normaler Text.

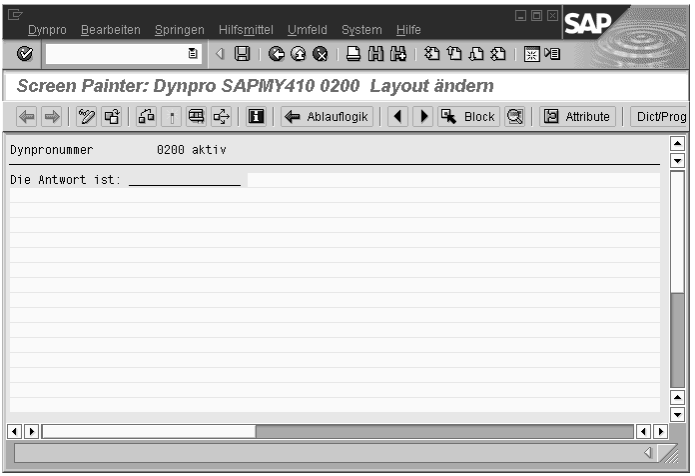

Abbildung 3.91 **© SAP AG**
Dynpro 200 des ersten Beispiels

Zum Abschluss der Bearbeitung des zweiten Dynpros ist in der Feldliste das Feld `fcode` zur Rückgabe des Funktionscodes einzutragen. Damit ist die Gestaltung des visuellen Teils des Dynpros bereits abgeschlossen.

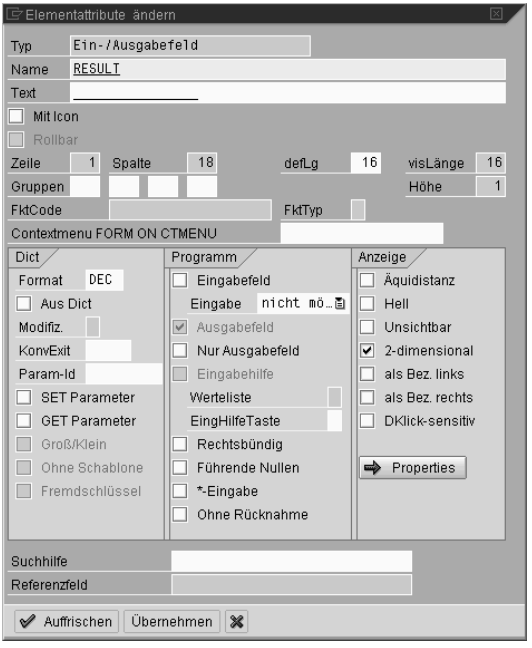

Abbildung 3.92 **© SAP AG**
Attribut-Dialog für Feld RESULT

In der Ablauflogik wird nur ein einziges Modul zur Auswertung des Funktionscodes benutzt. Der im nächsten Listing dargestellte Quellcode demonstriert dies.

```
PROCESS BEFORE OUTPUT.
*
PROCESS AFTER INPUT.
  MODULE user_command_0200.
```

Auch das User-Command-Modul selbst ist kürzer als das des ersten Dynpros, da es nur einen einzigen Funktionscode auswerten muss. An dieser Stelle der Anwendung muss lediglich die Rückkehr zum Dynpro 100 programmiert werden. Das folgende Listing demonstriert dies.

```
MODULE user_command_0200 INPUT.
  CASE fcode.
    WHEN 'FT03'.
      LEAVE TO SCREEN 100.
  ENDCASE.
ENDMODULE.                 " USER_COMMAND_0200  INPUT
```

Menüs und Funktionstasten

Der nächste Schritt zur funktionsfähigen Anwendung ist das Anlegen eines Status. Nur durch Elemente eines Status kann der Anwender aktiv mit der Anwendung kommunizieren. Innerhalb einer Anwendung existiert eine einzige so genannte Oberfläche, die wegen der eindeutigen Verbindung mit der Anwendung nicht nähert bezeichnet wird. Sie umfasst einen oder mehrere Status, von denen jeder einen eindeutigen Namen besitzt. Ein Dynpro kann in verschiedenen Phasen der Abarbeitung durchaus mehrere Status setzen, ein Status kann aber auch in mehreren aufeinander folgenden Dynpros benutzt werden. Details zum Status und zur Pflege der einzelnen Elemente wurden bereits im Abschnitt 3.3 („Die Oberfläche“) erläutert. An dieser Stelle soll daher die Bedienung des Menu Painter im Mittelpunkt stehen.

Während der Programmierung einer Anwendung wird zum Anlegen neuer Elemente überwiegend der Navigationsmechanismus genutzt. Wechseln Sie dazu zum Quelltext des Moduls `USER_COMMAND_0100` und führen Sie innerhalb der `SET PF-STATUS`-Anweisung einen Doppelklick auf den Namen des Status aus. Da noch kein Status existiert, müssen Sie zunächst bestätigen, dass ein solches Objekt angelegt werden soll (siehe Abbildung 3.93).

Das System fordert nun zur Eingabe eines Kurztextes und zur Auswahl des Statustyps auf (Abbildung 3.94).

Für dieses Beispiel muss der Statustyp DIALOGSTATUS selektiert werden. Der Kurztext ist wie immer frei wählbar. Nach Bestätigung der Eingabe befinden Sie sich bereits im Grundbild des Menu Painter, hier in der Vollbildansicht (Abbildung 3.95).

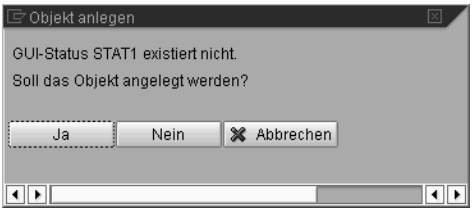

Abbildung 3.93 **© SAP AG**
Status anlegen

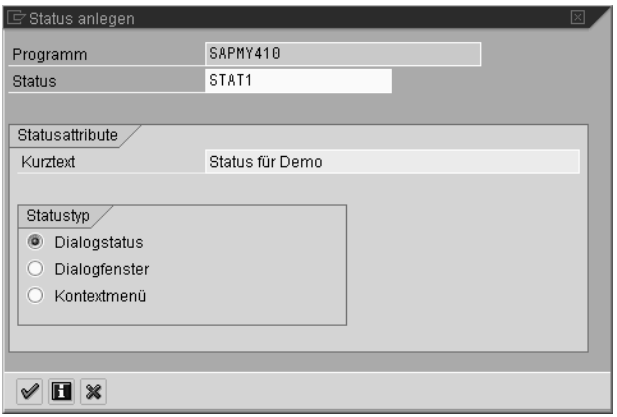

Abbildung 3.94 **© SAP AG**
Kurztext und Statustyp festlegen

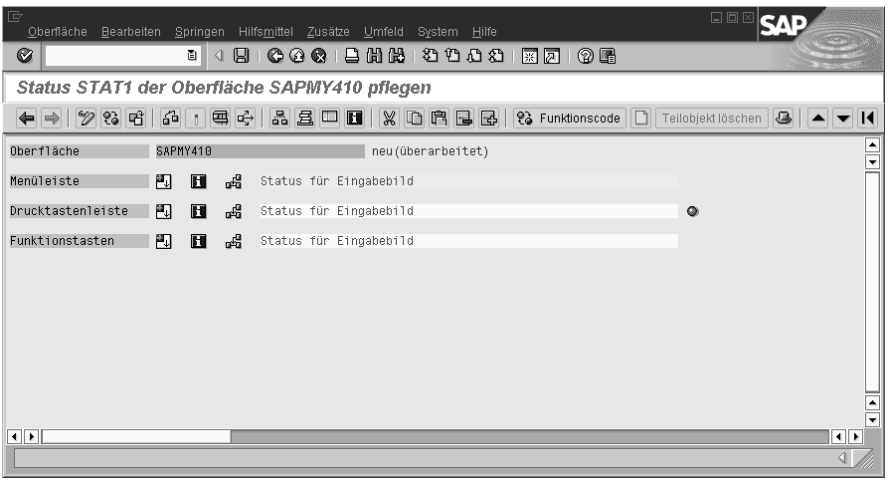

Abbildung 3.95 **© SAP AG**
Grundbild des Menu Painter

Öffnen Sie nun durch einen Klick auf das Symbol mit dem grünen Plus-Zeichen die Arbeitsfläche für das Menü. Direkt unter dem Text MENÜLEISTE finden Sie ein weiteres Symbol mit der Beschriftung NORMEN EINBLENDEN. Lösen Sie diese Funktion aus. Der Menu Painter fügt einige Standard-Menüeinträge ein. Mit einem Doppelklick auf den ersten, mit <OBJEKT> bezeichneten Menüeintrag können Sie das dazugehörige Pulldown-Menü öffnen. Abbildung 3.96 zeigt Ihnen den Menu Painter in diesem Zustand.

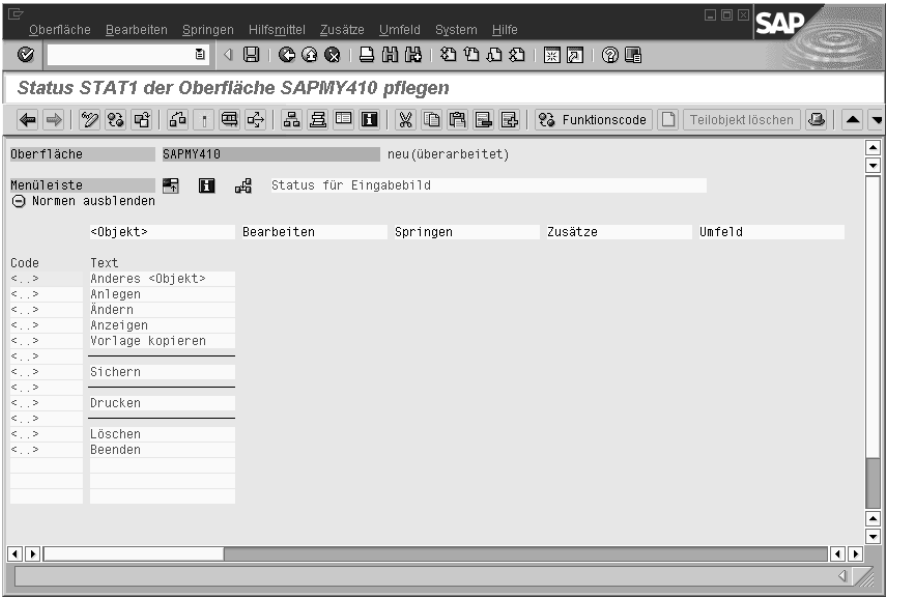

Abbildung 3.96 **© SAP AG**
Menu Painter mit vorbelegtem Menü

Die in der Ablauflogik bereits ausgewerteten Funktionscodes müssen nun im Menü hinterlegt werden. Sie können in den entsprechenden Spalten des Menüs links den Funktionscode und rechts den beschreibenden Text eintragen. Außerdem müssen Sie den Platzhalter <OBJEKT> durch eine geeignete Bezeichnung für das Menü ersetzen. Die nicht benötigten Einträge können Sie der besseren Übersicht wegen entfernen. Es stört allerdings auch nicht, wenn sie erhalten bleiben, da Menüeinträge ohne Funktionscode vom System automatisch ausgeblendet werden. Das Aussehen des Menu Painter zu diesem Zeitpunkt zeigt Ihnen Abbildung 3.97.

In diesem Menü werden nur die vier Funktionscodes abgelegt, mit denen eine mathematische Operation ausgelöst wird. Der außerdem noch verwendete Funktionscode `FT03` wird im Menü SPRINGEN und dort für die Funktion ZURÜCK angelegt. Diese Menüfunktion soll die Rückkehr zum vorangegangenen Dynpro auslösen.

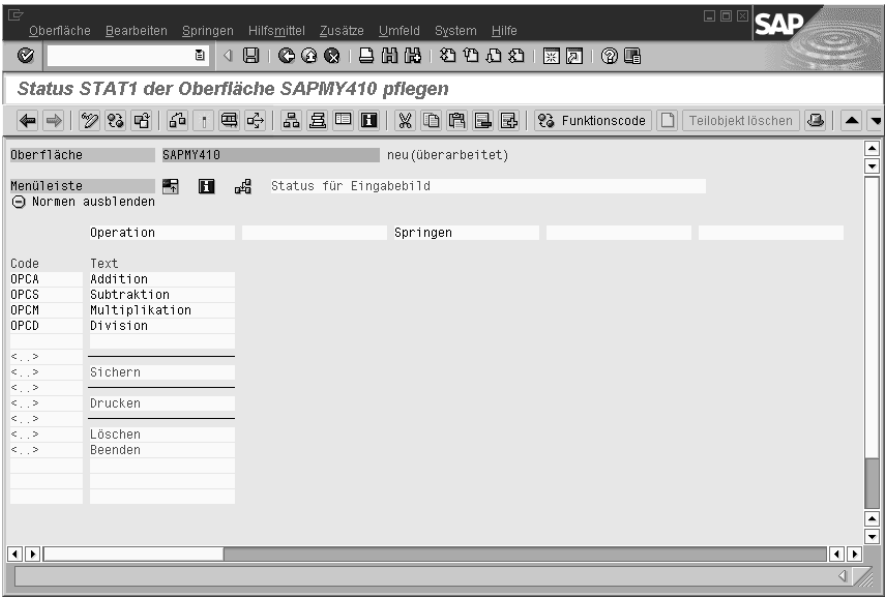

Abbildung 3.97 **© SAP AG**
Pflege des ersten Pulldown-Menüs

Diese Funktionalität gehört zu den Standard-Funktionen, die laut SAP-Style Guide stets mit der Funktionstaste F3, der Menüfunktion SPRINGEN | ZURÜCK und dem entsprechenden Symbol (grüner, nach links gerichteter Pfeil) ausgelöst werden sollen. Aus diesem Grund ist im SPRINGEN-Menü bereits ein Eintrag für diese Funktion vorhanden.

Die Menüs BEARBEITEN, ZUSÄTZE und UMFELD werden in diesem Beispiel nicht benötigt. Sie können sie löschen, indem Sie den Cursor auf das Menü stellen und die Menüfunktion BEARBEITEN | LÖSCHEN bzw. die entsprechende Schaltfläche in der Drucktastenzeile auslösen.

Das soeben erzeugte Menü reicht im Prinzip aus, um die Anwendung zu bedienen. Mehr Komfort bieten aber Funktions- und Drucktasten. Schließen Sie dazu den Menü-Arbeitsbereich durch einen Klick auf das Symbol mit dem roten Minus-Zeichen und öffnen Sie stattdessen den Arbeitsbereich für die Funktionstasten (Abbildung 3.98).

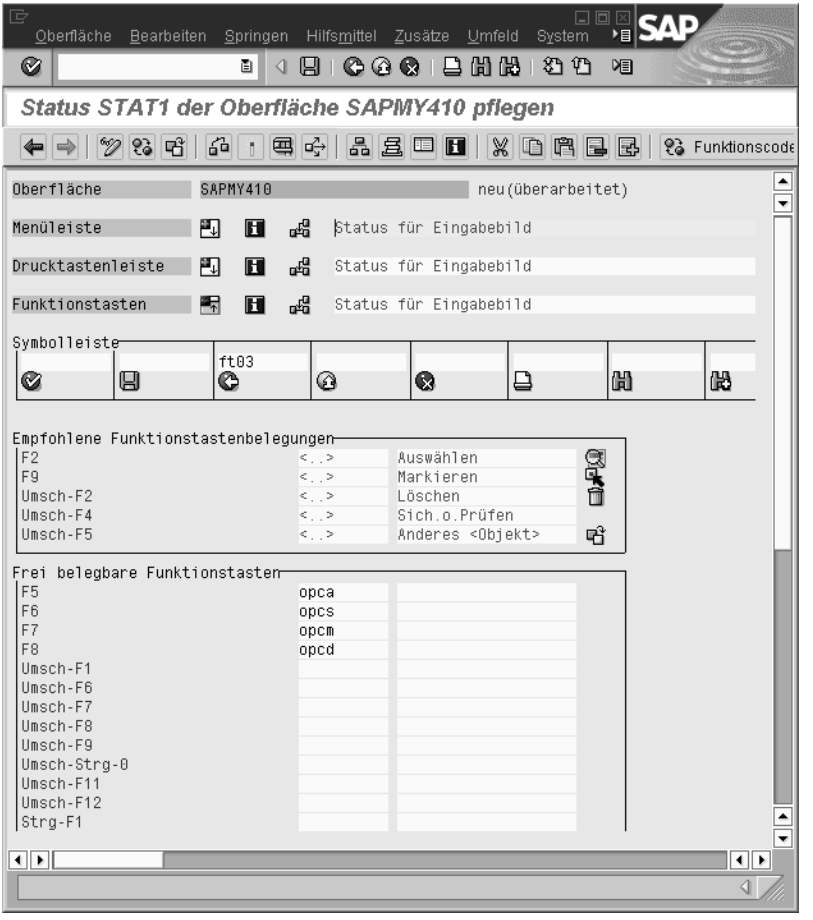

Abbildung 3.98 **© SAP AG**
Pflege der Funktionstasten

Tragen Sie, wie aus der Abbildung ersichtlich, die vier `OPCx`-Funktionscodes bei den Funktionstasten F5 bis F8 und den Funktionscode `FT03` beim Zurück-Symbol (grüner Pfeil) ein. Bei den Einträgen für die Funktionstasten müssen Sie keine Bezeichnungen pflegen. Diese werden automatisch aus dem Menü übernommen.

Ähnlich verfahren Sie bei den Drucktasten. Speichern Sie zunächst den aktuellen Status ab und öffnen Sie den Arbeitsbereich für die Drucktasten. Tragen Sie dort, wie aus Abbildung 3.99 ersichtlich, die vier `OPCx`-Funktionscodes in die ersten vier Felder ein.

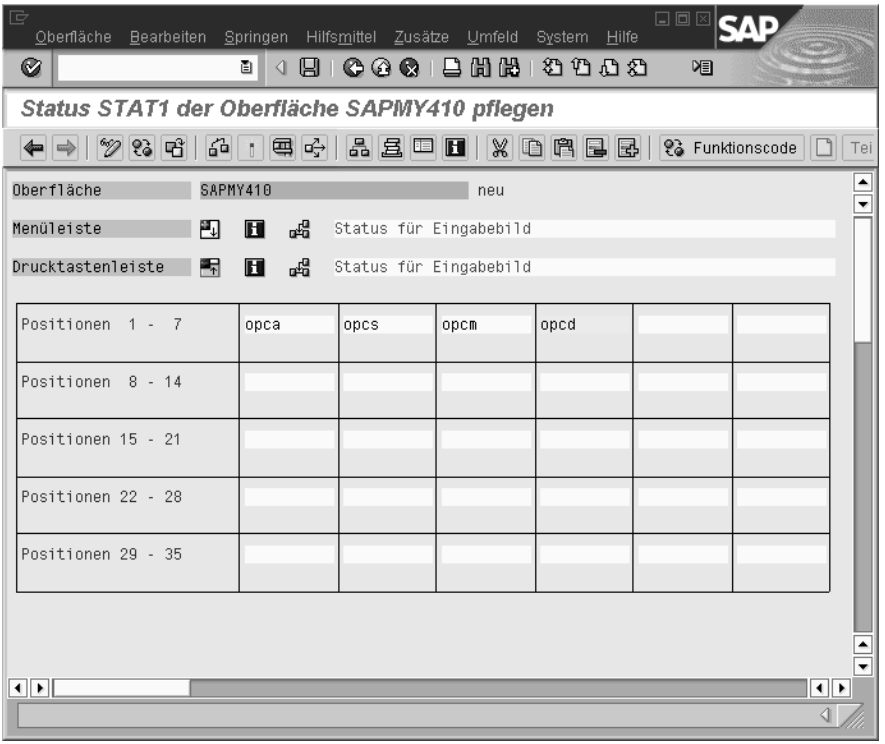

Abbildung 3.99 **© SAP AG**
Pflege der Drucktasten

Die Pflege des Status ist damit abgeschlossen. Sie können ihn nun generieren (Menüfunktion OBERFLÄCHE | GENERIEREN oder Schaltfläche in Drucktastenzeile) und mit der Funktionstaste F3 bzw. den entsprechenden Menüfunktionen zum Quelltext-Editor zurückkehren. Dort legen Sie, ebenfalls mit Hilfe des Navigationsmechanismus, noch den Titel an. Dieser Titel erscheint bei laufender Anwendung in der Kopfzeile des Fensters. Zur Pflege des Titels müssen Sie lediglich den gewünschten Text in einem Popup (Abbildung 3.100) eintragen.

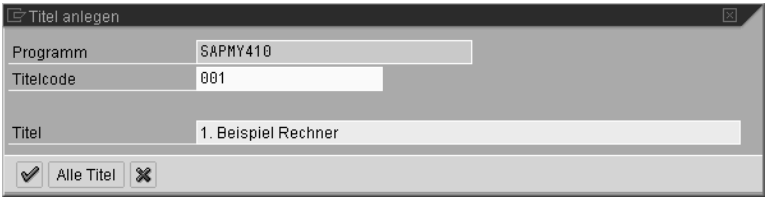

Abbildung 3.100 **© SAP AG**
Pflege des Titel-Textes

Mit der Pflege des Status haben Sie alle Objekte erzeugt, die Sie für eine lauffähige Anwendung benötigen. Die Pflege des Fenstertitels ist optional. Sie können nun zur Objektliste des Repository-Browsers zurückkehren und zur Sicherheit die gesamte Anwendung nochmals aktivieren.

Die Anwendung ist in einem Zustand, in dem sie ausgeführt werden könnte. Allerdings kann eine Dialoganwendung nur dann ausgeführt werden, wenn Sie mit einer Transaktion verbunden wurde. Diese muss jetzt angelegt werden. Positionieren Sie dazu die Maus innerhalb der Objektliste des Object Navigator auf den Eintrag des Modul-Pools und rufen Sie durch einen Klick mit der rechten Maustaste das Kontextmenü auf. Wählen Sie die Funktion ANLEGEN | TRANSAKTION. Das System fordert nun in einem Popup zur Angabe des Transaktionsnamens (hier `Y401`), eines beliebigen Kurztextes sowie der Auswahl der Transaktionsart auf. Für letztgenannte Einstellung wird vom System bereits die Einstellung Dialogtransaktion angeboten. Diese müssen Sie übernehmen (siehe Abbildung 3.101).

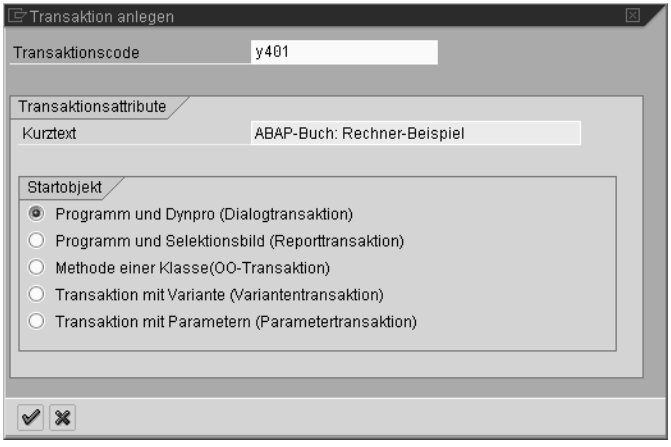

Abbildung 3.101 **© SAP AG**
Transaktion anlegen

Im nächsten Schritt fordert das R/3-System zur Eingabe zusätzlicher Informationen auf, die vom Transaktionstyp abhängen. Für Dialogtransaktionen sind auf jeden Fall der Programmname und das Start-Dynpro einzutragen. Außerdem muss innerhalb des Abschnitts GUI-Konformität mindestens das Auswahlfeld SAPGUI FÜR WINDOWS markiert werden. Alle anderen Einstellungen können aus Abbildung 3.102 übernommen werden.

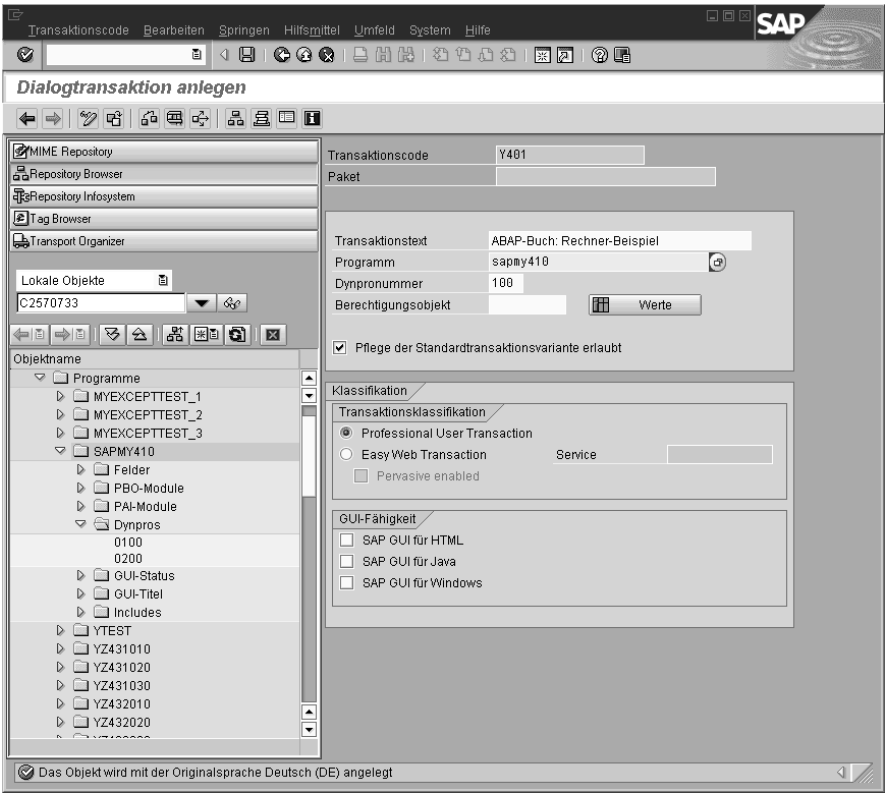

Abbildung 3.102 **© SAP AG**
Eigenschaften einer Transaktion pflegen

Natürlich kann die Transaktionspflege auch direkt über den Transaktionscode `SE93` aufgerufen werden.

Nachdem die Angaben eingetragen und gesichert wurden, kann die erste Dialoganwendung mittels der Eingabe des Transaktionscodes im OK-Feld ausgeführt werden. Dazu benutzen Sie innerhalb der Transaktionspflege das Symbol zum Testen oder sie verwenden den soeben erzeugten Transaktionscode für den Direktaufruf. Geben Sie im OK-Feld den Wert `/nY401` ein und betätigen Sie die ↵-Taste. Dadurch wird die aktuelle Transaktion abgebrochen und eine neue gestartet.

Nach Start der soeben erzeugten Transaktion erscheint das Dynpro 100, in dem Sie zwei Operanden eintragen (Abbildung 3.103).

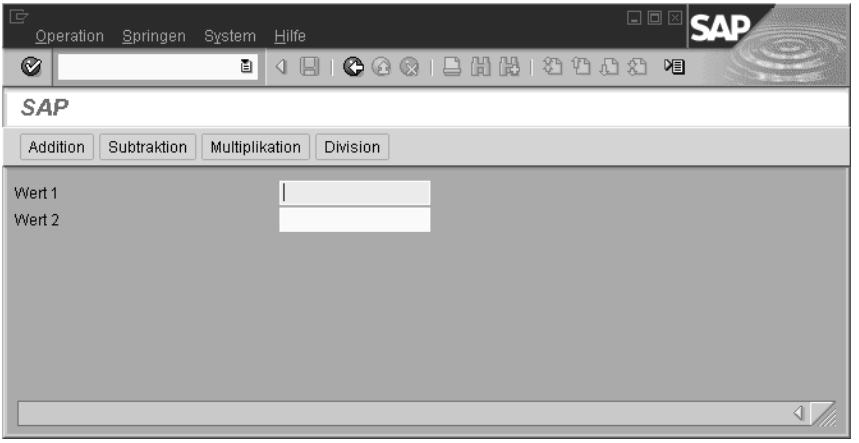

Abbildung 3.103 **© SAP AG**
Startbild des ersten Beispiels

Durch eine der Schaltflächen der Drucktastenzeile lösen Sie eine der vier Operationen aus. Das Ergebnis zeigt die Anwendung im zweiten Dynpro an (Abbildung 3.104)

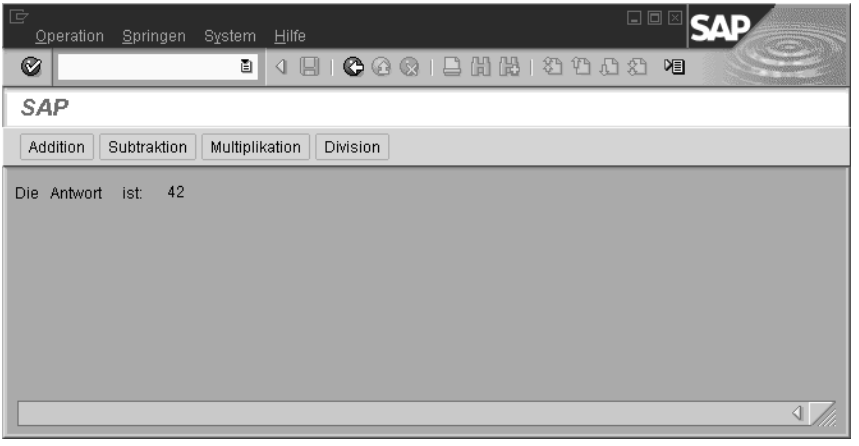

Abbildung 3.104 **© SAP AG**
Das zweite Dynpro des Beispiels

Die eben erzeugte Anwendung zeigt zwar wichtige Merkmale einer Dialoganwendung, ist in vielen Punkten aber noch verbesserungsbedürftig. So fehlt beispielsweise eine Prüfung, mit der eine Division durch 0 verhindert werden kann. Außerdem ist das Beenden der Anwendung aus dem zweiten Dynpro heraus etwas umständlich, da zunächst zum 1. Dynpro zurückgekehrt werden muss. Bei Eingabe eines nicht korrekten Werts in eines der Operanden-Felder

(z.B. „123-456“) tritt eine weitere Besonderheit des SAP-Systems in Erscheinung. Das System selbst erkennt, dass der eingegebene Wert nicht zum Datentyp des Feldes passt und erzeugt eine Fehlermeldung. Diese Prüfung findet statt, bevor die Module des PAI-Abschnittes ausgeführt werden. Die im User-Command-Modul programmierten Auswertungen, z.B. die des Funktionscodes `FT03`, finden somit nicht statt. Eine Fortsetzung der Anwendung ist daher erst möglich, nachdem der Fehler verursachende Wert durch eine korrekte Eingabe ersetzt wurde.

Vervollständigen der Funktionalität

Dieser Abschnitt beschreibt weitere Details zur Dialogprogrammierung. Im Mittelpunkt steht jetzt nicht mehr die prinzipielle Funktionsweise einer solchen Anwendung, sondern die Steigerung des Bedienkomforts. Die Anwendung wird dazu kopiert, die Erweiterungen finden in der Kopie (Programm `SAPMY420`) statt.

Zunächst zur Prüfung der Feldinhalte. Neben einigen automatisch vom System ausgeführten Prüfungen können bzw. müssen natürlich auch individuell programmierte Prüfungen stattfinden. Falls diese Prüfungen einen Fehler erkennen, müssen die fehlerhaften Einträge in den Dynpro-Feldern korrigiert werden, die Bearbeitung des Dynpros beginnt also von vorn.

Die Feldprüfungen in einem Dynpro können feldbezogen ausgeführt werden. Das heißt, dass für jedes Feld oder eine Gruppe logisch zusammengehöriger Felder eine eigene Prüfroutine erstellt werden kann. Feld(er) und Prüfroutine werden mit einer speziellen Anweisung in der Ablauflogik miteinander verbunden. Löst eine Prüfroutine eine Fehlermeldung aus, werden nur die Eingabefelder des Dynpros eingabebereit, die mit der Prüfroutine verbunden wurden. Der Cursor steht im ersten zu bearbeitenden Feld. Dieses Verfahren erhöht die Übersicht und verhindert neue Fehler durch unbeabsichtigtes Überschreiben korrekter Werte. Außerdem können Feldprüfungen davon abhängig gemacht werden, ob der Inhalt des Feldes überhaupt bearbeitet wurde. Dies erspart unnötige Prüfungen und verbessert die Performance einer Anwendung.

Das Prinzip der Prüfroutinen kann in der Demoanwendung relativ einfach demonstriert werden. Dazu wird in der Ablauflogik des Dynpros 100 eine zusätzliche Anweisung im PAI-Teil eingefügt:

```
PROCESS AFTER INPUT.
  FIELD value2 MODULE check_0100.
  MODULE USER_COMMAND_0100.
```

Anschließend wird die Ablauflogik gespeichert und generiert. Mittels Doppelklick auf die Zeichenkette `CHECK_0100` wird das fehlende Modul angelegt. Im Modul wird geprüft, ob der aktuelle Funktionscode eine Division auslöst. Ist dies der Fall, wird der Wert `value2` auf 0 geprüft.

```
MODULE check_0100 INPUT.
  IF fcode = 'OPCD' AND value2 = 0.
    MESSAGE ID 'YZ4' TYPE 'E' NUMBER '000' with
               'Division by Zero!'.
  ENDIF.
ENDMODULE.                    " CHECK_0100  INPUT
```

Nach Eingabe des Quellcodes, Generieren des Dynpros sowie Aktivieren des Gesamtprogramms kann die Anwendung erneut getestet werden. Die Eingabe einer 0 im Feld `value2` sollte nun zu einer Fehlermeldung führen, wenn Sie eine Division ausführen. Die Pflege der Nachrichtentexte sowie die Kommandos zum Auslösen einer Nachricht wurden bereits in Abschnitt 3.2.16 beschrieben. Falls Sie, so wie in diesem Abschnitt beschrieben, die Nachrichtenklasse und die Dummy-Nachricht angelegt haben, erhalten Sie die aus dem Listing ersichtliche Fehlermeldung.

Durch das Auslösen eines Fehlers mit dem Kommando `MESSAGE` beginnt die Verarbeitung des aktuellen Dynpros von vorn. Dabei sind alle Felder eingabebereit, die durch eine `FIELD`-Anweisung in der Ablauflogik mit dem Fehler auslösenden Modul verbunden wurden. Dies ist im vorliegenden Fall nur das Feld `value2`.

Nach wie vor besteht das Problem, dass ein Dynpro, in dessen PAI-Abschnitt eine Fehlermeldung ausgelöst wird, nicht beendet werden kann. Die automatischen Prüfungen sind vom eben eingefügten Modul ja nicht betroffen. Aus diesem Grund existiert ein spezieller Funktionscode-Typ, der durch bestimmte Module ausgewertet wird, die wiederum vor den automatischen Fehlerprüfungen abgearbeitet werden. Diese Module werden stets als erstes Modul im PAI-Abschnitt eines Dynpros aufgerufen. Die Ablauflogik des Dynpros 100 ist daher nochmals zu ergänzen:

```
PROCESS AFTER INPUT.
  MODULE exit_0100 AT EXIT-COMMAND.
  FIELD value2 MODULE check_0100.
  MODULE user_command_0100.
```

Der Zusatz `AT EXIT-COMMAND` am Modul `exit_0100` bewirkt, dass dieses nur dann aufgerufen wird, wenn der Funktionscode den Funktionstyp E für „Exit-Kommando“ hat. Im Modul wird für die beiden Exit-Funktionscodes das Beenden der Anwendung programmiert. Es kommt eine `CASE`-Anweisung zum Einsatz, obwohl die Aufgabe auch mit einer kürzeren `IF`-Anweisung zu erfüllen wäre. Eine `CASE`-Anweisung kann aber wesentlich einfacher erweitert werden, falls später zusätzliche Funktionscodes ausgewertet werden müssen.

```
MODULE EXIT_0100.
  CASE FCODE.
    WHEN 'FT12' OR 'FT15'.
      LEAVE PROGRAM.
```

```
  ENDCASE.
ENDMODULE.
```

Natürlich müssen die beiden Funktionscodes auch im Status angelegt werden, damit sie überhaupt ausgelöst werden können. Wie aus den Bezeichnungen der Funktionscodes hervorgeht, werden die beiden Funktionstasten F12 und F15 benutzt. Diese Funktionstasten haben eine vordefinierte Bedeutung. Der Funktionstaste F12 ist das Symbol mit dem schräg gestellten roten Kreuz zugeordnet. Die entsprechende Menüfunktion ist BEARBEITEN | ABBRECHEN. Zur Funktionstaste F15 gehört der nach oben gerichtete gelbe Pfeil bzw. die Menüfunktion BEENDEN im ersten Pulldown-Menü. Das Anlegen von Menüfunktionen und Funktionscodes wurde bereits beschrieben. An dieser Stelle soll nur auf das Setzen des Funktionstyps eingegangen werden. Fügen Sie zunächst die beiden genannten Funktionscodes in das Menü und die Symbolleiste ein. Die Symbolleiste pflegen Sie unter der Zwischenüberschrift für die Funktionstasten. Anschließend springen Sie mit der Menüfunktion SPRINGEN | OBJEKTLISTEN | FUNKTIONSLISTE zur Bearbeitung der Funktionscodes. In dieser Liste erscheinen alle Funktionscodes, die in den verschiedenen Status der Oberfläche benutzt werden. Durch einen Doppelklick auf einen der beiden neuen Funktionscodes erhalten Sie ein Popup (Abbildung 3.105), in dem Sie den Funktionstyp eintragen können. In der Spalte TYP wird für den Funktionscode FT12 und FT15 der Typ E für Exit-Code eingetragen. Der Status wird jetzt abgespeichert und die neue Oberfläche generiert. Der Menu Painter kann verlassen werden.

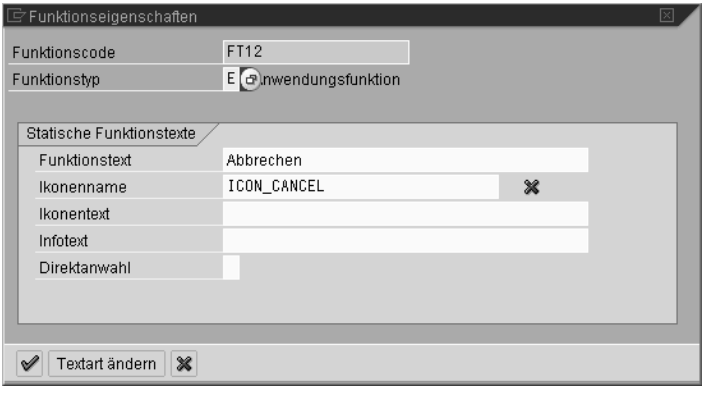

Abbildung 3.105 **© SAP AG**
Funktionstyp setzen

Der Vollständigkeit halber können Sie nun auch noch das Dynpro 200 überarbeiten. Auch dort können Sie ein Exit-Modul einbinden.

```
PROCESS BEFORE OUTPUT.
*
PROCESS AFTER INPUT.
```

```
  MODULE exit_0200 AT EXIT-COMMAND.
  MODULE user_command_0200.

MODULE EXIT_0200.
  CASE FCODE.
    WHEN 'FT12'.
      LEAVE TO SCREEN 100.
    WHEN 'FT15'.
      LEAVE PROGRAM.
  ENDCASE.
ENDMODULE.
```

Sie können die Anwendung nun testen. Die Funktionstaste F15 bzw. das Beenden-Symbol beendet das Programm. Im Dynpro 100 führt das Abbrechen-Symbol bzw. die Funktionstaste F12 ebenfalls zum Programmende. Im Dynpro 200 hingegen bewirkt dieser Funktionscode analog zu `FT03` die Rückkehr zum Dynpro 100. Bedeutsam ist hier, dass Sie das Dynpro 100 auch dann mit F12 oder F15 beenden können, wenn ein falscher Wert in einem der beiden Eingabefelder eingetragen wurde.

Dieses Beispiel macht deutlich, warum die Funktionscodes in zwei Gruppen aufgeteilt und getrennt ausgewertet werden. Bestimmte Aktivitäten wie das Abspeichern von Daten oder die Verzweigung in weitere Dynpros dürfen nur stattfinden, wenn die Einträge im Dynpro korrekt sind. Andere Funktionen wie das Abbrechen einer Anwendung müssen aber gerade bei fehlerhaften Eingaben möglich sein. Am einfachsten lassen sich diese Anforderungen realisieren, wenn die Exit-Funktionscodes vor Ausführen der Feldprüfungen ausgewertet werden, alle anderen aber danach. Falls ein Exit-Modul abgearbeitet wird, dort aber kein Abbruch der Bearbeitung erfolgt, z.B. weil eine entsprechende Sicherheitsabfrage nicht bestätigt wurde, werden alle anderen Module ausgeführt. In diesem Fall können auch die anderen Module einen Exit-Funktionscode auswerten. Meist wird davon Gebrauch gemacht, um die im Dynpro erfassten Werte vor dem Verlassen der Anwendung abspeichern zu können. Das ausführliche Beispiel in Kapitel 7 demonstriert diese Programmiertechnik.

3.7.2 Das Dynpro im Detail

Kernstück der Dialoganwendungen sind die Dynpros. Die im Fullscreen des Dynpros sichtbaren Eingabeelemente entsprechen denen des SAA/CUA-Standards. Es stehen unterschiedliche Elemente zur Verfügung, nicht nur das bisher verwendete Editierfeld. Die Eigenschaften der Elemente im Dynpro können zudem noch über eine Reihe von Attributen modifiziert werden. In der Ablauflogik und den diversen Modulen werden Anweisungen benutzt, die speziell auf das Dynpro-Konzept zugeschnitten sind. Diese Informationen stehen im Mittelpunkt dieses Abschnitts.

Dynprotypen

Das Dynpro in seiner Grundform nimmt stets die gesamte Arbeitsfläche ein. In speziellen Fällen, z.B. bei der Anzeige von Zusatzinformationen, sind Popup-Dynpros mit fest programmierter Größe sinnvoller. Auch die Selektionsbildschirme von Reports sind Dynpros, deren Verarbeitung durch das System aber auf andere Weise ausgeführt wird als die normaler Dynpros. Man unterscheidet vier verschiedene Dynpro-Typen, von denen Sie aber nur drei wirklich bearbeiten können. Dynpros des vierten Typs werden automatisch durch das System generiert.

Normal

Alle bisher in den Demoanwendungen erzeugten Dynpros entsprechen diesem Typ. Die Dynpros nehmen bei Ausführung die gesamte Arbeitsfläche ein. Sie verfügen über eine Ablauflogik, die vom Anwender bearbeitet werden kann und muss. Normale Dynpros werden nacheinander abgearbeitet. Die Reihenfolge der Abarbeitung wird durch das Attribut FOLGE-DYNPRO oder durch Anweisungen in der Ablauflogik (`SET SCREEN ...`, `LEAVE ...`) festgelegt. Die Abarbeitung dieser Dynpros ähnelt daher der sequenziellen Abarbeitung von Anweisungen eines Programms.

Subscreen

Subscreens sind Dynpros, die zur Laufzeit dynamisch in ein anderes Dynpro eingebunden werden müssen. Sie sind nicht eigenständig lauffähig. Derartige Subscreens müssen speziellen Anforderungen genügen. So dürfen sie keine Exit-Funktionscodes verarbeiten und kein Feld zur Aufnahme des Funktionscodes besitzen. Außerdem können Subscreens die aktuelle Oberfläche (Status) nicht beeinflussen. Subscreens werden in der Ablauflogik des übergeordneten Dynpros mit einem speziellen Kommando aufgerufen. Ausführlichere Angaben zur Verwendung dieser Dynpros finden Sie daher im Unterkapitel „Die Kommandos der Ablauflogik“ (S. 404).

Modales Dialogfenster

Spezielle Anweisungen der Ablauflogik erlauben es, von der üblichen sequenziellen Abarbeitung mehrerer Dynpros abzuweichen und ein beliebiges Dynpro quasi als Unterprogramm aufzurufen. Dazu dient das Kommando `CALL SCREEN`. Werden so aufgerufene Dynpros mit `LEAVE TO SCREEN 0` beendet, erfolgt nicht die Beendigung des gesamten Programms, sondern nur der Rücksprung zum aufrufenden Dynpro.

Spezielle Zusätze zum `CALL`-Kommando erlauben es, derartige Dynpros als Popup anzuzeigen. Das aufrufende Dynpro wird dann nicht durch das neue ersetzt, sondern dieses wird in einem eigenen Fenster angezeigt. Eingaben sind nur noch in diesem Fenster möglich. Das Menü der Anwendung und das OK-Feld sind nicht wirksam. Das Popup reagiert nur noch auf Funktions- und

Drucktasten, die im aktuellen Status definiert wurden. Die Drucktasten werden in einem Popup am unteren Rand des Fensters angezeigt. Falls ein Dynpro als Popup aufgerufen werden soll, muss es den Dynpro-Typ MODALES DIALOGFENSTER erhalten.

Selektions-Dynpro

Ein Selektions-Dynpro generiert das System automatisch, wenn in einer Anwendung Kommandos benutzt werden, die einen Selektionsbildschirm definieren. Diese Dynpros können zwar in den Screen Painter geladen, dürfen aber keinesfalls verändert werden. Die Selektions-Dynpros verfügen über eine Ablauflogik, diese enthält aber Module, die direkt durch das System bereitgestellt werden. Diese Module existieren nicht als echter ABAP-Programmcode und können somit auch nicht modifiziert werden. Jede Änderung der Ablauflogik von Selektions-Dynpros führt mit hoher Wahrscheinlichkeit zu Fehlfunktionen des Reports!

Feldarten im Dynpro

In der sichtbaren Oberfläche des Dynpros werden bei der Gestaltung Elemente zur Ein- und Ausgabe von Daten angelegt. Diese Elemente werden als Dynpro-Felder bezeichnet. Bei Bearbeitung des Dynpros im pseudografischen Screen Painter werden die Felder durch Gruppen verschiedener alphanummerischer Zeichen dargestellt. Diese Zeichen übernehmen die Rolle eines Platzhalters. Die Felder können entweder manuell durch Eingabe der jeweiligen Platzhalterzeichen im Fullscreen oder durch diverse Hilfsmittel erzeugt werden. Die Eigenschaften der Felder werden durch die Feldart und verschiedene Attribute bestimmt. Die Feldart und ein Teil der Attribute können durch den Programmierer bei der Gestaltung des Dynpros festgelegt werden. Einige spezielle Attribute werden aber automatisch durch den Screen Painter vorgegeben und entziehen sich der Bearbeitung durch den Programmierer.

Der Screen Painter verfügt über mehrere Werkzeuge. Eines davon ist der Layout-Editor, mit dem die visuelle Oberfläche des Dynpros gestaltet wird.

Der Screen Painter kennzeichnet Felder, gleichgültig welchen Typs, immer durch einen farbig vom Hintergrund abgesetzten Block. Ein solcher Block kann nicht mehr zeichenweise, sondern nur noch über diverse Funktionen des Layout-Editors bearbeitet werden. Der Layout-Editor ist selbst ein Dynpro. Dies bedeutet, dass manuelle Eingaben (z. B. beim manuellen Anlegen eines Feldes) erst dann erkannt und ausgewertet werden, wenn die PAI-Bearbeitung des Dynpros des Fullscreen-Editors ausgeführt wird. Dies erfolgt durch Auswahl einer Menüfunktion oder Drucktaste bzw. durch Betätigen der ↵-Taste. Erst dann findet beispielsweise die Umwandlung einer Zeichenkette in ein echtes Dynpro-Feld statt.

Die im Dynpro möglichen Feldarten lassen sich in drei große Gruppen einteilen:

❶ Elementare Elemente mit Bearbeitungsfunktionalität

❷ Statische Elemente zur grafischen Gestaltung des Dynpros

❸ Komplexe Elemente (Controls)

Jeweils ein Vertreter der ersten beiden Gruppen wurde im Demobeispiel bereits vorgestellt. Für diese beiden Gruppen existiert ein grundlegendes Element, von dem alle anderen Elemente der Gruppe abgeleitet werden.

Dieser Abschnitt soll zunächst alle Feldarten und ihre Erzeugung beschreiben. Anschließend werden die Attribute erläutert, mit denen Sie die Eigenschaften der Felder zusätzlich modifizieren können.

Ein Eingabefeld besteht üblicherweise aus zwei Elementen: einer Feldbenennung, also dem Bezeichner bzw. im SAP-Sprachgebrauch dem Schlüsselwort, und der Feldschablone, dem eigentlichen Eingabebereich. Beide erscheinen auf dem Bildschirm. Das Schlüsselwort ist ein Textelement, das der Anwender des Programms nicht ändern kann. Die Feldschablone hingegen stellt das eigentliche Eingabefeld dar, in dem der Editiervorgang stattfindet. Damit der editierte Wert im Programm auch verfügbar ist, muss er natürlich mit einem Datenfeld verbunden werden. Dieses Datenfeld bestimmt dann automatisch einige Eigenschaften des Eingabefelds.

Wenn Sie die folgenden Erläuterungen an Ihrem System nachvollziehen möchten, legen Sie entweder ein neues Programm oder ein neues Dynpro innerhalb eines der bereits existierenden Anwendungen an. Im Top Include werden die folgenden Datendeklarationen eingetragen. Nach der Modifikation des Quelltextes muss die Anwendung auf jeden Fall aktiviert werden.

```
TABLES:
  tadir.

DATA:
  rb1 TYPE c,
  rb2 TYPE c,
  rb3 TYPE c,
  cb1 TYPE c,
  cb2 TYPE c,
  cb3 TYPE c.
```

Schlüsselfelder

Schlüsselfelder sind statische Textelemente innerhalb der Dynpro-Arbeitsfläche. Sie dienen zum Einblenden allgemeiner Informationen oder der Bezeichnungen von Eingabefeldern. Schlüsselfelder werden meist mit dem Einfügewerkzeug (Menüfunktion SPRINGEN | DICT./PROGRAMMFELDER) erstellt. Falls mit diesem Werkzeug Dictionary-Felder in das Dynpro übernommen werden, besitzen die

Eingabe- und Schlüsselfelder automatisch einen Bezug zum Dictionary. Der jeweilige Text wird der Beschreibung eines Datenelements im Data Dictionary entnommen. Beim Anlegen eines Feldes mit dem Einfügewerkzeug können Sie festlegen, welcher der zur Verfügung stehenden Texte des Datenelements ausgewählt werden soll. Diese Auswahl wird im Attribut MODIFIZIERT des jeweiligen Feldes abgelegt.

Bei der Benutzung des Einfügewerkzeugs werden üblicherweise Feldschablonen und Schlüsselfelder gleichzeitig angelegt. Sie können aber auch jede Feldart einzeln erzeugen.

Sie können den Inhalt eines per Einfügewerkzeug eingefügten Schlüsselfelds nachträglich verändern. Derartig modifizierte Felder werden durch einen Kennbuchstaben im Attribut MODIFIZIERT des jeweiligen Feldes gekennzeichnet. Weiterhin ist es möglich, Schlüsselfelder ohne Dictionary-Bezug anzulegen. Dazu wird der gewünschte Text direkt im Dynpro eingetragen.

Sowohl die Modifikation existierender Schlüsselfelder mit Dictionary-Bezug als auch die Erzeugung individueller Schlüsselfelder ohne Dictionary-Bezug führen zu einem erhöhten Aufwand bei einer eventuellen Übersetzung. Nach Möglichkeit sollte daher auf diese Variante verzichtet werden.

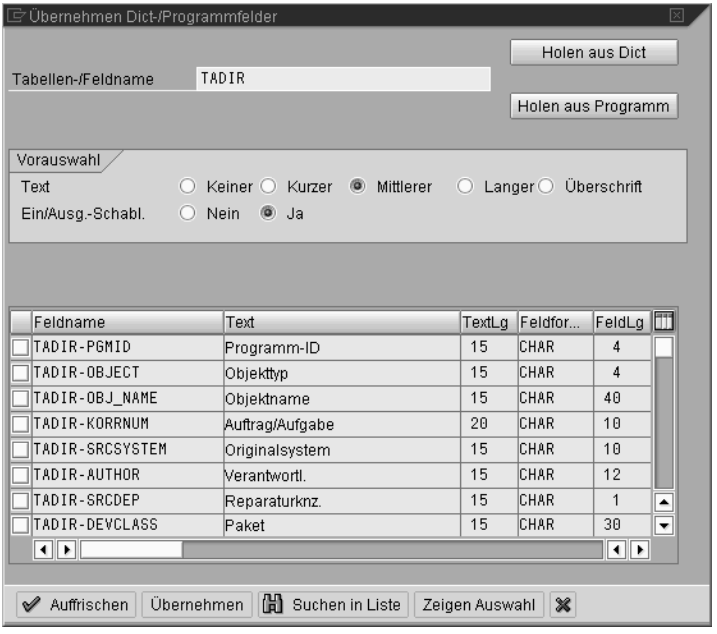

Abbildung 3.106 **© SAP AG**
Einfügewerkzeug für Tabellenfelder

Schlüsselfelder und die nachfolgend beschriebenen Feldschablonen werden am einfachsten gemeinsam mit einem speziellen Werkzeug eingefügt. Dieses Werkzeug ist mit der Menüfunktion SPRINGEN | DICT./PROGRAMMFELDER erreichbar. Dieses Werkzeug führt den Dialog über ein Popup (Abbildung 3.106).

Tragen Sie im Eingabefeld TABELLEN-/FELDNAME den Namen der Dictionary-Tabelle (oder Struktur) ein, aus der Sie Felder übernehmen möchten. Betätigen Sie dann die Schaltfläche HOLEN AUS DICT. Die Tabelle im unteren Teil des Popups listet nun alle Felder der Tabelle auf. Sie können ein oder mehrere Felder durch einen Klick auf die Checkbox in der ersten Tabellenspalte markieren. Falls sich im Dynpro bereits Felder dieser Tabelle befinden, so sind diese Felder in der Tabelle enthalten, aber deaktiviert. Im Rahmen VORAUSWAHL können Sie nun die Art des Schlüsselfeldes festlegen und bestimmen, ob eine Feldschablone eingefügt werden soll. Danach übernehmen Sie die ausgewählten Felder mit der Schaltfläche ÜBERNEHMEN. Sie gelangen dadurch wieder in den Layout-Editor. Dort fügen Sie die Felder durch einen Doppelklick an der gewünschten Position ein. Falls Sie mehrere Felder ausgewählt haben, werden diese auf einmal als Block eingefügt. Im Dynpro muss daher ausreichend Platz für alle Felder sein.

Feldschablonen

Für die eigentlichen Eingabefelder sind so genannte Feldschablonen zu erstellen. Da über Eingabefelder Daten ein- oder ausgegeben werden sollen, müssen diese Felder wegen des Datentransfers Bezug zu einem Dictionary- oder zumindest einem programminternen Feld besitzen. Sie werden daher normalerweise ebenfalls mit der mehrfach erwähnten Eingabehilfe erzeugt.

Im Layout-Editor werden die Feldschablonen durch einige spezielle Zeichen symbolisiert. Dabei steht jedes Zeichen im Platzhalter für ein Zeichen des Eingabefeldes. Als Standardplatzhalter dient der Unterstrich. Andere Zeichen symbolisieren spezielle Eigenschaften des Dynpro-Feldes, wobei die konkrete Funktion vom Zeichen selbst und von dessen Position abhängt. Ein Fragezeichen in der ersten Position kennzeichnet beispielsweise ein Muss-Eingabefeld. Weitere Zeichen, die eine Sonderfunktion auslösen, wenn sie an der ersten Stelle der Feldschablone stehen, sind der Stern und das Ausrufungszeichen. Alle anderen Zeichen, unabhängig von ihrer konkreten Position, dienen als Maske. Das bedeutet, dass an der jeweiligen Stelle im Eingabefeld genau dieses Zeichen eingegeben werden muss. Die Prüfung erfolgt automatisch bei Abarbeitung des Dynpros in einer Anwendung. Beim Anlegen eines Feldes mittels des Einfügeprogramms wird eine Standard-Feldschablone erzeugt, die nur aus Unterstrichen besteht. Sollen Maskenzeichen eingefügt werden, muss dies im Attribute-Popup im Feld FELDTEXT erfolgen. Wenn die Feldschablone beispielsweise aus den Zeichen

___-___/___

besteht, muss der Anwender nach drei eingetragenen beliebigen Zeichen ein „-"-Zeichen eingeben, nach drei weiteren beliebigen Zeichen ein „/"-Zeichen. Die korrekte Eingabe in Muss-Felder und die Eingabe von Maskenzeichen wird durch die Steuerlogik des Systems geprüft. Fehleingaben werden erkannt und per Ausschrift in der Statuszeile erläutert. Der Cursor steht dann automatisch im (ersten) fehlerhaften Feld. Werden vorhandene Daten in einem Feld mit Schablonenzeichen angezeigt, füllt die Dynpro-Logik die freien Zeichen der Maske nacheinander mit den Zeichen des Datenfeldes, es gehen also keine Daten verloren. Falls ein Feld den Inhalt `123456789` besitzt, würde es mit obiger Schablone in der Form `123-456/789` angezeigt und dann auch so abgespeichert werden.

Auch Feldschablonen können manuell erzeugt werden. Dazu werden im Dynpro die Platzhalterzeichen eingetragen. Neben dem Unterstrich kann die Feldschablone auch mit dem Fragezeichen beginnen. Dadurch wird das Feld zum Muss-Eingabefeld. Maskenzeichen können ebenfalls sofort eingefügt werden. Die Zeichenkette im Dynpro kann bis zur Auswahl einer Funktion im Layout-Editors oder Betätigung der ↵-Taste bearbeitet werden. Erst dann wird sie durch den Layout-Editor erkannt und in ein echtes Dynpro-Feld umgewandelt. Alle weiteren Attribute für das Feld müssen dann im Attribute-Popup oder den verschiedenen Varianten der Feldlistenbearbeitung gesetzt werden. Dies betrifft insbesondere die Verbindung mit einem Datenfeld.

Auswahlknöpfe

Auswahlknöpfe, allgemein oft auch als Radiobuttons bezeichnet, sind Eingabeelemente, die lediglich eine Ja-Nein-Information aufnehmen können. Entweder ist ein Auswahlknopf selektiert oder nicht. Mehrere Auswahlknöpfe werden zu einer Gruppe zusammengefasst. Innerhalb einer Gruppe kann immer nur ein Auswahlknopf markiert sein. Eine Gruppe ist nicht mit einem umgebenden Rahmen identisch. Es handelt sich vielmehr um eine Dynpro-interne Zusammenfassung, die visuell nicht erkennbar ist. Die Zusammengehörigkeit der Felder kann allerdings durch einen zusätzlichen Rahmen hervorgehoben werden. Funktionell notwendig ist er nicht.

Zu jedem Auswahlknopf gehört ein Datenfeld im Programm. Ist der Auswahlknopf selektiert, steht im zugehörigen Datenfeld ein „X"-Zeichen, ansonsten ein Leerzeichen. Das zugehörige Datenfeld hat daher den Typ C und die Länge 1.

Auswahlknöpfe können nicht direkt erstellt werden. Für jeden Auswahlknopf müssen Sie zunächst eine Feldschablone erzeugen und durch einen Eintrag in der Feldliste des Dynpros mit einem Datenfeld verbinden. Fügen Sie dazu die drei programminternen Felder `rb1` bis `rb3` in das Dynpro ein, wie in Bild 3.107 dargestellt.

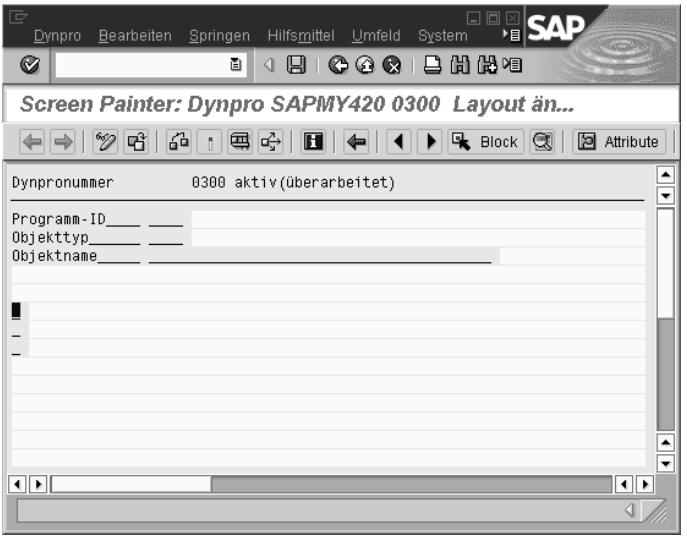

Abbildung 3.107 **© SAP AG**
Ausgangspunkt für die Erstellung von Radiobuttons

Das Einfügen der Felder kann entweder mit dem bereits hinreichend beschriebenen Einfügewerkzeug oder aber manuell erfolgen. Um die Felder manuell anzulegen, tragen Sie zunächst die drei Unterstriche „_“ im Dynpro ein und weisen dann mittels des Attribute-Popups die entsprechenden Feldnamen zu.

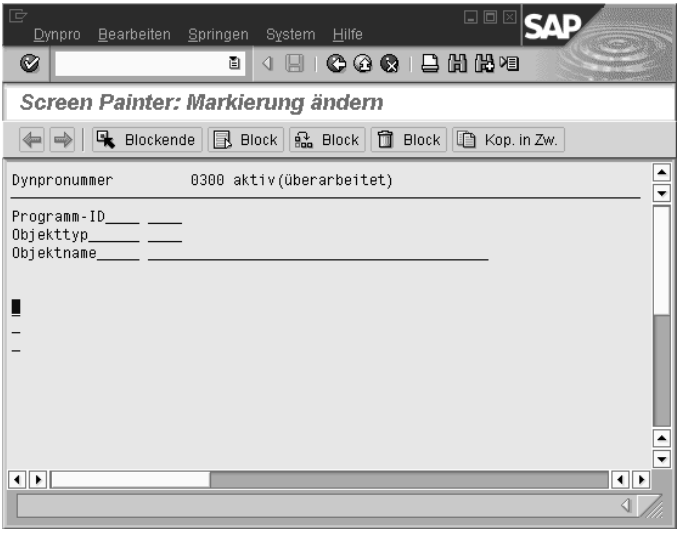

Abbildung 3.108 **© SAP AG**
Markieren des Blockbeginns

Bei der Übernahme der Felder mit Hilfe des Einfügewerkzeuges können Sie diese nur am linken Rand des Dynpros einfügen. Sie müssen die Felder daher noch innerhalb des Dynpros verschieben. Führen Sie dazu einen Doppelklick in das oberste Feld aus. Dadurch wird das Feld markiert (Abbildung 3.108).

Ein weiterer Doppelklick auf das unterste Feld markiert den gesamten, hier aus drei Feldern bestehenden Block (Abbildung 3.109). Die horizontale Position des Blockendes muss natürlich nicht direkt unter dem Beginn liegen. Sie können auch echte rechteckige Blöcke markieren, z.B. solche, die aus Feldschablonen und Schlüsselfeldern bestehen.

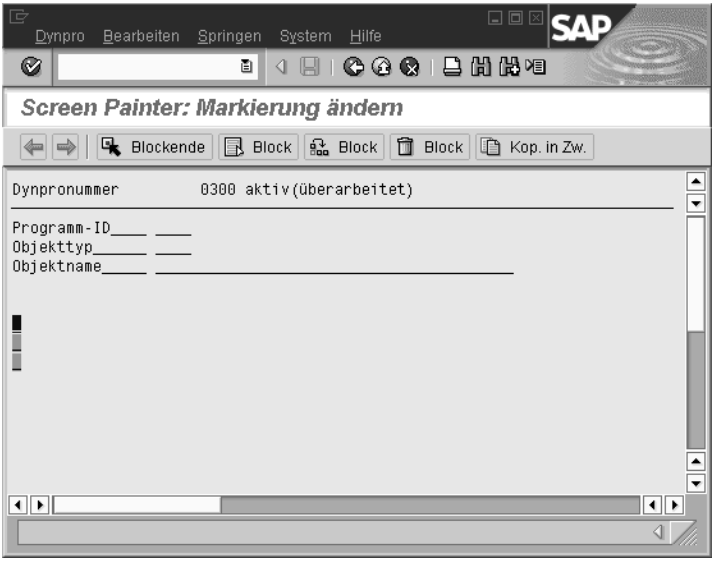

Abbildung 3.109 **© SAP AG**
Blockende festlegen.

Der zu kopierende Block ist markiert. Sie können den Cursor nun auf die Zielposition stellen (Abbildung 3.110) und den Block durch die Schaltfläche BLOCK VERSCHIEBEN bzw. die Tastenkombination ⇧ F1 an diese Position verschieben.

Der Layout-Editor verschiebt die Felder. Sie befinden sich danach wieder im Grundbild des Editors (Abbildung 3.111) und können weitere Elemente bearbeiten.

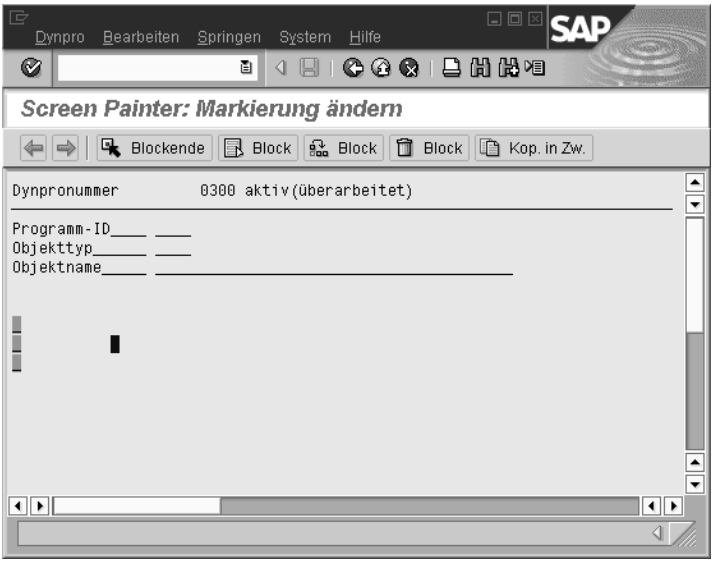

Abbildung 3.110 **© SAP AG**
Festlegen der Zielposition

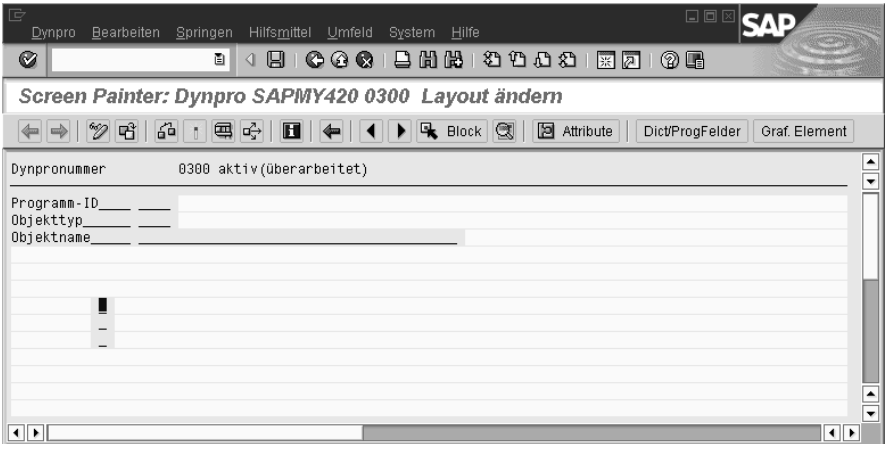

Abbildung 3.111 **© SAP AG**
Endergebnis des Verschiebevorgangs

Nun können Sie die einfachen Eingabefelder in Auswahlknöpfe umwandeln. Dazu platzieren Sie den Cursor auf das linke obere Dynpro-Feld. Es ist egal, ob es sich dabei um ein Schlüsselfeld oder eine Feldschablone handelt. Dann betätigen Sie die Drucktaste GRAFISCHES ELEMENT.

Der Fullscreen-Editor wechselt daraufhin die Ansicht (Abbildung 3.112). Das neue Dynpro dient zur Bearbeitung der so genannten grafischen Elemente, zu denen unter anderem die Auswahlknöpfe gehören. In diesem neuen Bild können Sie die Elemente nur noch über die verschiedenen Funktionen des Screen Painters und nicht durch manuelle Einträge im Fullscreen bearbeiten.

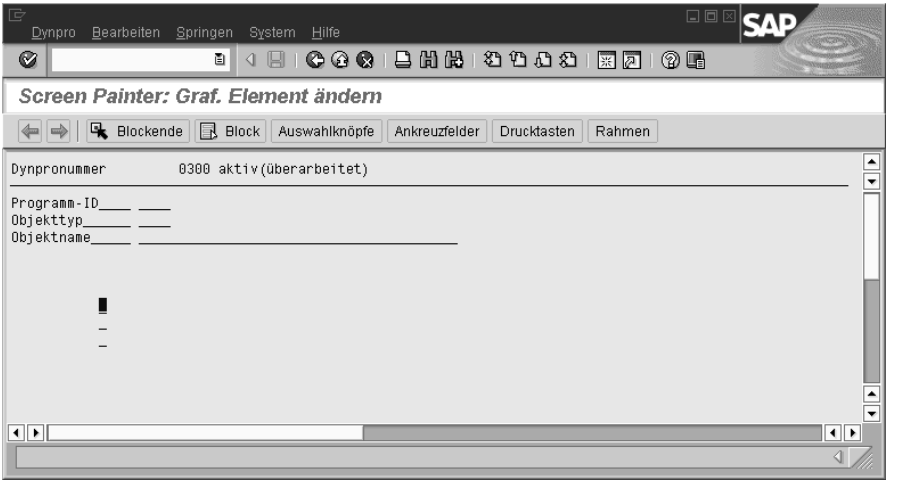

Abbildung 3.112 **© SAP AG**
Grafische Elemente definieren

Das Feld, auf dem vorher der Cursor platziert war, ist farbig hervorgehoben. Der Cursor wird nun auf die rechte untere Ecke der Gruppe der Auswahlknöpfe platziert. Ein nachfolgender Druck auf die Drucktaste BLOCKENDE MARKIEREN oder ein Doppelklick fasst alle Elemente zu einem Block zusammen, auf den sich dann die nachfolgende Funktion bezieht. Diese nächste Funktion besteht in der Betätigung der Drucktaste AUSWAHLKNÖPFE. Sie verwandelt alle Feldschablonen im markierten Bereich in Auswahlknöpfe. Diese werden im Fullscreen-Editor durch ein entsprechendes Symbol dargestellt (siehe Abbildung 3.113).

Die Ansicht des Fullscreen-Editors, genauer gesagt: der Status und damit die verfügbaren Funktionen, wechseln nach Erzeugen der Auswahlknöpfe nochmals. Die Auswahlknöpfe müssen noch zu einer Gruppe verbunden werden, wodurch sichergestellt wird, dass nur eines der Felder markiert werden kann. Die dazu erforderliche Funktion ist über die Drucktaste GRAFISCHE GRUPPE DEFINIEREN erreichbar. Sie bezieht sich auf alle Felder, die im vorher markierten Block enthalten sind.

Sie können diese Funktionen unmittelbar nacheinander ausführen, weil die Felder nach dem Umwandeln in Auswahlfelder weiterhin markiert sind. Anderenfalls müssen Sie vor dem Definieren der grafischen Gruppe zunächst alle Felder markieren.

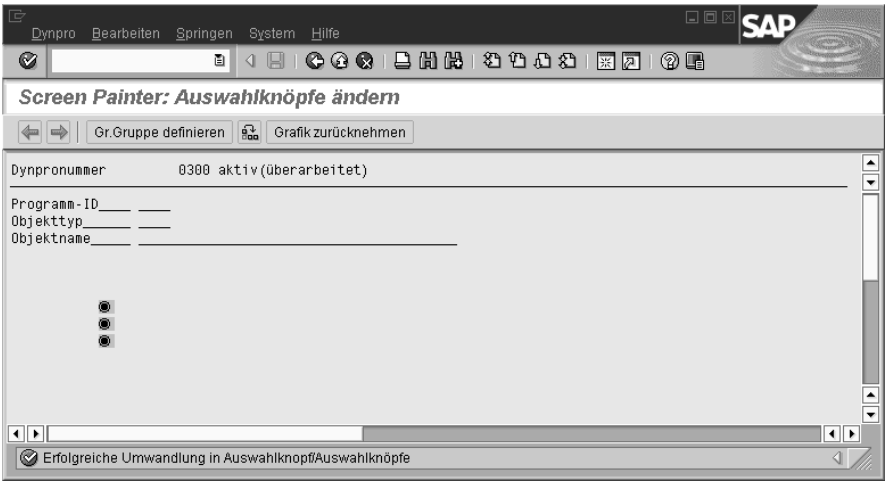

Abbildung 3.113 **© SAP AG**
Das Dynpro mit drei Auswahlfeldern

Die Auswahlfelder müssen nicht zwangsweise direkt untereinander stehen. Es ist auch möglich, sie in mehreren Spalten anzuordnen oder zwischen den einzelnen Feldern Leerzeilen einzufügen. Es ist nicht erforderlich, alle Felder gemeinsam in Auswahlfelder umzuwandeln, dies kann durchaus auch feldweise erfolgen. Der Fullscreen-Editor erfordert dabei aber, dass nach jeder Definition eines grafischen Elements das Werkzeug für grafische Elemente verlassen wird. Die grafische Gruppe ist dann nach Definition aller Auswahlknöpfe zu erzeugen. Dazu wird der Cursor wieder auf das Element platziert, das sich am weitesten links oben befindet. Mittels der Drucktaste GRAFISCHES ELEMENT wird zunächst das Werkzeug zur Bearbeitung der grafischen Elemente aufgerufen. Da sich der Cursor bereits auf einem grafischen Element befindet, ist die Funktion zum Definieren der Gruppe sofort verfügbar. Der Cursor wird nun so weit rechts unten platziert, dass das dadurch festgelegte Rechteck alle Felder einschließt, die in die Gruppe aufgenommen werden sollen. Dann wird die Gruppe mit der entsprechenden Drucktaste erzeugt.

Um eine Gruppe aufzulösen oder einen Auswahlknopf wieder in eine einfache Feldschablone zu verwandeln, reicht es aus, den Cursor auf das jeweilige Element zu platzieren und erneut die Drucktaste GRAFISCHES ELEMENT zu betätigen. Im darauf folgenden Dynpro des Fullscreen-Editors werden dann nur die Funktionen angeboten, die für das markierte Element verfügbar sind. Der Fullscreen-Editor arbeitet also kontextsensitiv oder objektorientiert. Die Zurücknahme der jeweiligen Eigenschaft (Gruppierung oder Auswahlknopf) erfolgt über GRUPPE AUFLÖSEN bzw. GRAFIK ZURÜCKNEHMEN.

Ankreuzfelder

Ankreuzfelder ähneln den Auswahlknöpfen, allerdings bestehen keine Abhängigkeiten der Ankreuzfelder untereinander. In einem Dynpro können beliebig viele Ankreuzfelder markiert werden. Es ist deshalb nicht notwendig, die Ankreuzfelder zu einer Gruppe zusammenzufassen. Ankreuzfelder müssen ebenfalls mit je einem Datenfeld verbunden werden. Für dessen Eigenschaften gelten dieselben Bedingungen wie bei den Auswahlknöpfen (Typ Char, Länge 1). Für jedes markierte Ankreuzfeld wird ein „X“ im Datenfeld eingetragen.

Ankreuzfelder werden auf dieselbe Weise erzeugt wie Auswahlknöpfe, mit dem Unterschied, dass die Drucktaste ANKREUZFELDER anstelle von AUSWAHLKNÖPFE benutzt wird. Im Fullscreen-Editor unterscheiden sich die Ankreuzfelder von den Auswahlknöpfen durch das Platzhaltersymbol, ein kleines Quadrat mit eingezeichneten Diagonalen (siehe Abbildung 3.114).

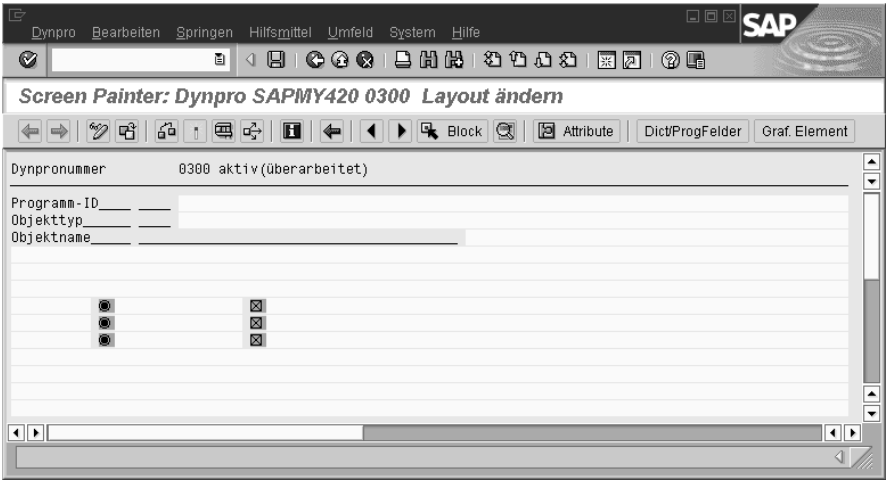

Abbildung 3.114 **© SAP AG**
Dynpro mit drei Ankreuzfeldern

Rahmen

Ein Rahmen dient lediglich zur optischen Abgrenzung von Feldern, er besitzt keine eigene Funktionalität. Er kann in einem Dynpro nur unter Verwendung eines Dynpro-Feldes erstellt werden, wobei dieses Feld keine Eingabemöglichkeit besitzen darf. Das bedeutet, dass sowohl Schlüsselfelder als auch Feldschablonen, bei denen das Attribut EINGABEFELD zurückgenommen wurde, als Ausgangspunkt für einen Rahmen dienen können. Die Verwendung von Feldschablonen ermöglicht es, die Überschrift des Rahmens dynamisch im Programm zu setzen.

Um einen Rahmen zu erzeugen, muss zunächst ein Dynpro-Feld angelegt werden. Dieses Feld kann durchaus Bezug zu einem Dictionary-Feld haben, dies ist

aber nicht Bedingung. Sollte das Dynpro-Feld auf einem Dictionary-Feld beruhen, kann auf einfache Weise eine (statische) Bezeichnung aus dem Dictionary übernommen werden. Sollte es nicht möglich sein, dem Dictionary einen geeigneten Text zu entnehmen, kann das Schlüsselfeld für den Rahmen auch manuell durch Eintragen der gewünschten Bezeichnung im Fullscreen-Editor angelegt werden. Bei der Platzierung des Feldes ist zu beachten, dass die linke obere Ecke des Rahmens unmittelbar links neben dem Dynpro-Feld erzeugt wird. Die Überschrift ist damit in der Kopfzeile des Rahmens immer linksbündig, andere Ausrichtungen sind nicht möglich. Abbildung 3.115 zeigt das Dynpro mit einem Text, der als Ausgangspunkt für einen Rahmen dient.

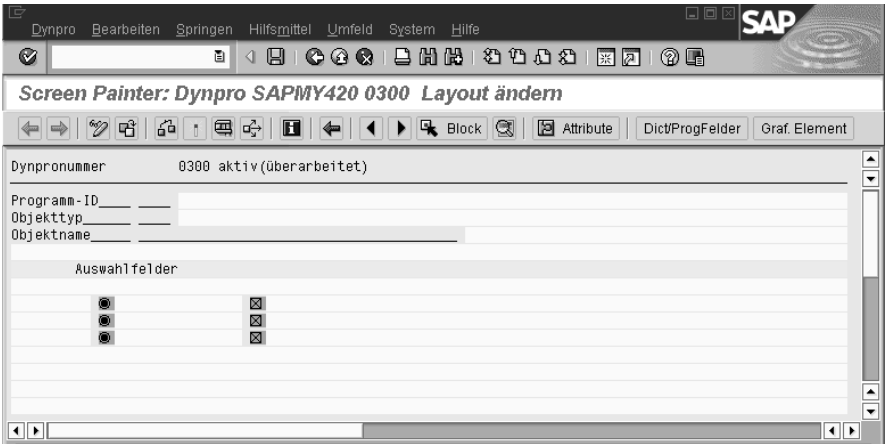

Abbildung 3.115 **© SAP AG**
Dynpro mit Rahmenüberschrift

Ist das Feld für die Bezeichnung des Rahmens vorhanden, wird der Cursor auf das Feld platziert und die Drucktaste GRAFISCHES ELEMENT ausgelöst. Im nachfolgenden Dynpro können Sie mit der Drucktaste RAHMEN die Erstellung des Rahmens einleiten. Der Status wechselt erneut, was an der geänderten Drucktastenleiste sofort erkennbar ist. Nun markieren Sie die rechte untere Ecke des Rahmens durch Positionieren des Cursors (Abbildung 3.116).

Der Rahmen wird dann entweder durch die Drucktaste ENDE RAHMEN oder durch einen Doppelklick mit der Maus erzeugt.

Der Rahmen darf keine anderen Dynpro-Felder berühren oder mit seinen Seitenkanten schneiden. Derartige Fehler erkennt der Layout-Editor und weist sie zurück. Im Layout-Editor wird der Rahmen durch farblich hinterlegte Linien symbolisiert (Abbildung 3.117).

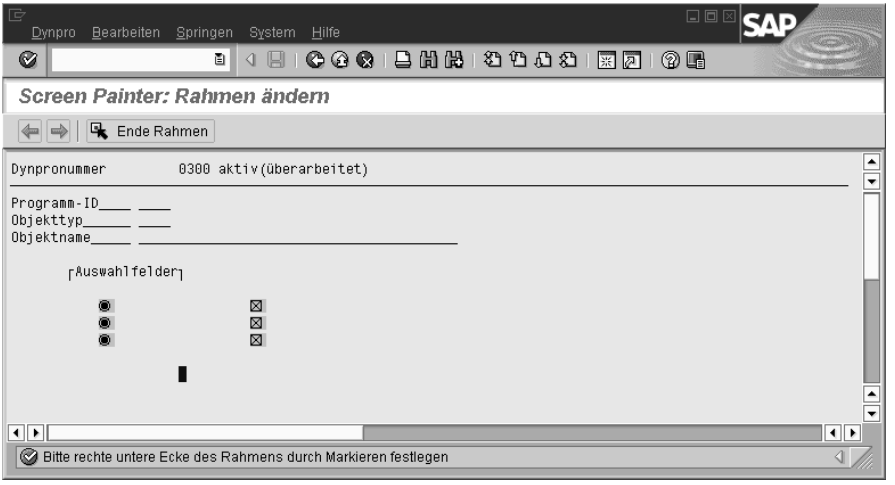

Abbildung 3.116 **© SAP AG**
Festlegen der Rahmenausdehnung

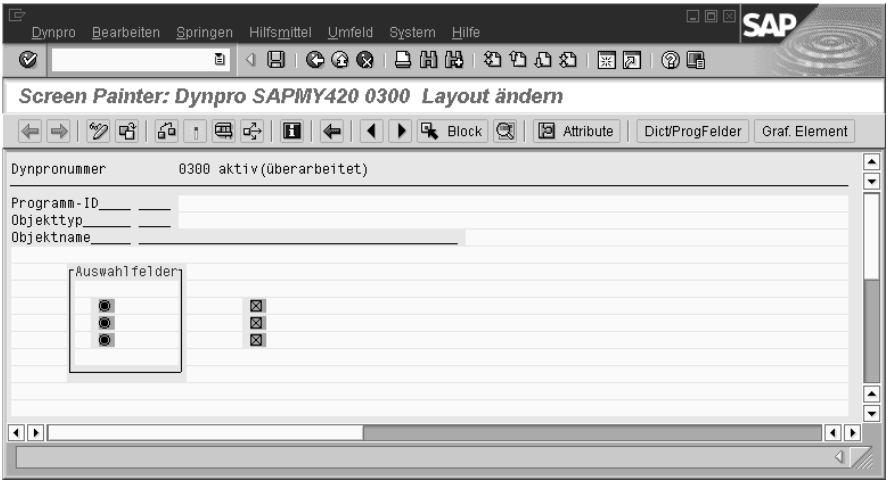

Abbildung 3.117 **© SAP AG**
Dynpro mit Rahmen

Drucktasten

Neben den Drucktasten in der Drucktastenzeile der Oberfläche können auch innerhalb des Dynpros derartige Bedienelemente angelegt werden. Die Funktionscodes, die von Dynpro-Drucktasten ausgelöst werden, sind denen der Oberfläche völlig gleichwertig. Diese Dynpro-Drucktasten werden wie ein Rahmen von einem bestehenden Dynpro-Feld abgeleitet. Dabei kann es sich wieder-

um um ein Schlüsselfeld oder eine Feldschablone ohne Eingabeberechtigung handeln. Letztere Variante ermöglicht das dynamische Setzen des in der Drucktaste angezeigten Textes.

Die Erzeugung einer Drucktaste verläuft ähnlich wie die der bereits beschriebenen Elemente. Es wird ein entsprechendes Dynpro-Feld angelegt, der Cursor wird auf dieses Feld platziert und die Drucktaste GRAFISCHES ELEMENT betätigt. Im nachfolgenden Dynpro steht dann die Drucktaste DRUCKTASTE zur Verfügung, um das Feld in eine solche zu verwandeln. Im allgemeinen Bild des Fullscreen-Editors müssen Sie der Drucktaste dann noch einen Funktionscode und einen Funktionstyp zuweisen. Dies erfolgt im Attribute-Popup für das entsprechende Element. Der Aufbau und die Felder dieses Popups werden nach der Beschreibung der diversen Elemente erläutert.

Drucktasten können mit einem Icon verbunden werden. Dieses kann zusätzlich zum Text eingefügt werden; es ist aber auch möglich, Drucktasten nur mit einem Icon zu versehen und auf den Text zu verzichten.

Beim Zuweisen von Icons ist der Typ des Dynpro-Feldes zu berücksichtigen. Für Schlüsselfelder kann ein Icon direkt im Attribute-Popup über das Attribut ICON NAME zugewiesen werden. Die Eingabehilfe für dieses Attribut bietet alle möglichen Icons und deren Bezeichnung zur Auswahl an. Nähere Informationen dazu finden Sie im Abschnitt über die Attribute von Dynpro-Feldern.

Subscreen

Bei den Dynpro-Attributen wurde bereits auf die Subscreens hingewiesen. Ein Dynpro mit diesem Typ kann zur Laufzeit der Anwendung dynamisch als Include in ein anderes Dynpro eingebunden werden. Dazu muss das übergeordnete Dynpro einen speziellen Bereich für das Subscreen-Dynpro reservieren. Dieser Bereich wird der Einfachheit halber ebenfalls als Subscreen bezeichnet. In einem Dynpro können mehrere solcher Subscreens eingefügt werden. Sie erhalten eine eindeutige Bezeichnung. Subscreens ermöglichen dem Anwender auf einfache Weise das Einfügen von eigenen Funktionen in SAP-Standardprogramme, falls diese durch Einbinden eines solchen Subscreens eine derartige Erweiterung überhaupt ermöglichen. Außerdem werden Subscreens oft benutzt, um in einem allgemein gehaltenen Rahmenprogramm durch Einblenden spezialisierter Subscreens unterschiedliche Objekte pflegen zu können.

Ein Subscreen wird durch die Menüfunktion BEARBEITEN | ANLEGEN ELEMENT | SUBSCREEN erzeugt. Ausgehend von der aktuellen Cursorposition wird der gesamte freie Platz im Dynpro markiert. Abbildung 3.118 zeigt den Layout-Editor zu diesem Zeitpunkt.

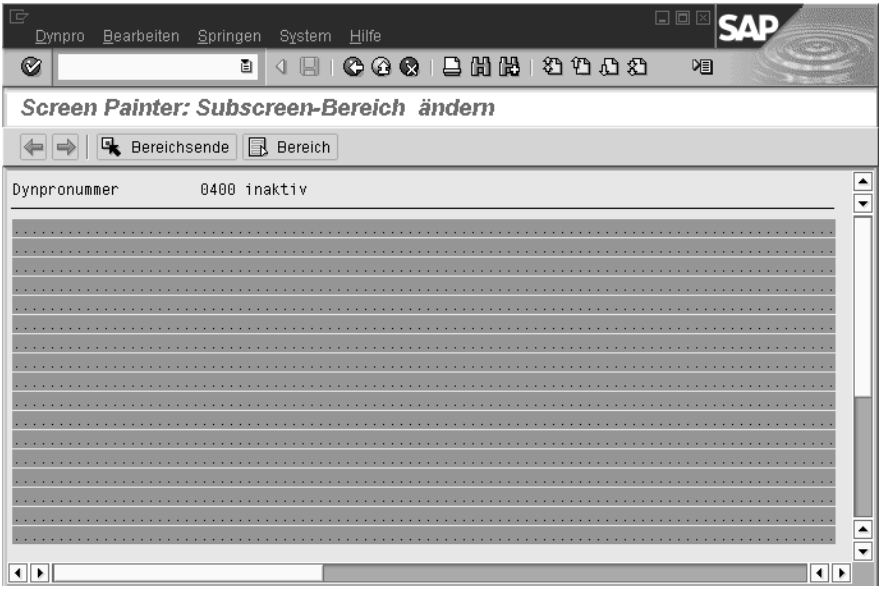

Abbildung 3.118 **© SAP AG**
Einfügen eines Subscreens

Ein Doppelklick auf einen beliebigen Punkt innerhalb des markierten Bereichs legt die Ausdehnung des Subscreen-Bereichs fest. Unmittelbar nach dem Doppelklick erscheint ein Popup (Abbildung 3.119), in dem die Attribute, insbesondere der Name des Subscreen-Bereiches gepflegt werden.

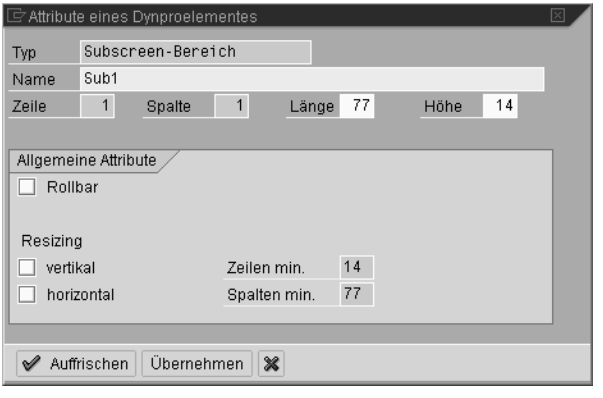

Abbildung 3.119 **© SAP AG**
Attribute für den Subscreen-Bereich

Das Anlegen des Subscreen-Bereiches ist damit abgeschlossen. Allerdings sind spezielle Anweisungen in der Ablauflogik erforderlich, um in den Subscreen-Bereich auch wirklich ein Dynpro einblenden zu können. Details dazu finden Sie im Abschnitt über die Kommandos der Ablauflogik.

Step Loops

Die bisher beschriebenen Dynpro-Elemente sind dazu gedacht, immer nur einen Datensatz gleichzeitig zu bearbeiten. Dies wirkt sich mitunter negativ auf die Ergonomie und die Geschwindigkeit einer Anwendung aus. So genannte Step Loops oder kurz Loops erlauben es, mehrere Datensätze gleichzeitig darzustellen und zu bearbeiten. Loops erfordern außer der Generierung entsprechender Elemente im Dynpro spezielle Anweisungen in der Ablauflogik. Details zur Programmierung finden Sie im weiteren Verlauf dieses Abschnitts.

Um einen Loop zu erzeugen, wird zunächst ein Block im Layout-Editor angelegt. Ein solcher Block besteht aus einer oder mehreren Feldschablonen. Theoretisch können auch Textfelder enthalten sein, in der Praxis macht man davon aus ergonomischen Gründen aber selten Gebrauch. Der Block kann mehrere Zeilen umfassen, wobei in der Praxis fast immer einzeilige Blöcke benutzt werden. Dieser Block ist die Vorlage für den Aufbau des Loops, der aus einer mehrfachen Wiederholung des Blocks besteht. Abbildung 3.120 zeigt eine vorbereitete Zeile für einen Step Loop.

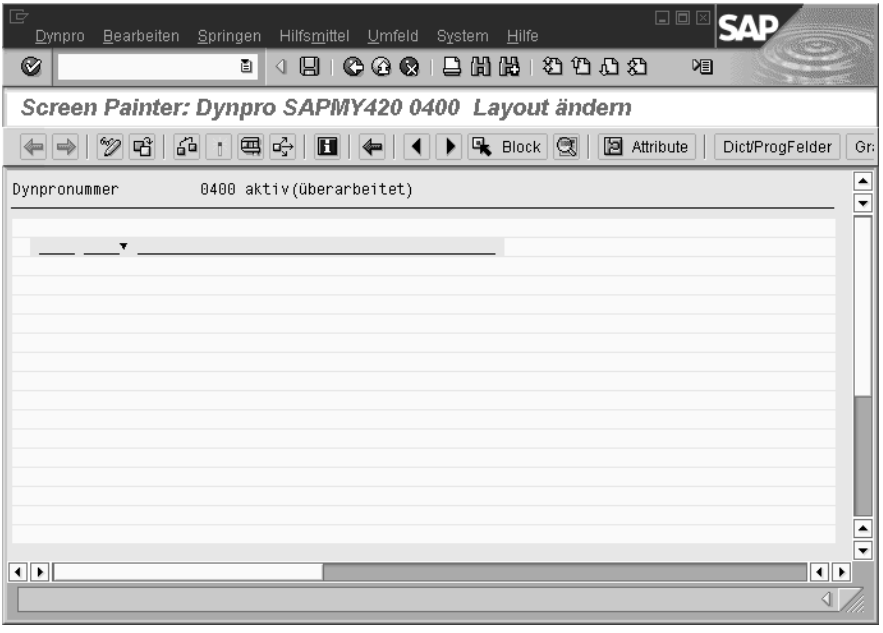

Abbildung 3.120 **© SAP AG**
Ausgangspunkt zum Anlegen eines Step Loops

Nach Gestalten des Blocks wird der Cursor auf das am weitesten links oben stehende Feld des Blocks platziert und die Menüfunktion BEARBEITEN | LOOP ausgewählt. Der Fullscreen-Editor wechselt in einen anderen Bearbeitungsmodus, in dem das Ende des Vorlageblocks, also das am weitesten rechts unten stehende Feld des zu vervielfältigenden Blocks markiert werden muss (Doppelklick an der gewünschten rechten unteren Ecke oder Platzieren des Cursors und Drücken von ENDE LOOPBLOCK). Abbildung 3.121 zeigt den Layout-Editor zu diesem Zeitpunkt.

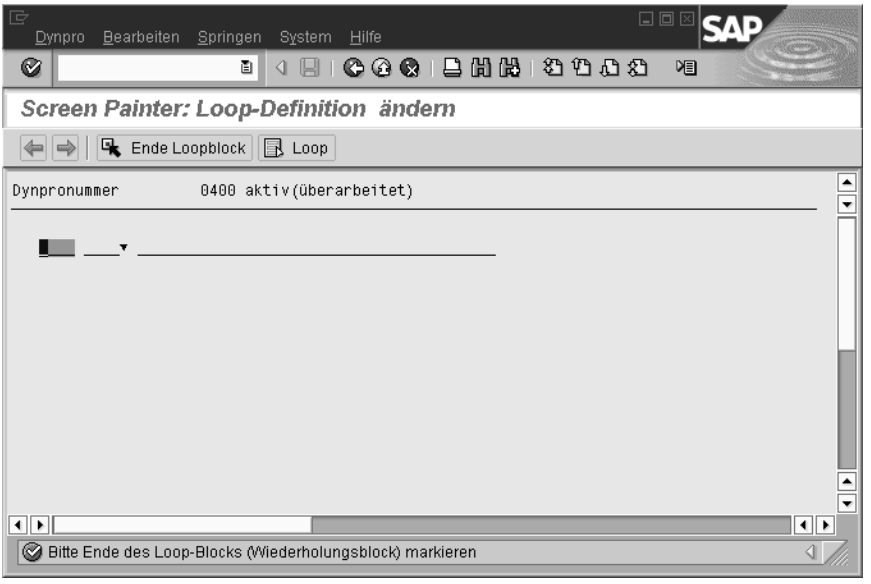

Abbildung 3.121 **© SAP AG**
Ende eines Loop-Blocks bestimmen

Nach der Definition des Blockendes erzeugt der Screen Painter automatisch eine weitere Wiederholung des markierten Blocks direkt unter diesem (siehe Abbildung 3.122). Sowohl der Vorlageblock als auch die Kopie werden in einer eigenen Farbe hervorgehoben. Der Fullscreen-Editor verbleibt im Bearbeitungsmodus für Loops, stellt aber einen neuen Status mit einigen neuen Funktionen bereit.

Ein Loop wird zunächst als so genannter *fixer Loop* angelegt. Ein fixer Loop erscheint auf dem fertigen Dynpro immer in der Größe, mit der er im Layout-Editor definiert wurde. Da der Screen Painter zunächst nur einen aus zwei Blöcken bestehender Loop erzeugt, können Sie mit der Drucktaste LOOP ENDZEILE den Loop-Bereich bis zur aktuellen Cursorposition ausweiten. Dazu wird der Cursor in die gewünschte Endzeile positioniert und dann die erwähnte Drucktaste betätigt.

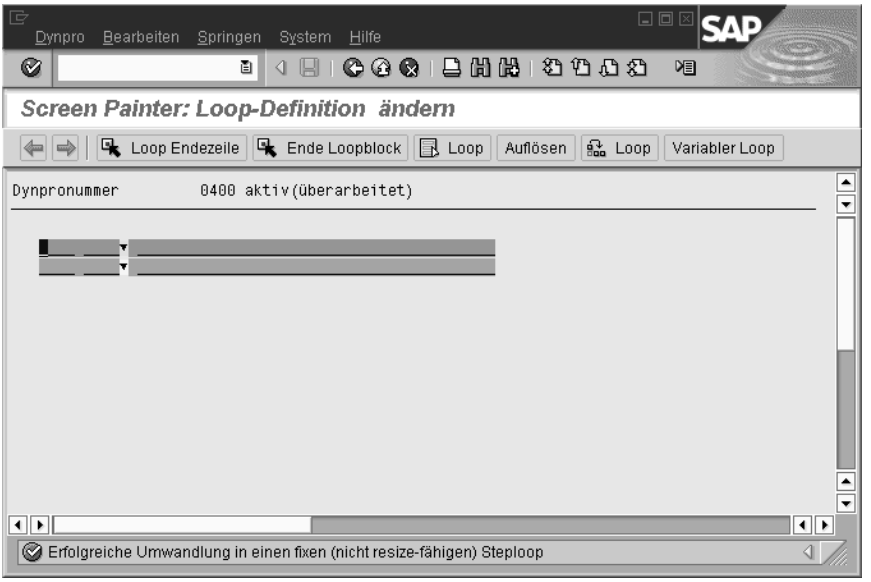

Abbildung 3.122 **© SAP AG**
Rudimentärer Step Loop mit zwei Blöcken

Eine weitere Drucktaste bzw. Menüfunktion erlaubt die Umwandlung in einen *variablen Loop*. Beim Verändern der Bildschirmgröße passt sich die Zahl der sichtbaren Wiederholungen an die aktuelle Fenstergröße an. Bei dieser Veränderung der Zeilenzahl bleiben die Abstände zu anderen Dynpro-Feldern über und unter dem Loop-Bereich stets erhalten. Sollte also zwei Zeilen unterhalb eines variablen Loops ein weiteres Feld eingefügt werden, bleiben auch bei Veränderungen der Fenstergröße unter dem Loop diese zwei Zeilen frei. Sollte um einen variablen Loop ein Rahmen gelegt werden, verändert sich die Größe des Rahmens zusammen mit der Größe des Loops.

Eine Besonderheit ist beim Positionieren der Schlüsselfelder für einen Loop zu beachten: Falls die Schlüsselfelder zusammen mit den Feldschablonen innerhalb des Loop-Blocks liegen, werden sie automatisch in jedem Folgeblock wiederholt. Dies stört oft die Übersichtlichkeit des Dynpros. Schlüsselfelder können daher auch außerhalb des eigentlichen Loops in Form einer Tabellenüberschrift platziert werden. Dann befinden sich Schlüsselfelder und Feldschablonen in unterschiedlichen Dynpro-Bereichen, was durch den Screen Painter nicht zugelassen wird. Beim Generieren des Dynpros erscheint eine entsprechende Fehlermeldung. Für diesen Fall ist die Bezugstabelle für die Schlüsselfelder unter einem zweiten Tabellennamen verfügbar. Dieser entsteht aus dem eigentlichen Tabellennamen durch Voranstellen eines Sterns „*“. Im Deklarationsteil der Anwendung ist diese Tabelle ebenfalls anzugeben. In diesem Fall also:

```
TABLES: TADIR, *TADIR.
```

Es bietet sich dabei an, die Schlüsselfelder und die Feldschablonen durch zwei getrennte Aufrufe des Einfügewerkzeugs in das Dynpro zu übernehmen. Aber auch die manuelle Änderung der Feldnamen in der Feldliste ist möglich.

Table Control

Ab Release 3.0 erleichtern *Controls* die Darstellung und Bearbeitung von Daten in Dynpros. Zu den Controls gehört der so genannte *Table Control*. Das ist eine moderne Variante der Step Loops. Table Controls bieten einen Tabellenarbeitsbereich an, der nicht mehr wie ein Loop aus einzelnen optisch voneinander getrennten Dynpro-Feldern besteht, sondern als in sich geschlossenes Objekt erscheint. Die Table Controls haben inzwischen die herkömmlichen Step Loops fast vollständig abgelöst. Sie sind an vielen Stellen der Entwicklungsumgebung zu finden. Beispielsweise sind die Tabellen innerhalb der Dynpro-Elementeliste (Abbildung 3.123) derartige Table Controls.

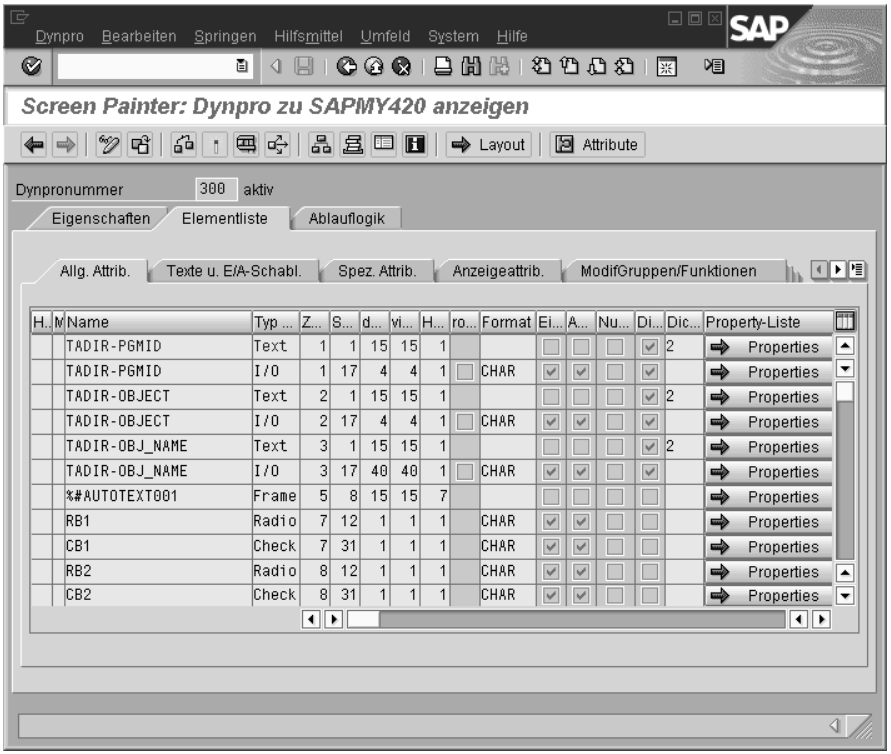

Abbildung 3.123 **© SAP AG**
Table Control zur Darstellung der Elementliste eines Dynpros

Die Verwendung von Table Views erfordert spezielle Anweisungen in der Ablauflogik, die denen für Step Loops ähneln. Erzeugt wird ein Table View ähnlich wie ein Loop mittels einer Drucktaste oder der Menüfunktion BEARBEITEN | ANLEGEN ELEMENT | TABLE CONTROL. Ab Version 6.x steht außerdem ein Wizard zum Anlegen eines Table Controls zur Verfügung. Dieser Wizard fragt in mehreren Dialogschritten alle Einstellungen zum Aufbau eines neuen Table Controls ab. Die herkömmliche Variante ist aber flexibler und ermöglicht intensivere Einblicke in Aufbau und Funktionalität der Table Controls. Auf den Wizard soll daher nicht näher eingegangen werden.

Im Dynpro wird vom Layout-Editor ein Bereich für den Table Control reserviert, dessen konkrete Abmessung durch einen Doppelklick an der gewünschten Position der rechten unteren Ecke bestätigt werden muss. Daraufhin ist in einem Popup (Abbildung 3.124) zunächst der Name des Table Views zu erfassen.

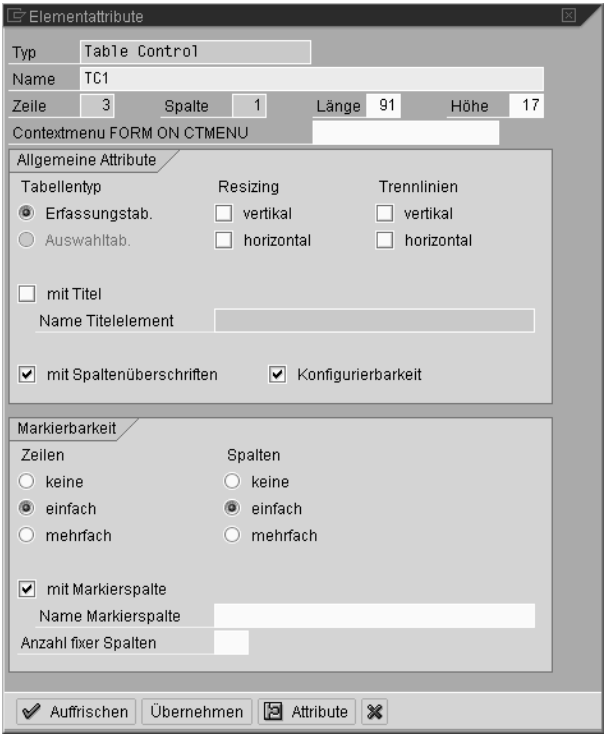

Abbildung 3.124 **© SAP AG**
Popup zur Pflege der Werte für ein Table Control

Im Modul-Pool muss später mit der Anweisung

```
CONTROLS tableview TYPE TABLEVIEW USING SCREEN dynpronumber.
```

ein gleichnamiges Datenobjekt mit dem Typ TABLEVIEW erzeugt werden. Da bei der Deklaration eines Table Views das Dynpro angegeben werden muss, kann ein Table View immer nur in einem Dynpro verwendet werden.

Nach Zuweisen des Namens kann der Table View im Layout-Editor bearbeitet werden. Die in diesem Bereich anzuzeigenden Felder werden mit der Drucktaste CTRL ELEMENTE eingefügt. Es erscheint das in Abbildung 3.125 dargestellte Popup, in dem die Feldnamen entweder manuell eingetragen oder mit der mehrfach erwähnten Eingabehilfe für Dictionary-Felder eingefügt werden. Bei der Benutzung der Eingabehilfe ist eine kleine Unstimmigkeit zu beachten: Um die ausgewählten Felder in den Table View zu übernehmen, ist im Popup für die Table View-Felder die Drucktaste EINSETZEN zu benutzen und nicht, wie man vermuten könnte, die Taste ÜBERNEHMEN. Die EINSETZEN-Drucktaste ist nur als Symbol verfügbar und wird daher leicht übersehen.

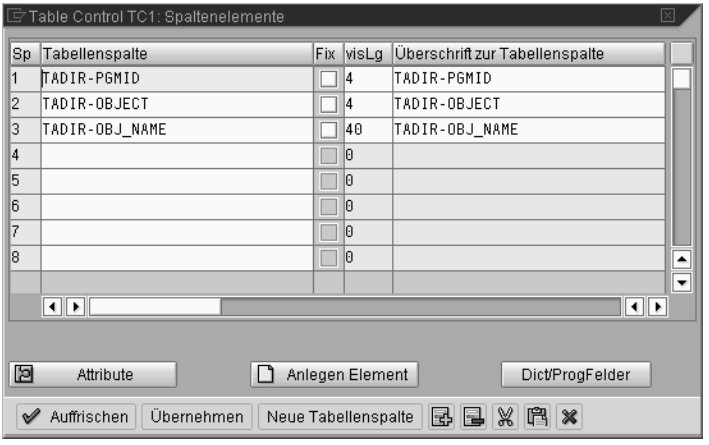

Abbildung 3.125 **© SAP AG**
Popup für Felder im Table View

Tab Strip

Mit Version 4.0 wurde ein weiteres Dynpro-Control zur Verfügung gestellt. Es wird als *Tab Strip* bezeichnet. Ein Tab Strip besteht aus einer Reihe so genannter Reiter und einem oder mehreren Subscreen-Bereichen. Durch Betätigen eines der Reiter schalten Sie den Inhalt des Subscreen-Bereiches um. Auf diese Weise können Sie in den Subscreen-Bereich eines Dynpros jeweils einen realen Subscreen einblenden und bearbeiten. Die Funktion eines Tab Strips können Sie unter älteren Versionen der R/3-Software natürlich auch durch einen herkömmlichen Subscreen-Bereich und einige Drucktasten nachbilden. Ein Tab Strip bietet Ihnen aber eine ansprechendere optische Oberfläche und eine intuitive Bedienung. Auch dieses Element ist aus der Bedienoberfläche aktueller R/3-Versionen nicht mehr wegzudenken.

Ein Tab Strip kann den oder die Subscreens mit zwei unterschiedlichen Methoden darstellen. Zunächst ist es möglich, innerhalb des gesamten Tab-Strip-Bereiches einen einzigen Subscreen-Bereich zu definieren, in den abwechselnd unterschiedliche Subscreen-Dynpros eingeblendet werden. Das Zuweisen der realen Subscreen-Dynpros zum Tab-Strip-Bereich wird dabei innerhalb der Ablauflogik der Anwendung programmiert. Für jeden Bildwechsel wird daher ein Serverzugriff notwendig.

Es ist allerdings auch möglich, beim Prozessieren eines Dynpros alle beteiligten Subscreen-Dynpros auf einmal zu verarbeiten und an den SAPGUI, also den Client, zu übertragen. Die Reiter im Tab Strip würden dann nur jeweils eines der lokal im Client verfügbaren Subscreen-Dynpros sichtbar schalten, alle anderen bleiben verborgen. Der Wechsel des Subscreen-Dynpros erfordert somit keinen Serverzugriff. Dadurch können allerdings auch keine in der Ablauflogik programmierten Prüfungen ausgeführt werden.

Die Programmierung der beiden Verhaltensweisen ist relativ einfach und wird im weiteren Verlauf des Abschnittes detailliert an einem Beispiel beschrieben.

Die Definition eines Tab-Strip-Controls ist relativ aufwändig. Ab 6.x bietet ein Wizard Unterstützung, der hier aber nicht näher beschrieben werden soll. Innerhalb der Pflegetransaktion für ein Dynpro rufen Sie die Funktion BEARBEITEN | ANLEGEN ELEMENT | TABSTRIP auf. Ähnlich wie bei Subscreens ist zunächst die rechte untere Ecke des Tableviews durch einen Doppelklick mit der Maus zu markieren. Danach erscheint ein Popup zur Namensvergabe (Abbildung 3.126).

Abbildung 3.126 **© SAP AG**
Namen für den Tab Strip festlegen

Nach dem Eintragen des Tab-Strip-Namens kehrt das Programm zum Hauptbild der Dynpro-Pflege zurück. Das System stellt einen Tab Strip ähnlich wie einen Subscreen dar (siehe Abbildung 3.127). Von diesem unterscheidet sich ein Tab Strip optisch vor allem durch die zusätzliche Zeile mit den Reitern. Vom System werden automatisch zwei solche Reiter mit den Namen `Tab1` und `Tab2` angelegt.

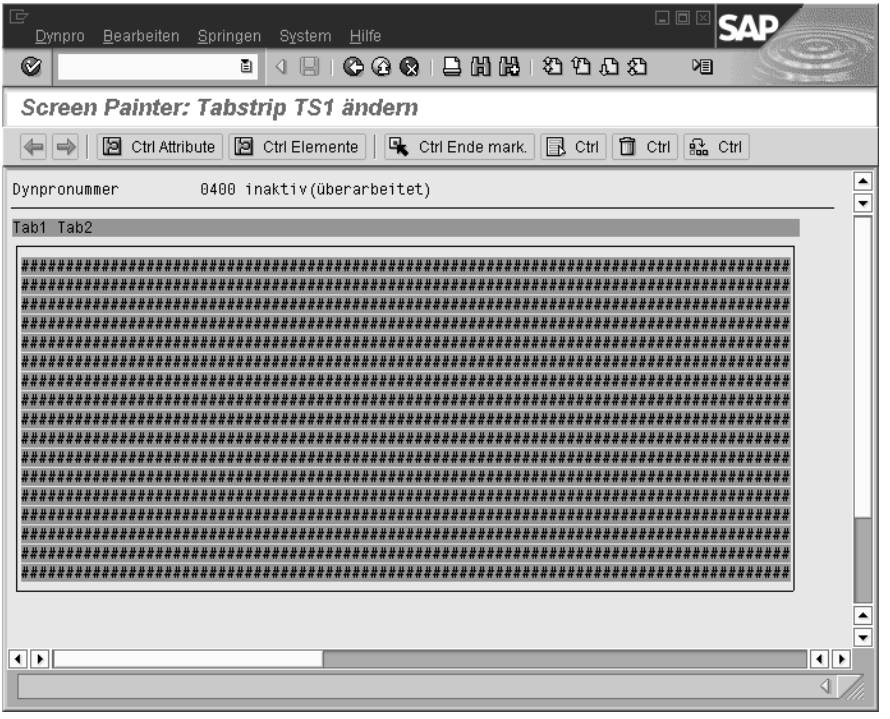

Abbildung 3.127 **© SAP AG**
Ansicht eines Tab Strips im alphanummerischen Screen Painter

Nach dem Erzeugen des Tab Strips müssen Sie die einzelnen Elemente pflegen. Insbesondere ist es erforderlich, die Reiter mit Funktionscodes zu versehen. Dazu wechseln Sie mit einem Doppelklick in den Tab-Strip-Bereich zur Pflegeoberfläche für das Control und rufen die Funktion BEARBEITEN | CTRLELEMENTE auf. Sie gelangen dadurch zu einem Popup, in dem die Tab-Reiter und die übrigen Elemente des Tab Strips gepflegt werden.

In diesem Bild (siehe Abbildung 3.128) müssen Sie den existierenden Reitern Funktionscodes zuweisen sowie den oder die zu verwendenden Subscreen-Bereiche benennen. Außerdem können Sie weitere Reiter erzeugen. Im unteren Table View tragen Sie den Namen der benutzten Subscreen-Bereiche ein. Sofern sie das Blättern zwischen den Registerkarten serverseitig programmieren, steht hier genau ein Subscreen-Name, der dann auch in der Spalte für alle Reiter eingetragen werden muss. Falls die clientseitige Blätterfunktion zum Einsatz kommen soll, muss unter jedes Subscreen-Dynpro ein eigener Subscreen-Bereich geschaffen werden. In der oberen Tabelle für die einzelnen Reiter muss in jeder Zeile genau der Subscreen-Bereich eingetragen werden, der mit dem einzublendenden Subscreen-Dynpro korrespondiert. Außerdem müssen die zum Umschalten benutzten Funktionscodes den Funktionstyp P für lokale GUI-Funktion zugewiesen bekommen.

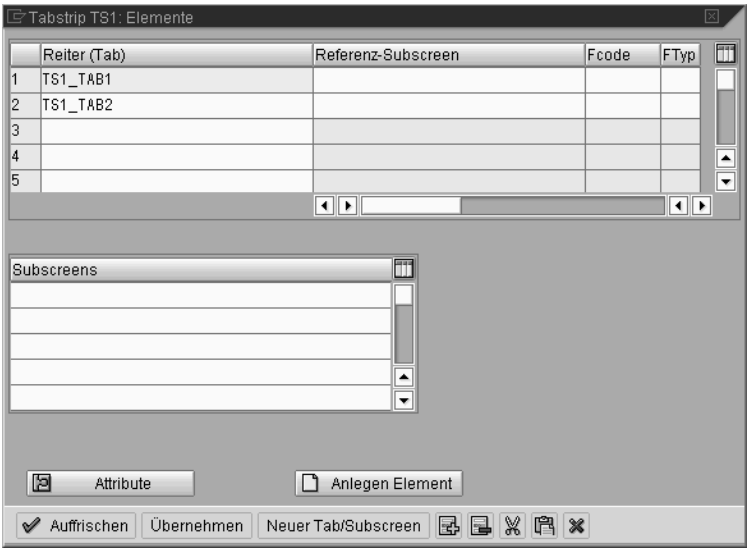

Abbildung 3.128 **© SAP AG**
Elemente des Tab-Strip-Controls pflegen

Die Namen der Subscreen-Bereiche sind frei wählbar. Sie werden später in der Ablauflogik der Anwendung benutzt. Nach dem Eintragen eines Namens betätigen Sie die ↵-Taste. Darauf fordert das System in einem weiteren Popup zur Bestätigung des Subscreen-Namens und zur Eingabe der Größe auf. In der Anwendung kann später nur die hier definierte Größe vom einzufügenden Subscreen belegt werden. Der Subscreen-Bereich und der dort später eingeblendete Subscreen darf natürlich nicht größer sein als der Tab-Strip-Bereich.

Im Anschluss an die Definition des Subscreen-Bereichs weisen Sie den Reitern die notwendigen Informationen zu. In die Spalte Referenz-Subscreen tragen Sie den Namen des eben definierten Subscreens ein. Die Spalte `fcode` nimmt einen frei wählbaren Funktionscode auf. Sie können das Popup nun durch die Drucktaste ÜBERNEHMEN verlassen. In der Pflegeoberfläche des Dynpros (Abbildung 3.129) hebt das System den Subscreen-Bereich hervor.

Die programmtechnische Behandlung eines Tab Strip ist relativ einfach und ähnelt der von Subscreens. Zunächst muss ein Tab Strip mit der Anweisung `CONTROLS` deklariert werden:

```
CONTROLS tabstrip TYPE TABSTRIP.
```

Die Betätigung eines der Reiter löst einen Funktionscode aus, der im User-Command-Modul ausgewertet wird, um die Nummer des einzufügenden Dynpros zu setzen. Dieses wird innerhalb der Ablauflogik mit der Anweisung `CALL SUBSCREEN` in den Subscreen-Bereich des Tab Strip eingeblendet.

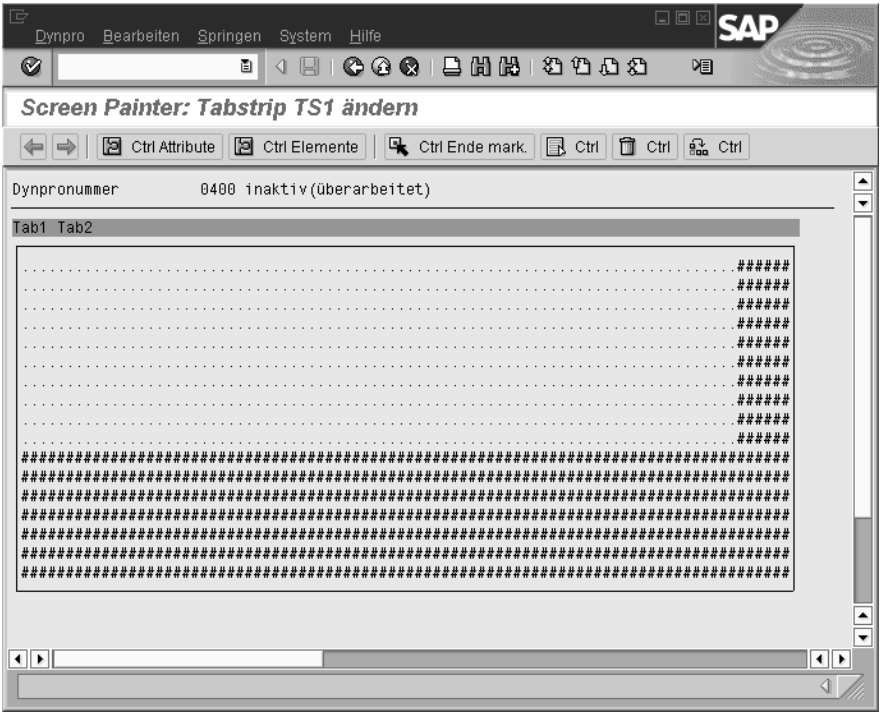

Abbildung 3.129 **© SAP AG**
Screen Painter mit Tab Strip und Subscreen-Bereich

Das mit der `CONTROLS`-Anweisung angelegte Objekt verfügt über drei Felder, die bei Bedarf alle drei dynamisch gesetzt werden können. Es handelt sich dabei um die Felder `prog`, `dynnr` und `activtab`. Im Feld `prog` wird der Name des Programms eingetragen, aus dem der einzublendende Subscreen entnommen werden soll. Dementsprechend nimmt das Feld `dynnr` die Nummer des einzufügenden Dynpros auf. Im Feld `activetab` schließlich ist der Name des Reiters einzutragen, der als aktiver Reiter erscheinen soll.

Icons

Ab Release 3.0 besteht die Möglichkeit, im Dynpro auch Icons zu verwenden. Bei Verwendung des Begriffs Icon ist zu beachten, dass die im deutschen Sprachraum oft verwendete und eigentlich sinnvolle Übersetzung Symbol hier nicht korrekt ist. Das SAP-System kennt sowohl „Icons“ als auch „Symbols“. Dabei handelt es sich um zwei verschiedene Elemente! An dieser Stelle soll nur auf die Icons eingegangen werden, da nur sie in Dynpros verwendet werden können. Im SAP-Sprachgebrauch wird der Begriff Icon mitunter mit *Ikone* eingedeutscht.

Icons können nur bestimmten Feldern ohne Eingabemöglichkeit zugeordnet werden. Dabei handelt es sich um Schlüsselfelder, Ausgabefelder und Drucktasten. Die Zuweisung zu Schlüsselfeldern erfolgt im Attribute-Popup. Dort steht ein entsprechendes Feld mit einer Eingabehilfe bereit. Die Zuweisung ist statisch, kann also zur Laufzeit der Anwendung nicht mehr geändert werden.

Bei Feldschablonen, bei denen das Attribut EINGABEFELD zurückgesetzt sein muss, ist das dynamische Zuweisen eines Symbols möglich. Dazu sind zwei Aktionen erforderlich. Im Dynpro sollte für das jeweilige Feld das Attribut MIT ICON angekreuzt werden. Dies ist nicht zwingend erforderlich, erleichtert aber die korrekte Berechnung der Ausgabelänge des Feldes durch den Screen Painter. Der zweite Schritt besteht im Zuweisen des Icons zum Feld. Es ist nicht möglich, den Namen des Icons direkt zuzuweisen. Vielmehr setzt der Funktionsbaustein `ICON_CREATE` den Namen des Icons in eine spezielle Zeichenkette um, die dem Datenelement zuzuweisen ist. Ein Funktionsbaustein ist, vereinfacht gesagt, ein spezielles Unterprogramm. Funktionsbausteine werden noch genauer besprochen. Dem erwähnten Baustein sind einige Parameter zu übergeben, mit denen das Aussehen des Anzeigefeldes beeinflusst werden kann. Dem erwähnten Funktionsbaustein können Sie einige optionale Parameter übergeben, um einige Darstellungsoptionen des Icons zu beeinflussen.

Attribute von elementaren Dynpro-Feldern

Positioniert man den Cursor auf ein Dynpro-Feld, ist mittels der Menüfunktion SPRINGEN | ATTRIBUTE ZUM FELD oder über die gleichnamige Drucktaste die Anzeige oder Pflege der Feldattribute möglich. Diese Funktion ruft ein Popup auf, in dem die Attribute aller Dynpro-Elemente – außer denen von Controls – gepflegt werden.

Das Aussehen des Popups hängt zum Teil von der Art des zu bearbeitenden Feldes ab. Während für Ein-/Ausgabefelder ein Popup gemäß Abbildung 3.130 bereitgestellt wird, erhalten Drucktasten ein anderes Popup (Abbildung 3.131).

Nicht für jeden der unten beschriebenen Feldtypen ist jedes Attribut pflegbar. Dazu finden Sie bei der Beschreibung der Feldtypen nähere Angaben.

Das Attribute-Popup ist in fünf Bereiche eingeteilt. Im oberen Bereich befinden sich Eingabe- oder Anzeigefelder für die allgemeinen Attribute. Der mittlere Teil des Dynpros besteht aus drei Spalten mit jeweils logisch zusammengehörenden Attributen. Am unteren Rand des Popups befinden sich zwei Felder, die zur Dictionary-Gruppe gezählt werden können, aber wegen ihrer Breite nicht in die entsprechende Spalte passen.

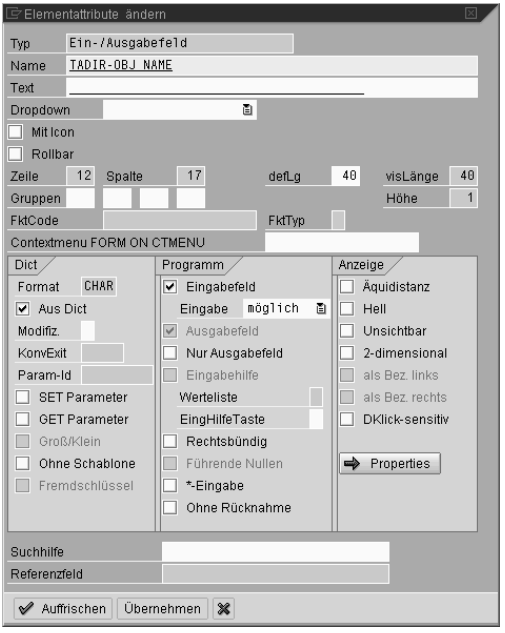

Abbildung 3.130 **© SAP AG**
Popup zur Pflege der Attribute eines Dynpro-Feldes

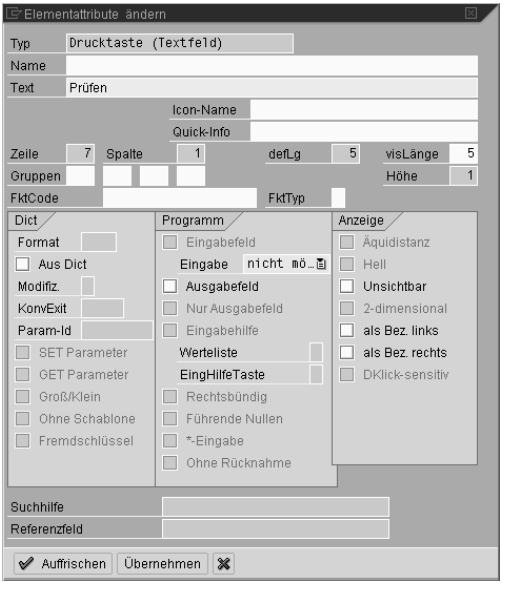

Abbildung 3.131 **© SAP AG**
2. Variante des Popups für Feldattribute

Typ

Der Feldtyp gibt Auskunft über die Art des Feldes und damit über seine Erscheinungsform auf dem fertigen Dynpro. Er kann nicht im Popup gesetzt oder verändert werden. Ein neu angelegtes Feld ist zunächst entweder ein Schlüsselfeld (ein Bezeichner) oder eine Feldschablone (Ein-/Ausgabefeld). Mit den Werkzeugen des Screen Painters kann der Feldtyp verändert werden, wobei sich auch das Aussehen des Eingabelementes ändert.

Name

Die meisten Dynpro-Felder müssen mit einem Daten- oder Tabellenfeld verbunden werden, um den Datenaustausch zu ermöglichen. Der Name dieses Feldes wird in diesem Attribut gespeichert. Er kann im Popup geändert werden.

Text

Der Inhalt dieses Feldes hängt vom Feldtyp ab. Für Dynpro-Felder mit Ein- oder Ausgabefunktion steht hier der Platzhalter, der im Fullscreen-Editor zu finden ist. Er besteht für Eingabefelder üblicherweise aus Unterstrichen „_“. Mitunter enthält der Platzhalter aber auch Musterzeichen.

Für Felder, die keine direkte Eingabe von Werten ermöglichen (z. B. Schlüsselfelder, Rahmen, Drucktasten), entspricht der Feldtext der später im Dynpro sichtbaren Bezeichnung.

Dropdown

Dieses Attribut besitzt nur für Ein-/Ausgabefelder eine Bedeutung. Es legt fest, ob in das Feld beliebige Werte manuell eingetragen werden können (Wert: `SPACE`) oder ob nur ein Wert aus einer vorgegebenen Liste ausgewählt werden kann (Wert: „L“ für Listbox). Im letztgenannten Fall erscheint später im Dynpro rechts neben dem Eingabefeld ein Symbol, mit dem die Vorschlagsliste eingeblendet werden kann. Die Vorschlagsliste kann durch das System selbst durch Auswertung von Wertetabellen oder aber zur Programmlaufzeit (Zeitpunkt PBO) mit Hilfe des Funktionsbausteins `VRM_SET_VALUES` erstellt werden.

Mit Icon

Ab Release 3.0 können in Dynpros Symbole (Icons) benutzt werden. Diese Icons können statisch als Attribut eines Dynpro-Feldes gesetzt (siehe Icon-Name) oder in Ausgabefeldern auch dynamisch angezeigt werden. Dazu ist ein Ausgabefeld zu erzeugen und das Feld MIT ICON zu markieren. Im PBO-Modul wird anschließend anhand des Icon-Namens durch einen Funktionsbaustein ein Identifikator für das Icon ermittelt. Dieser Identifikator wird dann dem Datenfeld, das mit dem Ausgabefeld des Dynpros zusammenarbeitet, zugewiesen. Im Dynpro erscheint dann nicht der Identifikator, sondern das Icon.

Icon-Name

Felder mit speziellem Typ (Drucktasten und Schlüsselfelder) können mit einem Icon ergänzt werden oder nur aus einem Icon bestehen. Für diese Felder ist das Eingabefeld ICON-NAME aktiv. Über die Werthilfetaste können die verfügbaren Icon-Namen ermittelt werden.

Rollbar

Bei Eingabefeldern, die nicht auf den Dictionary-Datentypen `TIMS`, `DATS`, `QUAN`, `CURR`, `DEC`, `INT1` und `INT2` beruhen, kann die sichtbare Länge des Eingabefelds kleiner als die Gesamtlänge gewählt werden. Dies ist aber nur möglich, wenn das Feld ROLLBAR angekreuzt wird. Während der Bearbeitung des Feldinhalts rollt der Inhalt bei Erreichen des rechten oder linken Feldrands weiter.

Zeile, Spalte

In diesen beiden Feldern stehen die Koordinaten des Dynpro-Feldes. Die Koordinaten können nur durch Verlagern des Feldes im Fullscreen-Editor geändert werden. Im Attribute-Popup sind diese Felder reine Anzeigefelder.

Definierte Länge, visuelle Länge

Die definierte Länge eines Dynpro-Felds gibt an, wie viele Zeichen dieses Feld maximal aufnehmen und bearbeiten kann. Für Eingabefelder muss die definierte Länge des Dynpro-Felds der Länge des korrespondierenden Datenfelds entsprechen, sonst wird der Inhalt des Datenfelds abgeschnitten. Dies kann in Ausnahmefällen erwünscht sein, z. B. wenn Zahlen mit einer maximalen Stellenzahl eingegeben werden sollen, die geringer ist als der maximal mögliche Wert. Beim Erzeugen von Dynpro-Feldern mit der Einfügehilfe des Fullscreen-Editors wird die definierte Länge vom zu Grunde liegenden Datenelement übernommen.

Die visuelle Länge ist die im Dynpro sichtbare Breite des Feldes. Sie entspricht normalerweise der definierten Länge. Ab Release 3.0 sind rollbare Felder möglich. Ist das entsprechende Attribut ROLLBAR gesetzt, wird das Feld für die sichtbare Länge eingabebereit. Dadurch kann die visuelle Länge geringer als die definierte Länge gewählt werden. Beim Editieren rollt der Feldinhalt beim Erreichen der rechten oder linken Begrenzung.

Höhe

Dieses Ausgabefeld stellt die Höhe des jeweiligen Dynpro-Elements dar.

Gruppen

In der internen Beschreibung eines Dynpro-Feldes werden vier Felder mit den Namen GROUP1 bis GROUP4 mitgeführt. Die Felder und deren Inhalte werden bei der dynamischen Modifikation eines Dynpros ausgewertet. Diese vier Felder können über die vier Eingabefelder des Attributes GRUPPEN mit beliebigen Werten belegt werden. Sie haben keine direkte Auswirkung auf die Darstellung

des Feldes im Dynpro, sondern stellen lediglich eine Klassifizierung dar. Zur dynamischen Bearbeitung von Dynpros folgen ausführliche Beispiele.

FktCode, FktTyp

Nicht nur in den Status der Oberfläche, sondern auch in Dynpros können Drucktasten definiert werden. Diese müssen natürlich auch einen Funktionscode besitzen, der wiederum einen Funktionstyp ausweist. Für Dynpro-Drucktasten können beide Werte im Attribute-Popup gesetzt werden. Die Werte können nicht mit der Funktionsliste der Oberfläche abgeglichen werden! Dies birgt einige Gefahren, da nun identische Funktionscodes mit unterschiedlichem Funktionstyp erzeugt werden könnten. Da der Funktionstyp beim Debuggen einer Anwendung nicht angezeigt wird, können sich scheinbar unerklärliche Abweichungen bezüglich der Auswertung der Funktionscodes ergeben.

Contextmenu

In diesem Feld können Sie einen Namen vorgeben, aus dem das System durch Voransetzen der Zeichenkette `ON_CT_MENU_` den Namen eines Unterprogramms bildet. Dieses Unterprogramm wird aufgerufen, wenn im entsprechenden Feld die rechte Maustaste gedrückt wird.

LoopTyp, LoopAnz

Zur erleichterten Bearbeitung von Massendaten können in einem Dynpro mehrere Datensätze einer Tabelle gleichzeitig bearbeitet werden. Dies erfolgt mit einem speziellen Eingabeelement, einer so genannten Loop. Diese Loop kann zwei verschiedene Typen besitzen, die Auskunft über die Anzahl der Eingabezeilen in der Loop geben. Beim Typ „Fix“ besitzt die Loop unabhängig von der Fenstergröße eine feste Zeilenzahl, die im Feld LOOPANZ ersichtlich ist. Die Anzahl der Zeilen kann nur im Fullscreen durch Bearbeiten der Loop verändert werden. Beim Typ „Variabel“ wird die Anzahl der Loop-Zeilen an die Größe des Bildschirmfensters angepasst. Im Feld LOOPANZ steht dann zwar auch die Anzahl der Loop-Zeilen aus dem Fullscreen-Editor, dies hat allerdings keine Auswirkungen auf die Darstellung im Dynpro.

Format

Dieses Anzeigefeld enthält das Datenformat des Datenfeldes, mit dem das Dynpro-Feld verbunden wurde. Bei Feldern mit Dictionary-Bezug wird der Datentyp automatisch gesetzt. Handelt es sich beim verbundenen Datenelement um ein programminternes Feld ohne Dictionary-Bezug, muss hier der Datentyp eingetragen werden. Dazu sollte die Eingabehilfe (Funktionstaste F4) genutzt werden, da in Dynpros nicht alle Datentypen zulässig sind.

Aus Dict

Mit diesem Ankreuzfeld legen Sie fest, ob alle Informationen für dieses Dynpro-Feld aus dem korrespondierenden Dictionary-Feld übernommen werden sollen. Dies betrifft z.B. bei Feldschablonen die Länge oder bei Schlüsselfeldern den Feldtext.

Modifiziert

Mit der Einfügehilfe können Sie verschiedene Schlüsseltexte aus dem Dictionary übernehmen, zusätzlich können Sie diese Texte nachträglich modifizieren. Die Art des Schlüsseltextes bzw. eine manuelle Modifikation dieser Texte wird durch den Inhalt dieses Attributes gekennzeichnet. Die Eingabehilfe für dieses Feld zeigt alle verfügbaren Werte.

KonvExit

Bei Ein- und Ausgabe von Feldern können Konvertierungsroutinen aktiv werden. Diese sind im System in Form von Funktionsbausteinen abgelegt. Diese Funktionsbausteine besitzen die Namen `CONVERSION_EXIT_xxxxx_INPUT` bzw. `CONVERSION_EXIT_xxxxx_OUTPUT`. Für die Identifizierung sind wegen des vorgegebenen Aufbaus lediglich fünf Zeichen erforderlich, die im Feld KONVEXIT eingetragen werden können.

Parameter-ID, SET-Parameter, GET-Parameter

Die Get/Set-Parameter sind transaktionsübergreifende Datenfelder. Sie werden in einem globalen Speicherbereich aufbewahrt, der programmübergreifend während der gesamten Sitzung eines Anwenders gültig ist. Diese Parameter können mit den Anweisungen `GET PARAMETER` bzw. `SET PARAMETER` gefüllt oder gelesen werden. Für Dynpro-Felder ist der Einsatz der ABAP-Anweisungen nicht nötig. Über die Flags SET-PARAMETER oder GET-PARAMETER wird der Datenaustausch mit dem Parameter freigegeben. Dadurch können beispielsweise wichtige Schlüsselfelder automatisch in Dynpros als Vorschlagswert übernommen werden.

Groß/Klein

Ist dieses Feld markiert, werden Groß- und Kleinbuchstaben unverändert übernommen, andernfalls erfolgt eine Umwandlung zu Großbuchstaben. Dies kann bei Schlüsselfeldern sinnvoll sein, um stets eine eindeutige Schreibweise zu gewährleisten.

Ohne Schabl.

Einige Zeichen, z.B. „?“ oder „!“, besitzen in Eingabefeldern eine spezielle Bedeutung. Mit dem Flag kann die Auswertung dieser Zeichen abgeschaltet werden, so dass sie wie andere Zeichen eingegeben werden können.

Fremdschl.

Für alle Felder, für die im Dictionary eine Fremdschlüsselprüfung definiert wurde, werden Eingaben unabhängig von eventuellen Feldprüfungen der Ablauflogik auch noch gegen die Prüftabellen geprüft. Eine solche Prüfung muss durch dieses Flag eingeschaltet werden. Ist keine automatische Fremdschlüsselprüfung erwünscht, kann das Flag deaktiviert werden.

Referenzfeld

Dieses Attribut ist nur bei Währungs- oder Einheitenfeldern belegt. Es enthält einen Feldnamen, in dem zur Laufzeit der Währungs- oder Einheitenschlüssel enthalten ist. Das Dynpro-Feld wird entsprechend der Eigenschaften der Währung oder der Maßeinheit aufbereitet. Das Referenzfeld wird bei Dictionary-Feldern direkt dem Tabellenfeld zugeordnet.

Eingabefeld

Nur wenn dieses Flag markiert ist, werden eingegebene Werte aus dem Dynpro-Feld in das Datenelement übertragen. Bei Schlüsseltexten ist dieses Flag ohnehin zurückgesetzt, bei Feldschablonen kann es deaktiviert werden. Das Feld ist dann inaktiv. Dieser Zustand wird durch eine andere Farbe des Feldhintergrundes signalisiert. Der Rahmen des Feldes bleibt allerdings erhalten.

Ausgabefeld

Wenn dieses Flag markiert ist, findet ein Datentransport vom programminternen Datenfeld zum Dynpro-Feld statt. Wenn das Flag nicht eingeschaltet ist, wird der Inhalt des korrespondierenden Datenfeldes nicht angezeigt. Die Übergabe vom Dynpro-Feld zum Datenfeld nach Bearbeitung des Dynpros ist von diesem Flag nicht betroffen.

Nur Ausgabefeld

Dieses Flag ist nur für Feldschablonen sinnvoll. Es bewirkt die Ausgabe des Feldinhaltes in der Form eines Schlüsselfeldes, also ohne den umgebenden Rahmen, den Feldschablonen normalerweise besitzen. Solche Felder können eingesetzt werden, um Texte in Form eines Schlüsselfeldes dynamisch anzuzeigen.

Eingabe

Dieses Feld bietet in einer Dropdown-Liste vier Einstellungsmöglichkeiten bezüglich der Eingabearten in diese Feld an. Mit der Einstellung „nicht möglich" werden Eingaben in das Feld verboten. Diese Einstellung ist Voraussetzung für die mögliche Markierung des NUR AUSGABE-Feldes. Freigeschaltet wird die Eingabe durch die Einstellung „möglich". Die beiden anderen Auswahlmöglichkeiten empfehlen oder erzwingen eine Eingabe in das jeweilige Eingabefeld. Beide Werte führen dazu, dass zur Laufzeit im jeweiligen Eingabefeld ein Symbol eingeblendet wird, das auf eine notwendige Eingabe hinweist. Bei Feldern mit dem

Attribut „empfohlen“ wird durch das System allerdings nicht geprüft, ob wirklich eine Eingabe erfolgt (Soll-Eingabe). Anders bei der Einstellung „obligatorisch“. Hier prüft das System, ob in das Eingabefeld ein Wert eingetragen wurde. Falls keine Eingabe erfolgt, kann das Dynpro nur durch die Exit-Funktion verlassen werden. Eine Fortsetzung der Anwendung ist somit nicht möglich.

Eingabehilfe

Dieses Attribut wird vom Screen Painter automatisch für Ein-/Ausgabefelder gesetzt und kann von Ihnen nicht direkt bearbeitet werden. Wenn dieses Feld markiert ist, dann steht für das Eingabefeld eine Eingabehilfe zur Verfügung. Diese kann durch unterschiedliche Mechanismen (Fremdschlüsselbeziehung, Werteliste an der Domäne, Suchhilfe, selbst programmierte Werthilfe) bereitgestellt werden.

Werteliste

Dieses Attribut können Sie nur für Eingabefelder setzen, bei denen auch das Attribut DROPDOWN BOX gesetzt ist. Der mögliche Wert (Leerzeichen oder „A“) legt fest, wie die Wertemenge für die Vorschlagsliste beschafft wird. Falls Sie dieses Feld mit einem Leerzeichen belegen, wird die Vorschlagsliste durch die existierenden Standard-Mechanismen (Domänenfestwerte, Wertetabellen, Suchhilfen) erzeugt. Falls jedoch ein „A“ eingetragen wird, müssen Sie die Werteliste zum Zeitpunkt PBO durch Aufruf des Funktionsbausteins `VRM_SET_VALUES` selbst mit Werten füllen. Diesen Baustein müssen Sie auch dann benutzen, wenn Sie eine eigene Werthilfe für den Zeitpunkt POV programmieren. Die POV-Module zählen aber zu den Standard-Methoden der Werthilfe und werden somit nur dann aufgerufen, wenn das Attribut WERTELISTE mit dem Leerzeichen belegt ist.

EingHilfeTaste

Mit diesem Attribut wird festgelegt, ob eine existierende Werthilfe für dieses Feld aktiv sein soll. Außerdem hat der Wert dieses Attributes Einfluss darauf, wann das Werthilfesymbol neben dem Feld angezeigt wird. Tabelle 3.38 zeigt die möglichen Einträge.

Wert	Zustand Werthilfe	Symbol für Werthilfe sichtbar
SPACE	Werthilfe aktiv	wenn Cursor im Feld steht
0	Werthilfe abgeschaltet	
1	Werthilfe aktiv	wenn Cursor im Feld steht
2	Werthilfe aktiv	immer

Tabelle 3.38
Wertebereich des Attributs Werthilfetaste

Rechtsbündig

Der Inhalt des Feldes wird rechtsbündig dargestellt. Dies gilt außer bei nummerischen Feldern nur für Ausgabefelder.

Führende Nullen

Rechtsbündig ausgerichtete nummerische Werte in Feldschablonen werden mit führenden Nullen versehen.

*-Eingabe

Die Verwendung dieses Attributes ist überholt. In Eingabefelder mit diesem Attribut kann an der ersten Stelle ein Stern „*“ eingetragen werden. In einem solchen Fall wird ein PAI-Modul für dieses Feld aufgerufen, das durch einen speziellen Zusatz gekennzeichnet ist.

Ohne Rücknahme

Wenn ein Anwender an der ersten Stelle eines Eingabefeldes ein „!“-Zeichen einträgt, wird das Feld nach Beendigung der Dateneingabe im Dynpro initialisiert, es wird also kein Wert übernommen. Dieser Mechanismus kann durch Setzen des Attributes OHNE RÜCKNAHME abgeschaltet werden. Dies ermöglicht es, auch an der ersten Stelle eines Feldes Ausrufungszeichen einzugeben.

Suchhilfe

Suchhilfen sind spezielle Eingabehilfen, mit denen Schlüsselwerte über Nicht-Schlüsselwerte gesucht werden können. Bis zur Version 3.x wurden dazu Matchcodes benutzt. Suchhilfen werden in eigenständigen Pflegeprogrammen generiert. Sie erhalten einen eindeutigen Namen. Wenn der Name einer Suchhilfe in diesem Attribut eingetragen wird, ist mittels der F4-Taste der Aufruf des Suchhilfe für das jeweilige Dynpro-Feld möglich. Ab Release 3.0 kann der Name des Matchcodes bzw. der Suchhilfe auch dynamisch zur Laufzeit übergeben werden. In diesem Fall steht im Attribut SUCHHILFE der Name des Feldes, dem ein Doppelpunkt vorangehen muss. Zur Laufzeit wird im PBO-Teil des Dynpros das Feld mit dem Namen des jeweils zu verwendenden Matchcodes belegt.

Äquidistanz

Mit diesem Feld kann veranlasst werden, dass Texte unter bestimmten Voraussetzungen in einer Äquidistanzschrift ausgegeben werden, also einer Schriftart, in der alle Zeichen gleich breit sind.

Hell

Das Feld wird bei Aktivierung durch eine andere Farbe oder Helligkeit hervorgehoben.

Unsichtbar

Dieses Attribut wird für Felder mit Passwortfunktion benutzt. Ein- und Ausgaben in Felder mit diesem Attribut sind nicht sichtbar, da im Feld nur Sterne angezeigt werden. Ein Beispiel ist das Feld für die Passworteingabe beim Anmelden an das R/3-System.

2-dimens.

Die Feldschablonen und Drucktasten im Dynpro werden so dargestellt, dass sich ein dreidimensionaler Eindruck ergibt. Diese Funktion kann mit diesem Flag abgeschaltet werden.

als Bez. links/als Bez. rechts

Diese Flags ist nur für Textfelder sinnvoll verwendbar. Wenn eines dieser Flags gesetzt ist, wird durch eine dünne Linie die Zusammengehörigkeit dieses Textfelds mit dem rechts (für als Bez. links) bzw. links (für: als Bez. rechts) von ihm stehenden Dynpro-Element symbolisiert.

Gegebenenfalls kann das Flag auch für Feldschablonen benutzt werden, die durch Rücknahme der Eingabefunktionalität und Abschalten der dreidimensionalen Darstellung auf dem Dynpro wie Textfelder erscheinen.

Dklick-sensitiv

Dieses Flag kann nur für Text- und Ein-/Ausgabefelder gesetzt werden, für die der Doppelklick eine Funktion auslöst. Das gesetzte Flag bewirkt eine Markierung des jeweiligen Feldes, damit auf die Sonderfunktion hingewiesen wird.

Properties-Dialog

Der Properties-Dialog ermöglicht die Pflege zusätzlicher Informationen zu einem Dynpro-Feld. Diese Informationen können ggf. durch den Screen Reader in akustische Informationen umgewandelt werden, um sehbehinderten Benutzern die Bedienung der Anwendung zu erleichtern. Der Properties-Dialog (siehe Abbildung 3.132) verfügt über zwei Elemente, die jeweils in einem weiteren Popup bearbeitet werden.

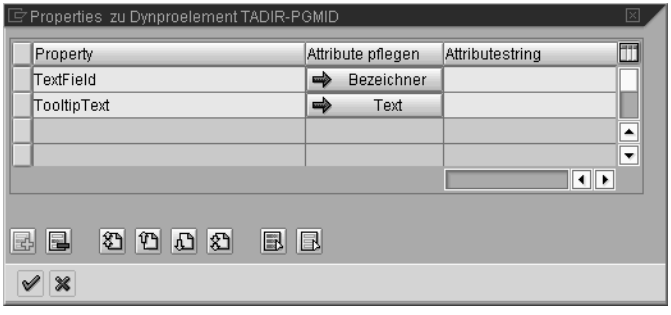

Abbildung 3.132 **© SAP AG**
Properties eines Dynpro-Feldes pflegen

Beim ersten Element handelt es sich um den Bezeichner des Feldes (siehe Abbildung 3.133). Er wird als einfacher Textstring abgebildet.

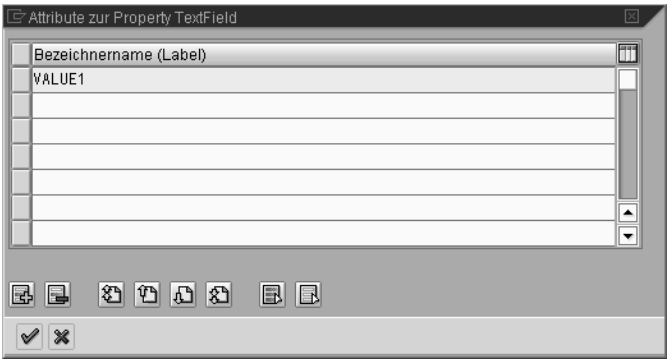

Abbildung 3.133 **© SAP AG**
Bezeichner für ein Textlabel pflegen

Das zweite Element ist der Inhalt eines Tooltip-Feldes. Er wird entweder durch ein vordefiniertes Textelement oder den Inhalt einer Variablen vorgegeben (siehe Abbildung 3.134).

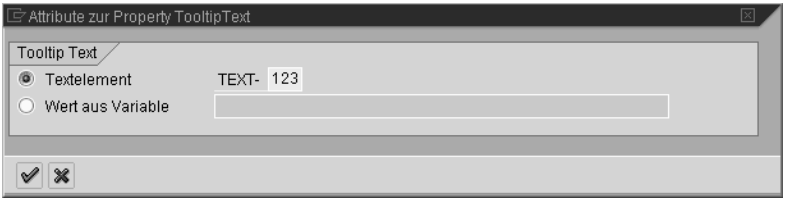

Abbildung 3.134 **© SAP AG**
Text für einen Tooltip festlegen

Attribute von Table Controls

Table Views verfügen über andere Attribute als die bisher beschriebenen Dynpro-Elemente. Für die Pflege der Attribute steht demzufolge auch ein anderes Popup (Abbildung 3.135) zur Verfügung. Die Attribute der Table Views sollen daher getrennt beschrieben werden.

Name

Das Attribut FELDNAME nimmt den Namen des Table Control auf. Der Table Control muss im Deklarationsteil des Programms mit der Anweisung `CONTROLS` deklariert worden sein.

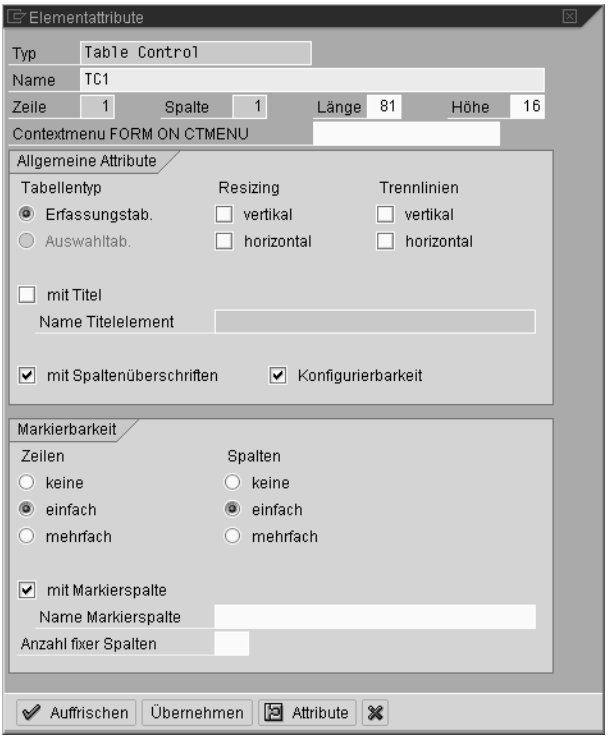

Abbildung 3.135 **© SAP AG**
Attribute-Popup für Table Controls

Contextmenu

In diesem Feld können Sie einen Namen vorgeben, aus dem das System durch Voransetzen der Zeichenkette `ON_CT_MENU_` den Namen eines Unterprogramms bildet. Dieses Unterprogramm wird aufgerufen, wenn im entsprechenden Feld die rechte Maustaste gedrückt wird.

Erfassungstabelle/Auswahltabelle

Diese beiden Auswahlfelder legen fest, ob die im Table Control dargestellten Daten bearbeitet werden können oder nur eine Anzeige möglich ist. Auswahltabellen ermöglichen es dem Anwender, Datensätze zu markieren. Was mit diesen Datensätzen im weiteren Verlauf des Programms geschieht, hängt dann von der Gestaltung des Programms ab.

Resizing vertikal, horizontal

Diese beiden Attribute steuern das Verhalten des Table Control bei Größenänderungen des SAPGUI-Fensters. Sind sie gesetzt, bleibt der Abstand des Table Control vom unteren bzw. rechten Fensterrand so lange wie möglich konstant.

Das bedeutet, dass der Table Control gegebenenfalls kleiner wird. Durch diese Maßnahme bleiben Felder unterhalb oder rechts vom Table Control sichtbar.

Sind die Attribute nicht gesetzt, bleibt die Größe des Table Control konstant. Falls das Fenster des SAPGUI kleiner wird als zur Darstellung des Dynpros erforderlich, erhält es automatisch Rollbalken.

Trennlinien vertikal, horizontal

Mit diesen beiden Attributen können Trennlinien zwischen den einzelnen Tabellenzeilen oder -spalten eingeblendet werden.

Mit Titel, Feldname Titel

Der Table Control kann eine Überschrift erhalten. Diese Überschrift wird in einem Datenfeld abgelegt, dessen Name im Attribut FELDNAME TITEL einzutragen ist. Das System erfragt bei Pflege dieses Attributs, ob ein Schlüsselfeld (fester Text) oder eine Feldschablone (dynamischer Text) benutzt werden soll.

Mit Spaltenüberschriften

Wenn dieses Attribut gesetzt ist, wird über jede Spalte des Table Control eine Spaltenüberschrift eingeblendet. Wie bei der Tabellenüberschrift handelt es sich auch bei der Spaltenüberschrift um ein Schlüsselfeld oder eine Feldschablone. Die entsprechenden Feldnamen werden im Popup zur Pflege der Table-Control-Felder eingetragen.

Konfigurierbarkeit

Dieses Attribut kann nur gesetzt werden, wenn auch das Attribut MIT SPALTENÜBERSCHRIFTEN aktiv ist. Der Anwender hat bei freigegebener Konfigurierbarkeit die Möglichkeit, selbst vorgenommene Änderungen im Layout (Spaltenanordnung und -breite) des Table Control abzuspeichern. Beim nächsten Aufruf des Dynpros erscheint der Table Control in der modifizierten Form.

Markierbarkeit

Mit den zu diesem Attribut gehörenden Auswahlfeldern können Sie festlegen, ob und wenn ja, wie viele Zeilen oder Spalten markiert werden können.

Mit Markierspalte, Name Markierspalte

Falls dieses Attribut gesetzt ist, wird vor jeder Tabellenzeile eine Drucktaste eingeblendet, mit der diese Zeile markiert werden kann. Bei aktiviertem Attribut muss ein Feldname zugewiesen werden. Markierte Zeilen werden durch ein „X“ im entsprechenden Datenfeld gekennzeichnet. Dieses Feld ist nur innerhalb der `LOOP`-Schleife zum Zeitpunkt PAI gültig.

Anzahl fixer Spalten

Im Table Control können Spalten am linken Rand der Tabelle fixiert werden. Diese Spalten bleiben an ihrer Position. Der horizontale Rollbalken beginnt erst

nach dieser Spalte. Die Anzahl der fixierten Spalten wird im Attribut ANZAHL FIXER SPALTEN eingetragen.

Die Kommandos der Ablauflogik

In der Ablauflogik eines Dynpros erfolgt der Aufruf der Programmmodule des Modul-Pools. Mittels der dort enthaltenen Anweisungen wird die Funktionalität des Dynpros festgelegt. Die Ablauflogik besteht aus mehreren Abschnitten, die zu bestimmten Zeitpunkten abgearbeitet werden. Diese Zeitpunkte werden durch vier verschiedene Zeitpunktanweisungen definiert:

- `PROCESS BEFORE OUTPUT.`
- `PROCESS AFTER INPUT.`
- `PROCESS ON HELP-REQUEST.`
- `PROCESS ON VALUE-REQUEST.`

Nicht in jeder Ablauflogik sind alle vier Zeitpunkte anzutreffen. Die letzten beiden sind relativ selten, da durch das System selbst bereits leistungsfähige Mechanismen zur Eingabehilfe und Hilfestellung gegeben werden. Die ersten beiden Anweisungen hingegen sind in jedem Dynpro zu finden. Sie werden daher beim Erzeugen eines neuen Dynpros automatisch in dessen Ablauflogik geschrieben. Es ist allerdings durchaus möglich, dass zu einem der genannten Zeitpunkte keine Aktionen ausgeführt werden, obwohl die Zeitpunktanweisung vorhanden ist.

Die in den verschiedenen Abschnitten aufgerufenen Module führen spezielle, jeweils nur dort sinnvolle Aktionen aus. Einige der Aktionen werden im folgenden erwähnt. Innerhalb der Ablauflogik werden spezielle Anweisungen benutzt, die den ABAP-Kommandos zwar ähneln, aber nicht mit ihnen identisch sind. Einige dieser Kommandos haben im Prinzip nur die Aufgabe, Module aus dem Modul-Pool aufzurufen. Es existieren daher nur wenige unterschiedliche Kommandos für die Ablauflogik. Ebenso wie eine ABAP-Anweisung kann die Wirkungsweise der Kommandos der Ablauflogik durch einige Parameter modifiziert werden. Auch wenn einige der nachfolgend beschriebenen Kommandos an ABAP-Kommandos erinnern – außer dem Namen bestehen kaum Gemeinsamkeiten. Mit dem Begriff Kommando sind in diesem Abschnitt die Kommandos der Ablauflogik, mitunter auch als Dynpro-Kommandos bezeichnet, gemeint.

Die Anweisung FIELD

Im PAI-Teil eines Dynpros können mit der Anweisung `FIELD` feldbezogene Aktivitäten ausgelöst werden. Die in Verbindung mit `FIELD` notierten Kommandos beziehen sich dann nur auf das nach `FIELD` notierte Feld. Die Grundform des Kommandos lautet

```
FIELD field.
```

In dieser Form bewirkt das Kommando lediglich das Ansprechen des Feldes. Sie ist nur in einer CHAIN-ENDCHAIN-Kette sinnvoll. Falls keine Zusätze folgen, können mittels der Doppelpunktvariante auch mehrere Felder angesprochen werden:

```
FIELD: field_1, field_2, ... field_n.
```

Bedeutsamer ist die Verbindung der Anweisung FIELD mit der Anweisung MODULE in der Form

```
FIELD field MODULE modul.
```

Das Modul wird aufgerufen. Falls innerhalb des Moduls eine Nachricht (Fehlermeldung) erzeugt wird, ist nur das in der FIELD-Anweisung angegebene Feld wieder eingabebereit. Weitere Zusätze, die zur MODULE-Anweisung gehören und dort beschrieben werden, erlauben es, die Abarbeitung des Moduls von bestimmten Vorbedingungen abhängig zu machen.

Eine zweite Variante der FIELD-Anweisung ermöglicht den direkten Test eines Feldes gegen eine Tabelle, ohne dafür ein eigenes Modul schreiben zu müssen. Sie lautet

```
FIELD field SELECT ...
```

Die SELECT-Anweisung ist auch als eigenständige Anweisung verfügbar. Sie wird daher separat beschrieben. In der Verbindung mit FIELD wird bei einer in der SELECT-Anweisung ausgelösten Fehlermeldung nur das angegebene Feld eingabebereit.

Die letzte Variante der FIELD-Anweisung kann nur für Felder des Typs CHAR und NUMC benutzt werden. Sie prüft das Feld gegen eine Liste von direkt im Quelltext notierten Werten:

```
FIELD field VALUES (value_list).
```

Die in runde Klammern einzuschließende Werteliste besteht aus einem oder mehreren Einzelwerten. Diese sind in Hochkommas einzuschließen und durch Kommas voneinander zu trennen. Mit dem Operator NOT kann ein nicht zulässiger Wert gekennzeichnet werden, mit BETWEEN AND bzw. NOT BETWEEN AND werden zulässige bzw. nicht zulässige Bereiche gekennzeichnet.

In einer FIELD-Anweisung können mehrere der beschriebenen Zusätze eingesetzt werden. Sie sind dann in der Form

```
FIELD field : VALUES ... ,
              MODULE ... ,
               ...        .
```

zu notieren.

Die Anweisung MODULE

Der Aufruf von Modulen aus dem Modul-Pool erfolgt mit der Anweisung

```
MODULE modul.
```

Diese Anweisung kann bei allen vier Zeitpunkten benutzt werden. Es existieren für sie einige Zusätze, mit denen die Arbeitsweise beeinflusst werden kann. Der bereits in einem Beispiel demonstrierte Zusatz

```
... AT EXIT-COMMAND.
```

bewirkt, dass das Modul nur aufgerufen wird, wenn ein Funktionscode mit dem Typ E ausgelöst wurde. Dieser Zusatz ist daher nur im PAI-Teil der Ablauflogik sinnvoll. Die typischen Aktivitäten, die durch einen Exit-Funktionscode ausgelöst werden, bestehen im Verlassen des Dynpros. Aus diesem Grund ist je Dynpro nur eine Anweisung mit diesem Zusatz sinnvoll. Sie sollte zudem die erste Anweisung des Zeitpunktes PAI sein.

Alle anderen Zusätze sind nur dann einsetzbar, wenn der Aufruf des Moduls in einer Feldzuordnung in Zusammenhang mit dem Kommando `FIELD` erfolgt. Die möglichen Zusätze und ihre Wirkung gehen aus Tabelle 3.39 hervor.

Zusatz	Wirkung
`ON INPUT`	Abarbeitung nur dann, wenn das zugehörige Feld einen Wert ungleich dem Initialwert hat.
`ON CHAIN-INPUT`	Abarbeitung nur dann, wenn mindestens ein Feld der Kette einen Wert ungleich dem Initialwert hat. (Nur in `CHAIN`-Klammern sinnvoll.)
`ON REQUEST`	Abarbeitung nur dann, wenn im zugehörigen Feld eine Eingabe erfolgte.
`ON CHAIN-REQUEST`	Abarbeitung nur dann, wenn in mindestens einem Feld der Kette eine Eingabe erfolgte. (Nur in `CHAIN`-Klammern sinnvoll.)
`ON *-INPUT`	Abarbeitung nur dann, wenn im zugehörigen Feld ein Stern "*" eingegeben wurde und das Feld die Eigenschaft Sterneingabe hat.
`AT CURSOR-SELECTION`	Abarbeitung nur dann, wenn mittels Doppelklick oder der Funktionstaste F2 eine Auswahl erfolgte.

Tabelle 3.39
Zusätze zum Dynpro-Kommando MODULE

Die Anweisungen CHAIN...ENDCHAIN

Diese beiden Kommandos bilden einen Rahmen. Sie schließen üblicherweise eine `FIELD`-Anweisung für mehrere Felder und einen oder mehrere zugehörige `MODULE`-Aufrufe ein. Diese Kommandos werden durch die `CHAIN`-Anweisung zu einem Verarbeitungsblock oder einer Verarbeitungskette verbunden. Wird innerhalb der `CHAIN`-Kette eine Fehlernachricht ausgelöst, werden alle in der Kette mit `FIELD` angesprochenen Felder wieder eingabebereit. Dieses Verhalten

ist z.B. sinnvoll, wenn ein Anwender in einem Dynpro mehrere Schlüsselwerte oder andere voneinander abhängige Werte eintragen muss. Ein Fehler kann dann nicht mehr eindeutig einem bestimmten Feld zugeordnet werden. Ein (theoretisches) Beispiel für eine CHAIN-Kette könnte so aussehen:

```
CHAIN.
  FIELD: CREDCARD-TYPE,
         CREDCARD-NUMBER.
  MODULE CHECK_CREDIT_CARD ON CHAIN-REQUEST.
ENDCHAIN.
```

Das Modul CHECK_CREDIT_CARD soll prüfen, ob eine Zahlung mit der angegebenen Kreditkarte akzeptiert werden kann. Die zulässigen Nummernbereiche einer Kreditkarte sind vom jeweiligen Kreditkarteninstitut abhängig. Ein Fehler kann sowohl entstehen, wenn aus Versehen eine falsche Kreditkartenart oder eine falsche Nummer eingegeben wurde. Aus diesem Grund müssen im Fehlerfall beide Felder eingabebereit werden. Dies gelingt nur mit Hilfe der CHAIN-Kette. Nach FIELD können zwar mehrere Felder notiert werden, wenn jedoch direkt in der FIELD-Anweisung ein Modul aufgerufen wird, setzt das System im Fehlerfall nur das letzte Feld der Feldliste auf Eingabebereitschaft.

Die Anweisung SELECT

Innerhalb der Ablauflogik ist eine spezielle Form der SELECT-Anweisung verfügbar, mit der im PBO-Teil ein Datensatz aus einer Tabelle gelesen und im Dynpro bereitgestellt werden kann. Im PAI-Teil wird die Anweisung benutzt, um Werte aus dem Dynpro gegen eine Tabelle zu prüfen. Dieses Dynpro-Kommando ist in Neuentwicklungen kaum noch gebräuchlich. Es kann ohne weiteres durch ein Modul ersetzt werden, in dem dann komfortablere Varianten der SELECT-Anweisung zur Verfügung stehen.

Die Grundform der SELECT-Anweisung der Ablauflogik lautet:

```
SELECT * FROM table
  WHERE tablefield = inputfield ...
```

In der WHERE-Klausel können Sie mehrere Selektionsbedingungen angeben, allerdings ist nur ein Test auf Gleichheit möglich. Der einzige zugelassene Operator ist damit das Gleichheitszeichen. In der WHERE-Klausel können beliebige Felder des aktuellen Programms verwendet werden, es existiert keine Beschränkung auf die Felder des aktuellen Dynpros. Es ist daher möglich, die Selektion in Abhängigkeit von Werten, die in anderen Dynpros erfasst wurden, auszuführen. Beispielsweise könnte man in einem Dynpro Schlüsselfelder abfragen und im PBO-Teil des nachfolgenden Dynpros anhand dieser Schlüsselfelder einen Datensatz lesen. Dazu muss der Inhalt des gelesenen Datensatzes in die Kopfzeile der Tabelle gestellt werden. Dies erfolgt mit dem Zusatz

```
... INTO table
```

Er kann nur in die Kopfzeile einer Dictionary-Tabelle übertragen werden, die Dynpro-Felder müssen also direkt mit den Tabellenfeldern verbunden werden. Ohne den Zusatz INTO würde die SELECT-Anweisung den Datensatz zwar finden, ihn aber nicht zur weiteren Bearbeitung bereitstellen. Dieses Verhalten ist beispielsweise im PAI-Teil im Zusammenhang mit Prüfungen sinnvoll.

Das Ergebnis der SELECT-Anweisung kann Auswirkungen auf den weiteren Programmablauf haben. Die einzige Möglichkeit, innerhalb der Ablauflogik den Programmablauf zu beeinflussen, besteht im Aussenden von Nachrichten. Dazu existieren zwei Zusätze, die vom Ergebnis des Suchvorgangs abhängig gemacht werden können. Der Zusatz

```
WHENEVER [NOT] FOUND
  SEND [ERRORMESSAGE | WARNING] number WITH messageparameters.
```

sendet eine Fehlernachricht aus, wenn die Datenselektion erfolgreich war. Anstelle von FOUND kann auch NOT FOUND benutzt werden, um die Fehlermeldung dann auszulösen, wenn kein Satz mit entsprechendem Schlüssel existiert. Eine harte Fehlermeldung (Abarbeitung des Dynpros von vorn) lösen Sie mit dem Zusatz ERRORMESAGE aus, mit WARNING erzeugen Sie eine einfache Warnung, die keinen Neubeginn der Dynproabarbeitung auslöst. Nach der Bestätigung der Warnung mit der Datenfreigabetaste (↵-Taste oder entsprechendes Symbol) wird die Abarbeitung des Dynpros fortgesetzt.

Die Aussendung einer individuellen Fehlernachricht ist nicht zwingend erforderlich. Das System generiert eine Standardfehlermeldung, wenn der WHENEVER-Zusatz nicht existiert und die Datenselektion erfolglos blieb. Dieses Verhalten ist im PBO-Teil zumeist unerwünscht. SELECT-Anweisungen im PBO-Teil senden daher meist eine Warnung aus, wenn der Datensatz nicht gefunden werden konnte.

Die SELECT-Anweisung kann, wie beim Kommando FIELD bereits erwähnt, als Zusatz zu dieser Anweisung verwendet werden. In diesem Fall wird das mit FIELD verbundene Feld wieder eingabebereit, wenn die SELECT-Anweisung eine Fehlermeldung erzeugt. Falls mehrere Felder eingabebereit werden sollen, ist SELECT zusammen mit einer FIELD-Anweisung für mehrere Felder in einer CHAIN-Kette zu benutzen. Nachfolgend drei typische Beispiele für die Verwendung der SELECT-Anweisung.

Suche eines Datensatzes im Abschnitt PBO:

```
PROCESS BEFORE OUTPUT.
...
SELECT * FROM KNA1
  WHERE KUNNR = KNA1-KUNNR
  WHENEVER NOT FOUND
    SEND WARNING 123 WITH KNA1-KUNNR.
```

Prüfen eines einzelnen Feldes im PAI-Abschnitt mit Standardnachricht:

```
FIELD KNA1-KUNNR SELECT * FROM KNA1
  WHERE KUNNR = KNA1-KUNNR.
```

SELECT in einer CHAIN-Kette:

```
CHAIN.
  FIELD: RECHNUNG-ZAHLART
         RECHNUNG-WAEHRUNG.
  SELECT * FROM ZULZART
    WHERE ZAHLART = RECHNUNG-ZAHLART
      AND WAEHRUNG = RECHNUNG-WAEHRUNG
    WHENEVER NOT FOUND
      SEND ERRORMESSAGE 345.
ENDCHAIN.
```

Die Anweisungen LOOP und ENDLOOP

Im Abschnitt über visuelle Elemente der Dynpros wurden Loops und Table Controls erwähnt. Sie ermöglichen die Bearbeitung mehrerer Datensätze in einem Dynpro. Da die einzelnen Blöcke eines Loops Kopien eines Musterblocks darstellen und daher identische Feldnamen enthalten, ist es nicht möglich, die Felder dieser Blöcke direkt anzusprechen. Für die Loop-Bereiche eines Dynpros ist daher eine spezielle Anweisung zur Bearbeitung der einzelnen Blöcke erforderlich.

Die beiden Anweisungen `LOOP` und `ENDLOOP` in der Ablauflogik bilden eine Schleife, in der alle Blöcke des Loop-Bereiches des Dynpros einmal durchlaufen werden. Zu diesem Zeitpunkt können die Felder des aktuellen Blocks über ihre Feldnamen angesprochen werden. Die Bearbeitung der Feldinhalte muss natürlich in einem Modul erfolgen, das in der `LOOP`-Schleife aufgerufen wird. Falls in einem Dynpro mehrere Loop-Bereiche enthalten sind, müssen dementsprechend auch mehrere `LOOP`-Anweisungen benutzt werden. Die Zuordnung von Loop-Bereich und Anweisung in der Ablauflogik erfolgt in der Reihenfolge der Deklaration der Bereiche im Dynpro.

Eine `LOOP`-Schleife ist sowohl zum Zeitpunkt PAI erforderlich, um die Dynpro-Felder mit Daten zu belegen, als auch zum Zeitpunkt PBO, um Daten zu prüfen und gegebenenfalls in die Datenbanktabelle zurückzuschreiben. Das Fehlen einer `LOOP-ENDLOOP`-Schleife verursacht beim Generieren eines Dynpros mit Step-Loop-Bereich eine Fehlermeldung. Eine einfache Ablauflogik für ein Dynpro mit einem Loop-Bereich könnte etwa folgendermaßen aussehen:

```
PROCESS BEFORE OUTPUT.
...
  LOOP.
    MODULE READ_LINE.
  ENDLOOP.
```

```
*
PROCESS AFTER INPUT.
...
  LOOP.
    MODULE PROCESS_LINE.
  ENDLOOP.
```

In diesem Fall wird die LOOP-Schleife für jede auf dem Dynpro sichtbare Zeile des Step Loop genau einmal durchlaufen. Innerhalb der Schleife steht das Feld SY-STEPL zur Verfügung, dessen Wert bei jedem Schleifendurchlauf um 1 erhöht wird. Nach Verlassen der Schleife ist der Wert für SY-STEPL undefiniert. Wenn Sie die Zahl der Step-Loop-Zeilen für weitere Auswertungen benötigen, muss der Wert von SY-STEPL innerhalb der LOOP-Schleife in ein globales Datenfeld der Anwendung gerettet werden. In dieser Form spricht die LOOP-Anweisung nacheinander alle Zeilen des Step Loop im Dynpro an. Die in diese Felder einzutragenden Daten müssen Sie gegebenenfalls in einem eigenen Modul beschaffen. Falls der Datenbestand nicht komplett auf dem Bildschirm dargestellt werden kann, müssen Sie eventuell auch das Blättern im Datenbestand selbst programmieren.

Im PAI-Teil der Ablauflogik ist ebenfalls eine LOOP-Schleife erforderlich, auch wenn Sie nicht die Absicht haben, geänderte Werte aus dem Dynpro zu lesen. Es ist somit auf jeden Fall ein LOOP-ENDLOOP-Paar erforderlich. Wenn Sie jedoch Daten aus dem Dynpro lesen und die geänderten Daten im Programm weiterverwenden möchten, so müssen Sie dies wiederum in einem speziellen Modul tun, in dem Sie über den Schleifenindex SY-STEPL die korrekte Tabellenzeile ermitteln.

Eine zweite Variante der LOOP-Anweisung stellt automatisch eine Verbindung zwischen einer internen Tabelle und dem Step Loop her. Die Version des Kommandos für den PBO-Zeitpunkt lautet:

```
LOOP AT itab CURSOR cursor.
ENDLOOP.
```

Mit dieser Anweisung wird stets die komplette interne Tabelle gelesen und in die Step-Loop-Felder übertragen. Dazu ist kein spezielles Modul innerhalb der LOOP-Schleife erforderlich, falls im Dynpro Felder der internen Tabelle benutzt werden. Falls die Tabelle mehr Zeilen besitzt, als auf einmal auf dem Bildschirm dargestellt werden können, erhält das Dynpro automatisch einen Rollbalken. Mit diesem Rollbalken kann die interne Tabelle im Loop-Bereich gerollt werden. Die Kommunikation zwischen Rollbalken und LOOP-Anweisung erfolgt automatisch, es müssen keine Funktionscodes erzeugt oder ausgewertet werden.

Der Cursor fungiert als Datensatzzeiger. Er sollte vom Feld SY-INDEX abgeleitet werden. Innerhalb der LOOP-Anweisungen enthält er den Index des gerade bearbeiteten Datensatzes, außerhalb der LOOP-Anweisung den Index des ersten in der Loop angezeigten Datensatzes. Der Wert dieses Feldes kann im Programm

dynamisch gesetzt werden. Dies hat, ebenso wie die Betätigung des Rollbalkens, eine Bewegung durch die Liste zur Folge.

Zum Zeitpunkt PAI ist der CURSOR-Zusatz nicht erforderlich. Allerdings wird der Wert für den Cursor innerhalb der LOOP-Schleife automatisch aktualisiert. Das Zurückschreiben der Werte aus dem Step Loop in die interne Tabelle erfordert ein dafür vorgesehenes Modul. Innerhalb dieses Moduls können Sie den Wert des Feldes cursor einsetzen, um den jeweiligen Datensatz zu identifizieren.

```
LOOP AT itab.
  [ MODULE modify_itab. ]
ENDLOOP.
```

Beide Varianten der LOOP-Anweisung können mit zusätzlichen Optionen versehen werden. Erwähnenswert ist vor allem der Zusatz

```
... INTO record
```

Mit diesem Zusatz wird der Datensatz einer internen Tabelle in einen separaten Arbeitsbereich geschrieben. Er ist erforderlich, um Tabellen ohne eigene Kopfzeile bearbeiten zu können.

Die LOOP-Anweisung für eine interne Tabelle kann zusätzlich zum Zusatz CURSOR auch noch die Zusätze

```
FROM first
```

und

```
TO last
```

verarbeiten. Die Felder first und last müssen ebenfalls von SY-INDEX abgeleitet werden. Sie müssen im Programm dynamisch versorgt werden, Direktwerte sind nicht zulässig. Die Angaben schränken den Scrollbereich auf die Datensätze zwischen den beiden Grenzen ein. Fehlt einer der Werte, so wird als Grenze der erste bzw. der letzte Datensatz angenommen.

Eine weitere Variante der LOOP-Anweisung ist zum Bearbeiten von Table Controls erforderlich. Für jeden Table Control existiert ein globales Datenobjekt mit einem eindeutigen Namen. Ähnlich wie bei Step Loops werden die einzelnen Zeilen eines Table Control durch die LOOP-Anweisung in der Ablauflogik durchlaufen. Die Grundform dieser Anweisung, die in dieser Form sowohl im PBO- als auch im PAI-Teil aufgerufen werden muss, lautet

```
LOOP WITH CONTROL tableview.
  MODULE ... .
ENDLOOP.
```

Dabei steht tableview für das mit der Anweisung

```
CONTROLS tableview TYPE TABLEVIEW USING SCREEN dynpronumber.
```

deklarierte Element. Für jede Zeile des Table Control wird ein Modul aufgerufen, in dem die einzelnen Felder der Tabellenzeile bearbeitet werden können (Füllen der Felder im PBO-Abschnitt, Auswerten im PAI-Abschnitt). Die Felder sind wie bei Loops über ihre Feldnamen erreichbar. Falls der Index der gerade aktuellen Zeile des Table Control erforderlich ist, kann er dem Feld

```
tableview-current_line
```

entnommen werden.

In dieser einfachen Form müssen die Felder des Table Control manuell mit Werten gefüllt werden. Diese Aufgabe wird wie bei einfachen Step Loops durch Angabe eines Cursors erheblich erleichtert. Mit dieser Anweisung können Sie eine interne Tabelle mit einem Table Control verbinden. Die Anweisung entspricht der bereits beschriebenen Variante für interne Tabellen, nur dass zusätzlich der Name des Table View angegeben wird. Zum Zeitpunkt PBO ist dazu die Anweisung

```
LOOP AT itab WITH CONTROL tableview CURSOR cursor.
ENDLOOP.
```

erforderlich. Sie sorgt dafür, dass die interne Tabelle durchlaufen und jede Zeile in eine Zeile des Table Views gestellt wird. Falls die Feldnamen im Table Control mit denen der internen Tabelle übereinstimmen, ist es nicht notwendig, dazu ein Modul aufzurufen. Der Cursor ist wiederum ein programminternes Datenfeld, das von `SY-INDEX` abgeleitet werden kann.

Zum Zeitpunkt PAI müssen Sie die Datensätze des Table View lesen, prüfen und in die interne Tabelle zurückschreiben. Die dazu notwendige `LOOP`-Anweisung und das aufzurufende Modul sind identisch mit den bereits beschriebenen Anweisungen für eine Loop über interne Tabellen.

```
LOOP AT itab.
  MODULE ... .
ENDLOOP.
```

Die Anweisung CALL SUBSCREEN

Der Begriff Subscreen wurde bereits erläutert. Es handelt sich dabei um einen speziellen Dynprotyp. Um einen Subscreen zur Laufzeit in ein anderes Dynpro einzufügen, muss dieses über einen so genannten Subscreen-Bereich verfügen, der durch einen eindeutigen Namen identifiziert ist. Zur Laufzeit wird ein Subscreen-Dynpro mit der Anweisung

```
CALL SUBSCREEN name INCLUDING program screen.
```

in ein anderes Dynpro eingefügt. Diese Anweisung muss natürlich im PBO-Teil der Ablauflogik stehen. Der Parameter `name` ist der Name des Subscreen-Bereichs, so wie er im Screen Painter erfasst wurde. Er steht direkt, also ohne Anführungsstriche, in der Anweisung. Die beiden anderen Parameter `program` und `screen` bezeichnen das Subscreen-Dynpro. Diese beiden Werte müssen entweder als

Zeichenkette, also in einfache Anführungsstriche eingeschlossen, in der Anweisung stehen, oder aber es werden Datenfelder verwendet, welche die gewünschten Werte enthalten. Die Verwendung von Feldern ermöglicht den wirklich dynamischen Aufruf von Subscreens.

Im PAI-Teil des übergeordneten Dynpros wird eine weitere, etwas einfacher gehaltene Anweisung erforderlich:

```
CALL SUBSCREEN name.
```

Beim Aufruf eines Subscreen-Dynpros mit `CALL` wird ein Teil seiner Ablauflogik ausgeführt. Steht das `CALL`-Kommando im PBO-Teil, erfolgt natürlich nur die Ausführung der PBO-Module. Die `CALL`-Anweisung im PAI-Teil des übergeordneten Dynpros führt dann die PAI-Module des eingebetteten Dynpros aus.

Eingebettete Dynpros dürfen weder die Oberfläche des aufrufenden Dynpros ändern noch den Funktionscode verarbeiten. In der Feldliste der Subscreen-Dynpros darf dem Feld zur Übergabe des Funktionscodes also kein Datenfeld zugewiesen werden. In der Ablauflogik eines Subscreen-Dynpros sind daher vor allem Initialisierungen der Dynpro-Felder bzw. Feldprüfungen sinnvoll.

Falls der Subscreen-Bereich nicht direkt in einem Dynpro definiert wurde, sondern über einen Tab Strip bereitgestellt wird, muss in der `CALL SUBSCREEN`-Anweisung anstelle des Programmnamens der Name des Tab Strip eingesetzt werden.

Kontextsensitive und Werthilfe

Bei der Beschreibung der Attribute eines Dynpro-Feldes wurde der Begriff der Werthilfe bereits erwähnt. Für jedes Eingabefeld auf dem Dynpro kann durch verschiedene Mechanismen der zur Verfügung stehende Wertevorrat angezeigt werden. Zwei der Hilfsmöglichkeiten (implizite Werthilfe und Suchhilfe) beruhen auf Dictionary-Elementen und werden dort beschrieben. Diese Elemente nutzen vor allem im Dictionary definierte Abhängigkeiten zwischen Tabellen, so genannte Fremdschlüssel, aus. Ihre Einsatzmöglichkeiten sind zwar recht groß, in speziellen Fällen reicht die angebotene Funktionalität aber nicht aus. In diesen Fällen kann für jedes Dynpro-Feld eine eigene Werthilfe geschrieben werden. Dies erfolgt durch Aufruf eines entsprechenden Moduls zum Zeitpunkt `PROCESS ON VALUE-REQUEST`. Der Aufruf ähnelt dem einer Feldprüfung:

```
PROCESS ON VALUE-REQUEST.
  FIELD value1 MODULE help_values.
```

Diese Anweisung sorgt dafür, dass im Dynpro neben dem Eingabefeld `value1` das Symbol für die Werthilfe erscheint. Die Betätigung dieses Symbols mit der Maus ist gleichbedeutend mit der F4-Taste. Wird diese Taste betätigt, arbeitet die Anwendung nur die Anweisungen nach dem Zeitpunkt `PROCESS ON VALUE-REQUEST` ab. Falls für das Feld, in dem sich der Bildschirmcursor in diesem Moment befand, mittels der `FIELD`-Anweisung eine Verbindung zu einem Modul existiert, wird dieses Modul abgearbeitet. Für den Inhalt dieses Moduls

existieren keine Einschränkungen. Auf irgendeine Art und Weise wird ein Wert ermittelt und am Ende des Moduls in das Dynpro-Feld gestellt. Als einfacher Test zur Verdeutlichung des Prinzips können obiger Zusatz zur Ablauflogik und das folgende Modul in die Beispielanwendung SAPMY410 eingebaut werden.

```
MODULE help_values.
  value1 = '12345679'.
ENDMODULE.
```

Falls für ein Feld eine solche Werthilfe existiert, werden eventuell existierende andere Werthilfen für dieses Feld (Fremdschlüssel oder Suchhilfe) wirkungslos.

Ähnlich wie die Werthilfe arbeitet die kontextsensitive Hilfe, die mit der Funktionstaste F1 aufgerufen wird. Für Datenelemente kann im Dictionary neben den vier unterschiedlich langen Schlüsselwörtern und der Kurzbeschreibung auch noch eine ausführliche Dokumentation erfasst werden. Ein Druck auf die F1-Taste zeigt in einem Popup diese Dokumentation an, sofern sie existiert. Falls nicht, erscheint zumindest die Kurzbeschreibung.

Falls in speziellen Fällen die Dokumentation nicht aussagekräftig genug ist oder für Felder ohne Dictionary-Bezug ebenfalls ein Hilfstext angezeigt werden soll, muss die selbst programmierte Hilfefunktion zum Einsatz kommen. Die Anwendungsweise unterscheidet sich von der Werthilfe durch die andere Zeitpunktanweisung (PROCESS ON HELP-REQUEST) und die erforderliche Funktionalität des Moduls. In der Praxis wird relativ selten von dieser Möglichkeit Gebrauch gemacht.

Nach Ausführung von Modulen der individuellen Eingabe- oder Werthilfe wird die Abarbeitung des Dynpros mit dem Zeitpunkt PBO von vorn begonnen. Der Zeitpunkt PAI wird nicht prozessiert.

LOOP für Datenbanktabellen

Neben der möglichen Pflege von internen Tabellen kann eine weitere Variante von LOOP direkt mit Datenbanktabellen zusammenarbeiten. Die Syntax für die LOOP-Anweisung lautet dann

```
LOOP AT databasetable.
ENDLOOP.
```

Diese Anweisung führt eine Reihe von Aktionen im verborgenen durch. Ihre Funktionalität ist daher wesentlich größer, als die Ähnlichkeit mit der Grundform der LOOP-Anweisung vermuten lässt. Diese Variante der LOOP-Anweisung setzt ein speziell aufgebautes Dynpro voraus. Neben dem eigentlichen Loop-Bereich, dessen Eingabefelder mit der Kopfzeile der Datenbanktabelle verbunden werden, muss im Dynpro ein zweiter Eingabebereich für die Schlüsselfelder der Tabelle existieren. Diese Eingabefelder werden ebenfalls mit der Datenbanktabelle verbunden, wobei aber die Sternvariante des Namens benutzt wird. Die Tabellen müssen nicht mit der TABLES-Anweisung deklariert werden.

Die typische Ablauflogik besteht aus wenigen Zeilen:

```
PROCESS BEFORE OUTPUT.
  LOOP AT databasetable.
  ENDLOOP.

*
PROCESS AFTER INPUT.
  LOOP AT databasetable.
    MODIFY databasetable.
  ENDLOOP.
```

Das PBO-`LOOP`-Kommando füllt die Loop-Blöcke mit den ersten Datensätzen der Tabelle. Im PAI-`LOOP` werden die Inhalte der Loop-Blöcke in die Tabelle zurückgeschrieben. Dazu ist ausnahmsweise kein Modul erforderlich. In Loops über Dictionary-Tabellen – und nur dort – steht eine Dynpro-Variante des Kommandos `MODIFY` zur Verfügung.

Die verborgenen Funktionen dieser `LOOP`-Anweisung betreffen das Blättern in der Tabelle. Bei Betätigung der Datenfreigabetaste werden die Inhalte der auf der *-Tabelle beruhenden Eingabefelder gelesen. Diese Werte werden in der Datenbanktabelle gesucht. Wenn ein derartiger Satz existiert, wird er zum ersten Satz im Loop-Bereich. Falls kein Satz existiert, wird der zum Suchmuster nächstgrößere Satz in den ersten Loop-Block gestellt. Sind die Felder für das Suchmuster leer, blättert die Ablauflogik ein Bild weiter.

Die `LOOP`-Schleife über eine Datenbanktabelle wird in eigenen Programmen nur selten eingesetzt, da mit anderen Programmiermethoden wesentlich komfortablere Anwendungen möglich sind. Bis zum Release 2.2 wurde diese Form der `LOOP`-Anweisung vor allem in automatisch generierten Pflegeprogrammen benutzt. Für bestimmte Tabellen konnte im Data Dictionary eine so genannte *Allgemeine Tabellenpflege* erlaubt werden. Mittels einer speziellen Menüfunktion (TABELLENDYNPRO ERZEUGEN) erzeugte das Dictionary ein Programm und ein Dynpro, das den oben aufgeführten Bedingungen genügt. Diese Programme können indirekt über die Transaktion `SM31` oder die Funktion SYSTEM | DIENSTE | ALLGEMEINE TABELLENPFLEGE aufgerufen werden. Die erwähnte Transaktion fordert vom Anwender den Namen einer Tabelle und ruft dann das entsprechende Pflege-Dynpro auf. Von dieser einfachen Möglichkeit der Datenpflege wird vor allem bei der Bearbeitung verschiedener Systemtabellen und beim Customizing Gebrauch gemacht. Es ist durchaus üblich, die Bausteine zur allgemeinen Tabellenpflege nach ihrer Erstellung manuell weiterzubearbeiten. Bei einer eventuellen Neuerstellung des Tabellen-Dynpros gehen diese Ergänzungen allerdings verloren. Die erzeugten Modul-Pools beginnen, abweichend von den sonstigen Namenskonventionen, mit den drei Buchstaben `MST`, gefolgt vom Namen der Tabelle. Da der Name eines Modul-Pool maximal acht Stellen lang sein darf, kann ein solches Pflegeprogramm nur für Tabellen erzeugt werden, deren Namen aus maximal fünf Zeichen bestehen.

Ab Release 3.0 stehen für die Tabellenpflege andere Werkzeuge bereit, die wesentlich komplexere Pflegeprogramme generieren. Die eben beschriebene Variante der LOOP-Anweisung hat in diesen Programmen keine Bedeutung mehr.

3.7.3 Komplexe Elemente im Dynpro – Ein Beispiel

Zur Demonstration der Subscreens sowie der anderen komplexeren Elemente soll nun ein kleines Beispiel vorgestellt werden. Es besitzt keinerlei praktische Bedeutung, sondern kombiniert einige der vorgestellten Kommandos und Elemente zu einer lauffähigen Anwendung. Das Startbild dieses Programms erlaubt es, eine von zwei Eingabemaske in einem Subscreen einzublenden. Die Umschaltung zwischen den Eingabemasken erfolgt durch Drucktasten in der Drucktastenzeile bzw. Menüfunktionen. Die Eingabemasken liefern Suchbegriffe für eine Abfrage in der Tabelle TADIR. Das Suchergebnis wird wahlweise in einem Step Loop oder einem Table Control dargestellt. Diese Elemente werden in normalen Dynpros dargestellt. Eine zweite Ausbaustufe der Anwendung fasst dann die beiden Eingabemasken in einem Tab Strip zusammen.

Legen Sie zunächst einen neuen Modul-Pool SAPMY430 an. Tragen Sie die folgenden Deklarationen im Top Include ein:

```
PROGRAM sapmy430 MESSAGE-ID yz4.
TABLES:
  tadir.

DATA:
  g_subscreen   LIKE sy-dynnr VALUE '1000',
  itadir        LIKE tadir OCCURS 10 WITH HEADER LINE,
  fcode         LIKE sy-tcode,
  excode        LIKE sy-tcode OCCURS 10 WITH HEADER LINE,
  target_screen LIKE sy-dynnr VALUE '2000',
  start_screen  LIKE sy-dynnr VALUE '100',
  i TYPE i.

CONTROLS tc1 TYPE TABLEVIEW USING SCREEN 3000.
CONTROLS ts1 TYPE TABSTRIP.
```

Nach Definition der globalen Daten muss das Programm generiert werden. Für die Funktionsfähigkeit ist diese Generierung nicht erforderlich, allerdings ist sie notwendig, damit beim nachfolgenden Bearbeiten des Dynpros das Einfügewerkzeug die programminternen Datenfelder findet.

Erzeugen Sie nun einen Status STAT1, der die in Tabelle 3.40 ersichtlichen Funktionen anbietet. Den Menüzweig EINSTELLUNGEN können Sie an der ersten Position des Menüs anlegen.

Funktionscode	Funktionstyp	Menü	Funktionstaste
FT03	E	Springen I Zurück	F3 (vordefiniertes Symbol in Symbolleiste aktivieren)
FT15	E	Bearbeiten I Abbrechen	F12 (vordefiniertes Symbol in Symbolleiste aktivieren)
SUB1		Einstellungen I Anwender-Daten	F5 (Drucktaste erzeugen)
SUB2		Einstellungen I Objekt-Daten	F6 (Drucktaste erzeugen)
STEP		Einstellungen I Step Loop	F7 (Drucktaste erzeugen)
TABC		Einstellungen I Table Control	F8 (Drucktaste erzeugen)

Tabelle 3.40
Funktionen des Status STAT1

Träger-Dynpro für Subscreens

Subscreens sind Dynpros, die dynamisch in speziell dazu vorgesehene Bereiche eines so genannten Träger-Dynpros eingeblendet werden können. Beide Dynpros müssen bestimmten Anforderungen genügen. Dabei sind die Arbeiten im Träger-Dynpro wesentlich umfangreicher als die im Subscreen. Zunächst wird das Träger-Dynpro erzeugt.

Arbeiten im Layout-Editor

Legen Sie ein neues Dynpro 100 an und wechseln Sie in den Layout-Editor. Im Träger-Dynpro muss nun ein Subscreen-Bereich angelegt werden. Dazu wird der Cursor auf die linke obere Ecke des einzufügenden Bereiches platziert. Anschließend wird die Drucktaste SUBSCREEN betätigt oder die Menüfunktion BEARBEITEN | ANLEGEN ELEMENT | SUBSCREEN gewählt. Nach Bestätigung der Eingabe füllt der Screen Painter den maximal zur Verfügung stehenden Platz im Dynpro mit einem aus Punkten bestehenden farbig hervorgehobenen Block aus. Der Cursor wird auf das gewünschte Ende des Subscreens (rechte untere Ecke) gesetzt. Die endgültige Bestätigung der Größe des Subscreens erfolgt mit der Drucktaste BEREICHSENDE MARKIEREN, einem Doppelklick mit der Maus oder der Menüfunktion BEARBEITEN | SUBSCR. ENDE MARK.

Der Screen Painter erfragt in einem Popup den Namen des Subscreen-Bereichs. Dieser kann freizügig gewählt werden. Hier wurde der Einfachheit halber die Bezeichnung „SUB“ verwendet.

Im Träger-Dynpro sind keine Eingabefelder erforderlich. Allerdings muss das OK-Code-Feld des Dynpros mit dem programminternen Feld `fcode` verknüpft werden, damit der Funktionscode korrekt übergeben werden kann.

Ablauflogik

In der Ablauflogik des Träger-Dynpros rufen Sie zusätzlich zu den bekannten Modulen (Setzen des Status, Auswerten des Funktionscodes) den oder die einzufügenden Subscreens auf. Dazu benutzen Sie das Kommando `CALL SUBSCREEN`. Dieses Kommando muss sowohl zum Zeitpunkt PAI als auch PBO aufgerufen werden. Das Kommando sorgt dafür, dass der korrekte Subscreen im Träger-Dynpro eingeblendet wird. Außerdem ruft es die Module des Subscreens auf. Die Syntax dieses Kommandos in den beiden Zeitpunkten ist unterschiedlich. Zum Zeitpunkt PBO müssen der Name des Subscreen-Bereiches im Träger-Dynpro sowie der Name des Modul-Pools und die Nummer des einzufügenden Dynpros übermittelt werden:

```
CALL SUBSCREEN subscreen_area
     INCLUDING program screen_number.
```

Dabei können die Werte für `program` und `screen_number` entweder als Literale (in einfache Anführungsstriche eingeschlossene Zeichenketten) oder als Inhalt eines Feldes notiert werden. Im letztgenannten Fall werden die Felder nicht – wie bei der dynamischen Übergabe in normalen ABAP-Kommandos – in runde Klammern eingeschlossen, sondern ohne weitere Zusätze eingefügt.

Im PAI-Teil der Ablauflogik ist der `INCLUDING`-Zusatz nicht mehr erforderlich:

```
CALL SUBSCREEN subscreen_area.
```

Im Träger-Dynpro sollten keine Prüfmodule aufgerufen werden, die sich auf Felder in Subscreens beziehen. Da Subscreens dynamisch ausgetauscht werden können, würden Prüfungen auf Felder in nicht aktiven Subscreens u. U. unnötige Fehlermeldungen verursachen. Außerdem würde dies die Modularisierung, die mit Subscreens erreicht werden soll, beeinträchtigt.

Die Ablauflogik des Dynpros 100 sieht wie folgt aus:

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
  CALL SUBSCREEN sub INCLUDING 'SAPMY430' g_subscreen .
*
PROCESS AFTER INPUT.
  MODULE exit_0100 AT EXIT-COMMAND.
  CALL SUBSCREEN sub.
  MODULE user_command_0100.
```

Im Modul status_0100 wird lediglich der Status gesetzt. Außerdem wird die Nummer des Start-Dynpros in einer globalen Variablen aufbewahrt. Dieses Feld wird später zur Navigation genutzt, da für den Tab Strip ein zweites Start-Dynpro geschaffen wird. Der Quelltext des Moduls ist denkbar einfach.

```
MODULE status_0100 OUTPUT.
  SET PF-STATUS 'STAT1'.
  start_screen = sy-dynnr.
ENDMODULE.                                 " STATUS_0100  OUTPUT
```

Die nachfolgende CALL-Anweisung der Ablauflogik bindet in den Subscreen-Bereich sub ein Subscreen-Dynpro ein, dessen Dynpro-Nummer im Feld g_subscreen steht. Auf diese Weise kann der Subscreen dynamisch geändert werden. Dies wird aus dem Quelltext des User-Command-Moduls (Funktionscodes SUB1 und SUB2) ersichtlich:

```
MODULE user_command_0100 INPUT.
  CASE fcode.
    WHEN 'SUB1'.
      g_subscreen = '1000'.
    WHEN 'SUB2'.
      g_subscreen = '1010'.
    WHEN 'STEP'.
      target_screen = '2000'.
    WHEN 'TABC'.
      target_screen = '3000'.
    WHEN 'SEL1'.
      SELECT * FROM tadir
        INTO TABLE itadir
        WHERE devclass = tadir-devclass
          AND author   = tadir-author.
      IF sy-dbcnt > 0.
        LEAVE TO SCREEN target_screen.
      ENDIF.
    WHEN 'SEL2'.
      TRANSLATE tadir-obj_name USING '*%'.
      SELECT * FROM tadir
        INTO TABLE itadir
        WHERE pgmid = tadir-pgmid
          AND object = tadir-object
          AND obj_name LIKE tadir-obj_name.
      IF sy-dbcnt > 0.
        LEAVE TO SCREEN target_screen.
      ENDIF.
  ENDCASE.

ENDMODULE.                 " USER_COMMAND_0100  INPUT
```

Die volle Funktionalität erschließt sich erst nach Kenntnis der Funktion und des Aufbaus der beiden Subscreen-Dynpros. Im Dynpro 1000 stehen zwei Eingabefelder für die Entwicklungsklasse (`tadir-devclass`) und den Programmautor (`tadir-author`) und eine Drucktaste, die den Funktionscode `SEL1` auslöst, zur Verfügung. Falls diese Drucktaste betätigt wird, dienen die Inhalte dieser beiden Felder zur Selektion von Einträgen aus der Tabelle `TADIR`. Die gelesenen Einträge werden in einer internen Tabelle `itadir` aufbewahrt. Ähnlich funktioniert das Dynpro 1010, das die drei Felder `tadir-pgmid`, `tadir-object` und `tadir-obj_name` sowie eine Funktionstaste für den Funktionscode `SEL2` besitzt.

Nach der Selektion der Datensätze wird ein Folge-Dynpro zur Darstellung der gefundenen Datensätze aufgerufen. Da die Darstellung sowohl in einem Step Loop als auch in einem Table View möglich sein soll, wird auch die Nummer des zur Darstellung aufzurufenden Dynpros dynamisch gesetzt. Im Startbild kann diese Nummer durch die beiden Funktionscodes `STEP` und `TABV` bzw. die entsprechenden Drucktasten in der Drucktastenzeile gewählt werden.

Zur Auswertung der beiden Exit-Funktionstasten reicht im Start-Dynpro ein sehr einfaches Modul.

```
MODULE exit_0100 INPUT.
  LEAVE TO SCREEN 0.
ENDMODULE.                                " EXIT_0100  INPUT
```

Subscreen-Dynpros anlegen

Das Subscreen-Dynpro kann wie jedes andere Dynpro auch erstellt werden. Folgende Punkte sind zu beachten:

- In der Feldliste des Subscreen-Dynpros darf das OK-Code-Feld nicht belegt werden.
- Das Subscreen-Dynpro darf keinen Status setzen.
- Die im Subscreen-Dynpro belegte Fläche muss auf den im Träger-Dynpro zur Verfügung stehenden Platz abgestimmt sein.
- Das Dynpro-Attribut SUBSCREEN muss gesetzt sein.

Zunächst wird der Subscreen 1000 angelegt, der über zwei Eingabefelder zur Erfassung der Suchbegriffe und eine Drucktaste verfügen muss. Legen Sie dazu im eben erzeugten Modul-Pool ein Dynpro 1000 an. Setzen Sie das Attribut Subscreen (siehe Abbildung 3.136).

Im Fullscreen werden die beiden bereits erwähnten Eingabefelder `tadir-author` und `tadir-devclass` für Entwicklungsklasse und Programmautor eingetragen. Abbildung 3.137 zeigt die Ansicht des Dynpros während der Bearbeitung.

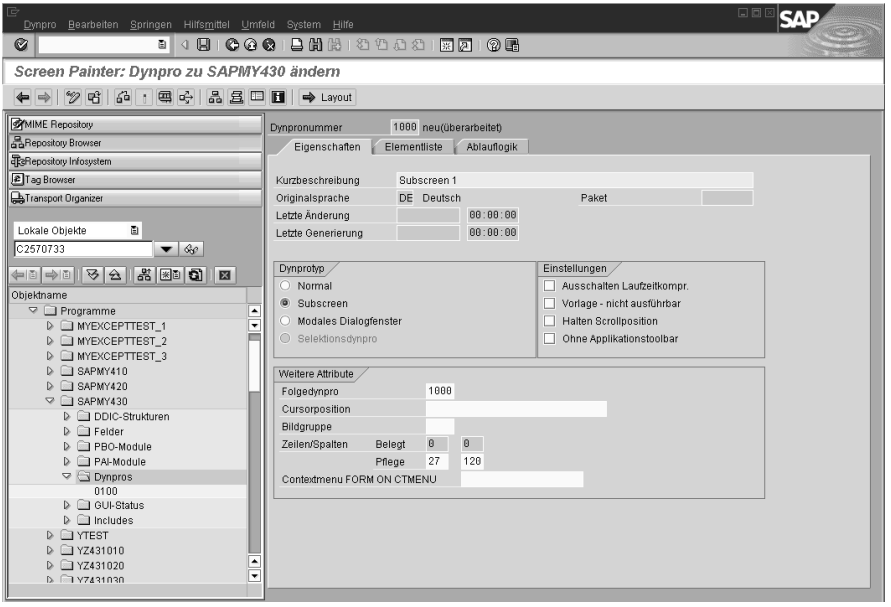

Abbildung 3.136 **© SAP AG**
Attribute eines Subscreen-Dynpros

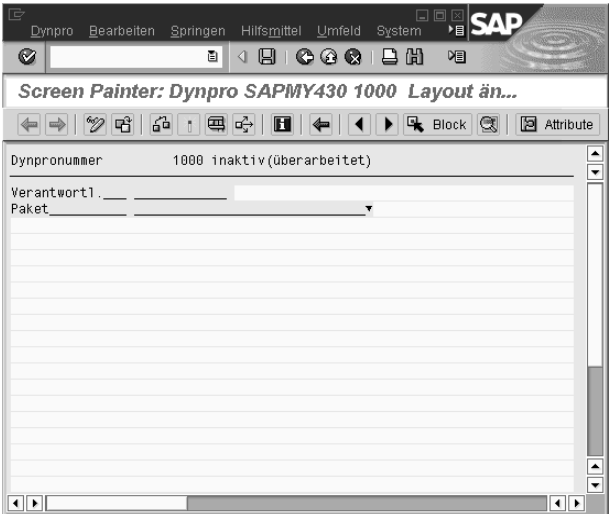

Abbildung 3.137 **© SAP AG**
Subscreen-Dynpro während der Bearbeitung

Um die Drucktaste zu erzeugen, tragen Sie den Text der Taste an der gewünschten Stelle im Dynpro ein (siehe Abbildung 3.138).

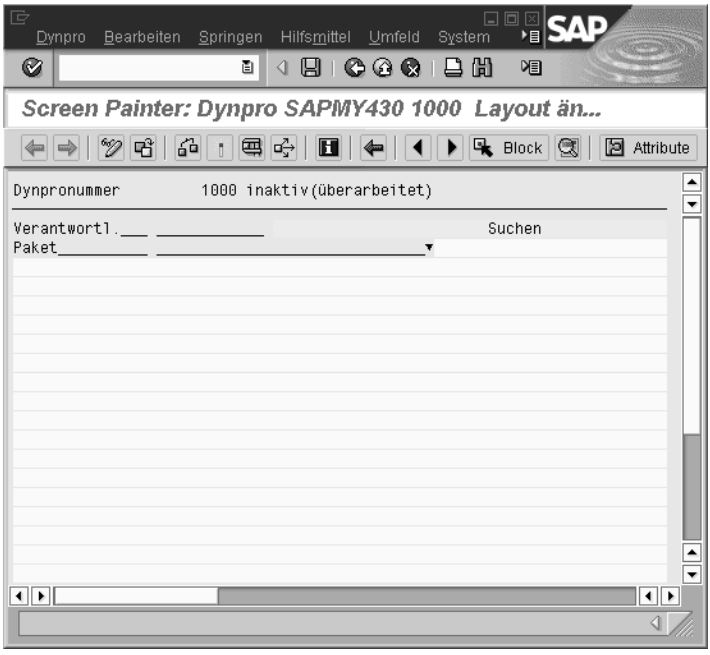

Abbildung 3.138 **© SAP AG**
Anlegen einer Drucktaste

Anschließend positionieren Sie den Cursor auf den Text und betätigen die Drucktaste GRAF.ELEMENT. Der Layout-Editor hebt den selektierten Text farblich hervor und bietet andere Bearbeitungsmöglichkeiten an (Abbildung 3.139).

Der Cursor steht immer noch auf dem Textfeld. Sie können dieses hervorgehobene Feld nun mit der Schaltfläche DRUCKTASTEN in eine Drucktaste umwandeln. Nach der Umwandlung, über deren Erfolg Sie durch eine Nachricht in der Statuszeile informiert werden, können Sie mit der Funktionstaste F3 oder der entsprechenden Menüfunktion zum Hauptbild des Layout-Editors zurückkehren. Dort wird die soeben erzeugte Drucktaste farblich hervorgehoben (Abbildung 3.140).

Abbildung 3.139 **© SAP AG**
Bearbeiten eines grafischen Elements

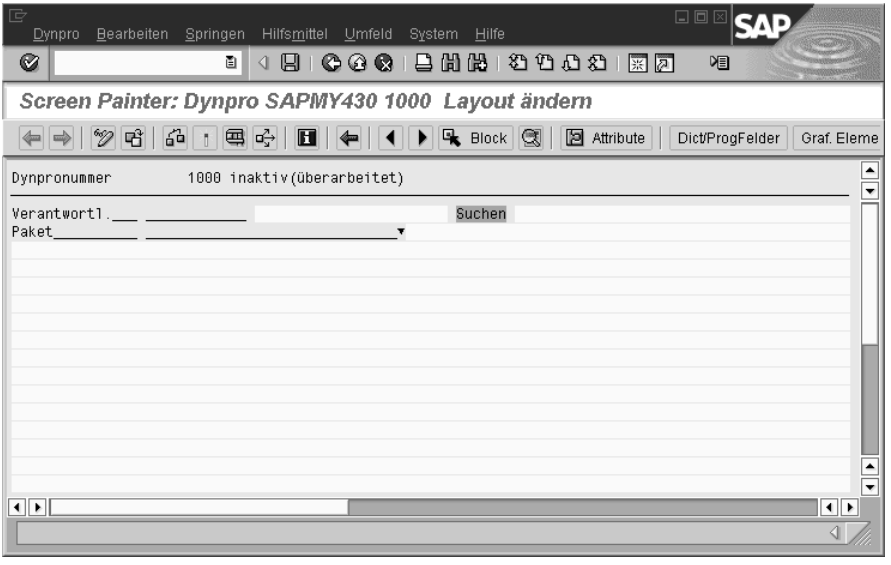

Abbildung 3.140 **© SAP AG**
Dynpro mit Drucktaste

Zum Abschluss muss dieser Drucktaste noch ein Funktionscode zugewiesen werden. Setzen Sie dazu den Cursor auf die Schaltfläche und betätigen Sie die ATTRIBUTE-Schaltfläche. Im nun aktiven Popup (Abbildung 3.141) tragen Sie im Feld NAME einen beliebigen Namen für die Drucktaste sowie im Feld FKTCODE den Funktionscode `SEL1` ein.

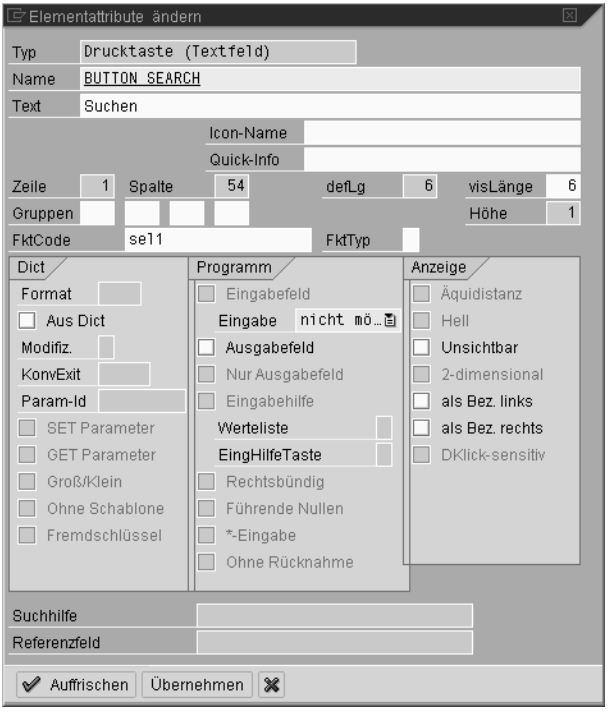

Abbildung 3.141 **© SAP AG**
Zuweisen des Funktionscodes zur Drucktaste

Auf dieselbe Art und Weise legen Sie ein weiteres Subscreen-Dynpro 1010 an. Nehmen Sie in dieses Dynpro die Felder `tadir-pgmid`, `tadir-object` und `tadir-obj_name` sowie eine Drucktaste für den Funktionscode `SEL2` auf.

Ablauflogik der Subscreen-Dynpros

Beide Dynpros verfügen natürlich über eine eigene Ablauflogik, die allerdings sehr rudimentär ist, da das User-Command-Modul auf jeden Fall fehlt. Ein Status wird ebenfalls nicht gesetzt. Der PAI-Teil ist somit völlig leer. Nur im PBO-Abschnitt der Ablauflogik erfolgt eine Initialisierung der Eingabefelder. Stellvertretend für die Ablauflogiken hier die des Dynpro 1000.

```
PROCESS BEFORE OUTPUT.
  MODULE init_1000.
*
PROCESS AFTER INPUT.
```

Die Quelltexte des Initialisierungsmoduls sind sehr einfach:

```
MODULE init_1000 OUTPUT.
 tadir-devclass = '$TMP'.
 tadir-author = sy-uname.
ENDMODULE.                            " INIT_1000  OUTPUT
```

Ähnlich einfach verläuft die Initialisierung im Dynpro 1010.

```
MODULE init_1010 OUTPUT.
  tadir-pgmid = 'R3TR'.
  CLEAR tadir-object.
  CLEAR tadir-obj_name.
ENDMODULE.                              " INIT_1010  OUTPUT
```

Nach der Programmierung der Ablauflogik und der beiden Module können Sie die beiden Dynpros generieren.

Step-Loops

Step Loops bestehen aus einer tabellenförmigen Wiederholung eines Blocks, der aus ein oder mehreren Feldern bestehen und sich durchaus über mehrere Zeilen erstrecken kann. Zur Bearbeitung der Inhalte eines Step Loops ist in der Ablauflogik der Einsatz der `LOOP`-Anweisung erforderlich. In diesem Beispiel sollen im Step Loop die Felder `pgmid`, `object`, `obj_name`, `devclass` und `author` der internen Tabelle `itadir` enthalten sein.

Arbeiten im Layout-Editor

Für die Demonstration des Step Loops wird ein weiteres Dynpro mit der Nummer 2000 erforderlich. Es handelt sich dabei nicht um einen Subscreen sondern um ein völlig normales Dynpro. Im Layout-Editor wird zunächst der Vorlageblock für eine Tabellenzeile angelegt. Übernehmen Sie dazu die oben genannten Felder der internen Tabelle `itadir` in das Dynpro. Einige dieser Felder besitzen eine sehr große Länge. Setzen Sie daher zunächst eine sinnvolle Ausgabelänge für die Felder, vor allem für `itadir-obj_name und itadir-devclass`. Die Ausgabelänge wird über das Attribute-Popup geändert. Damit Sie dort im Feld VISLÄNGE einen passenden Wert eintragen können, muss das Flag ROLLBAR markiert werden. Ordnen Sie danach alle Felder in der dritten Zeile des Dynpros an. Die Zeile darüber ist für die Spaltenüberschriften vorgesehen.

Bevor nun wirklich der Step Loop erzeugt wird, sollte die Überschrift eingefügt werden. Übernehmen Sie dazu mit Hilfe des Einfügewerkzeuges die passenden Schlüsselfelder aus der Tabelle TADIR. Die Feldschablonen dürfen dabei nicht übernommen werden. Verschieben Sie anschließend die Schlüsselfelder an die korrekte Position über den Feldschablonen. Unter Umständen müssen Sie die Position der Feldschablonen an die Position der Bezeichner in der Überschrift anpassen oder den Text einzelner Elemente der Überschrift mit Hilfe des Attribute-Popup ändern.

Setzen Sie den Cursor nun auf das erste Feld in der Zeile mit den Feldschablonen. Abbildung 3.142 zeigt den Layout-Editor zu diesem Zeitpunkt.

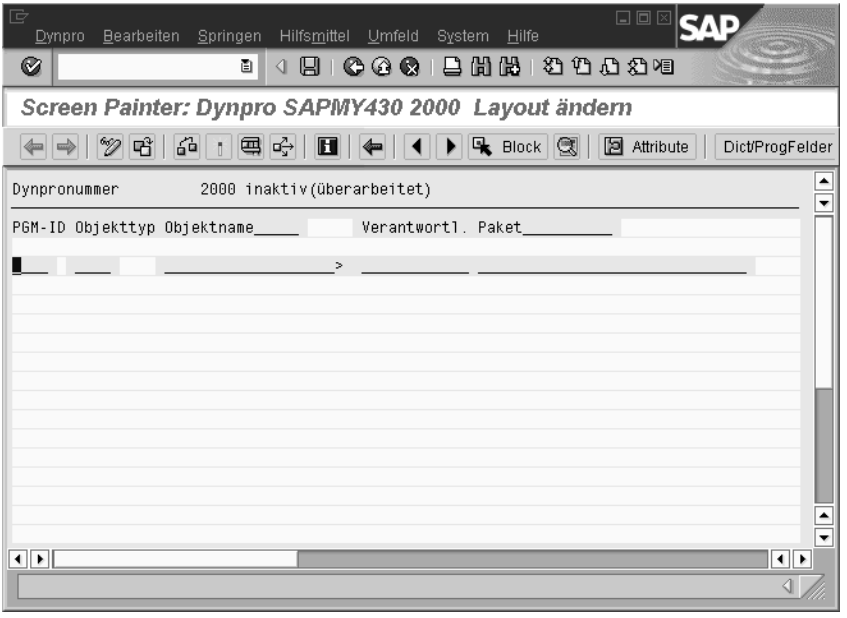

Abbildung 3.142 **© SAP AG**
Layout-Editor mit Tabellenüberschrift und Vorlagezeile

Rufen Sie nun die Menüfunktion ANLEGEN | LOOP auf. Der Layout-Editor wechselt nun in einen anderen Bearbeitungsmodus. Markieren Sie durch einen Doppelklick auf das letzte Feld der Zeile das Ende des Loop Blocks. Der Layout-Editor wechselt nochmals den Modus (siehe Abbildung 3.143)

Im Layout-Editor finden Sie nun eine aus zwei Zeilen bestehende Tabelle vor. Außerdem bietet die Oberfläche einige speziell für Step Loops vorgesehene Bearbeitungsfunktionen. Nach Erzeugen eines Step Loops ist dessen Größe, also die Anzahl der Tabellenzeilen, zunächst fest. Es handelt sich dabei um einen so genannten fixen Loop.

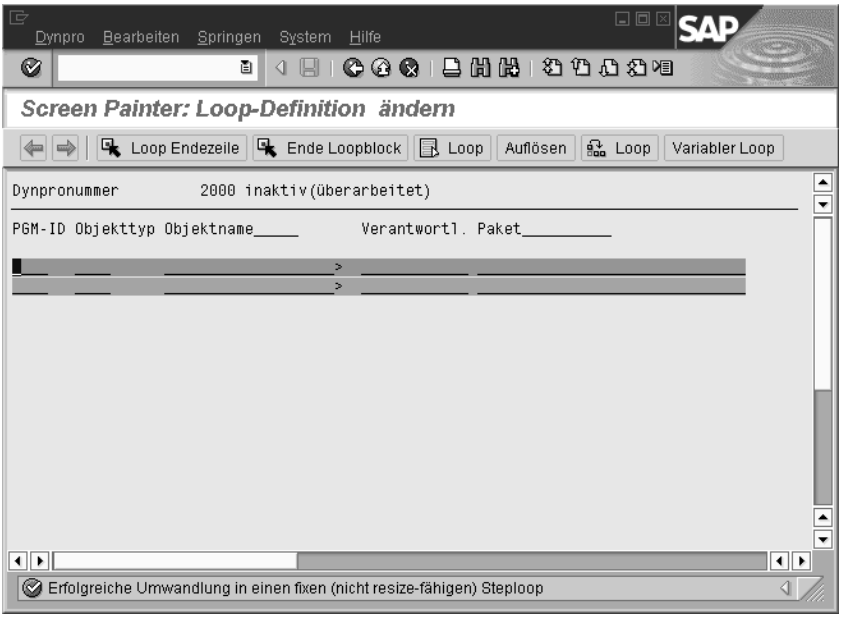

Abbildung 3.143 **© SAP AG**
Step Loop mit zwei Blöcken

Sie können auf die Zahl der Tabellenzeilen Einfluss nehmen, indem Sie den Cursor auf die gewünschte Position stellen und die Drucktaste LOOP ENDEZEILE betätigen oder die Menüfunktion BEARBEITEN | LOOPENDEZEILE MARK auslösen. Sie können den Cursor aber auch auf eine beliebige Stelle im Loop-Bereich platzieren und den Loop mit der Drucktaste VARIABLER LOOP oder der Menüfunktion BEARBEITEN | LOOP UMWANDELN | VARIABLER LOOP in einen variablen Loop umwandeln. In diesem Beispiel sollte die letztgenannte Variante zum Einsatz kommen. Da die Zahl der Zeilen im Loop automatisch an den freien Platz angepasst wird, reichen die beiden Zeilen im Dynpro aus.

Ablauflogik

Auf die Darstellung der individuell programmierten Blätter-Funktion soll verzichtet werden, da sie in der Praxis kaum noch eine Rolle spielt. Hier soll die interne Tabelle `itadir` komplett mit dem Step Loop verknüpft werden. Dazu wird die Ablauflogik des Dynpros 2000 wie folgt erweitert:

```
PROCESS BEFORE OUTPUT.
  LOOP AT itadir CURSOR i.
  ENDLOOP.
*
PROCESS AFTER INPUT.
  MODULE exit_2000 AT EXIT-COMMAND.
```

```
  LOOP AT itadir.
    MODULE modify_itab.
  ENDLOOP.
```

Das Übertragen der Zeilen der internen Tabelle in den Step Loop erfolgt automatisch ohne zusätzliche Anweisungen. Sollen allerdings geänderte Werte aus dem Dynpro zurück in die interne Tabelle übertragen werden, so sind dazu zusätzliche Anweisungen erforderlich. Im PAI-Abschnitt wird zur Demonstration im `LOOP`-Block ein Modulaufruf eingefügt, der diese Aufgabe übernimmt. Das Modul selbst ist wieder sehr einfach:

```
MODULE modify_itab INPUT.
  MODIFY itadir INDEX i.
ENDMODULE.                    " MODIFY_ITAB  INPUT
```

Damit Sie die Anwendung geordnet beenden oder zum Startbild zurückkehren können, muss das Exit-Modul entsprechend gestaltet werden. Natürlich dürfen Sie nicht vergessen, in der Feldliste das OK-Code-Feld mit dem programminternen Feld `fcode` zu verknüpfen.

```
MODULE exit_2000 INPUT.
  CASE fcode.
    WHEN 'FT03'.
      LEAVE TO SCREEN start_screen.
    WHEN 'FT15'.
      LEAVE TO SCREEN 0.
  ENDCASE.
ENDMODULE.                                " EXIT_2000  INPUT
```

Nach Generieren und Aktivieren aller Teile der Anwendung sowie Erzeugen einer Dialogtransaktion (`Y430`) können Sie die Anwendung bereits testen. Abbildung 3.144 zeigt die fertige Anwendung.

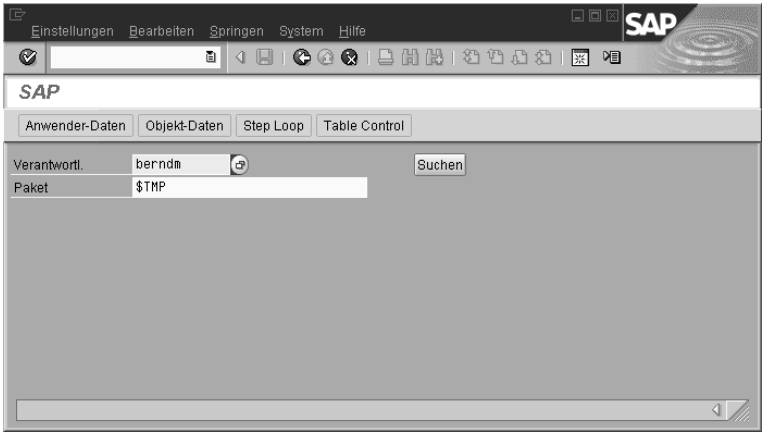

Abbildung 3.144 **© SAP AG**
Startbild der Anwendung

Wählen Sie über eine der beiden ersten Drucktasten eine der beiden Eingabemasken. Tragen Sie dort die von Ihnen gewünschten Selektionskriterien ein und betätigen Sie die SUCHEN-Schaltfläche im Dynpro. Falls Datensätze gefunden werden, die Ihrem Selektionswunsch entsprechen, listet die Anwendung diese Datensätze im Dynpro 2000 auf. Sie können dort mit dem Rollbalken durch die Tabelle blättern. Es ist auch möglich, Daten zu ändern. Diese Änderungen haben keinen Einfluss auf den Inhalt der Datenbanktabelle. Sie werden allerdings in der internen Tabelle abgespeichert. Das Umschalten zum Table Control ist momentan noch nicht möglich, da dieses Dynpro noch nicht programmiert wurde. Mit der Funktionstaste F3 gelangen Sie zurück zum Startbild, mit F12 können Sie die Anwendung beenden.

Table Control

Ab Version 3 der R/3-Software existiert neben den Step Loops ein weiteres Element, um Daten in Tabellenform darzustellen. Es handelt sich dabei um das so genannten Table Control, dessen Aussehen an die unter verschiedenen Windows-Anwendungen verfügbaren Tabellen angelehnt ist. Allerdings betreffen die Änderungen nicht nur das Aussehen, sondern auch die Funktionalität und das Programmiermodell.

Arbeiten im Layout-Editor

Legen Sie zunächst ein neues Dynpro mit der Nummer 3000 an. Es handelt sich wieder um ein normales Dynpro, keinen Subscreen. Positionieren Sie den Cursor in die linke obere Ecke der Arbeitsfläche und rufen Sie die Menüfunktion BEARBEITEN | ANLEGEN ELEMENT | TABLE CONTROL auf. Der Layout-Editor reserviert nun den gesamten freien Platz im Dynpro für das neue Control (siehe Abbildung 3.145). Sie müssen nun durch einen Doppelklick in der markierten Fläche die Größe des Controls festlegen. Sie können die Abmessungen des Controls später noch ändern. An dieser Stelle können Sie die Ausdehnung daher so groß wie möglich wählen.

Unmittelbar nach dem Doppelklick zur Bestimmung der Größe erscheint automatisch ein Popup, in dem Sie die Attribute für das Control festlegen müssen (Abbildung 3.146).

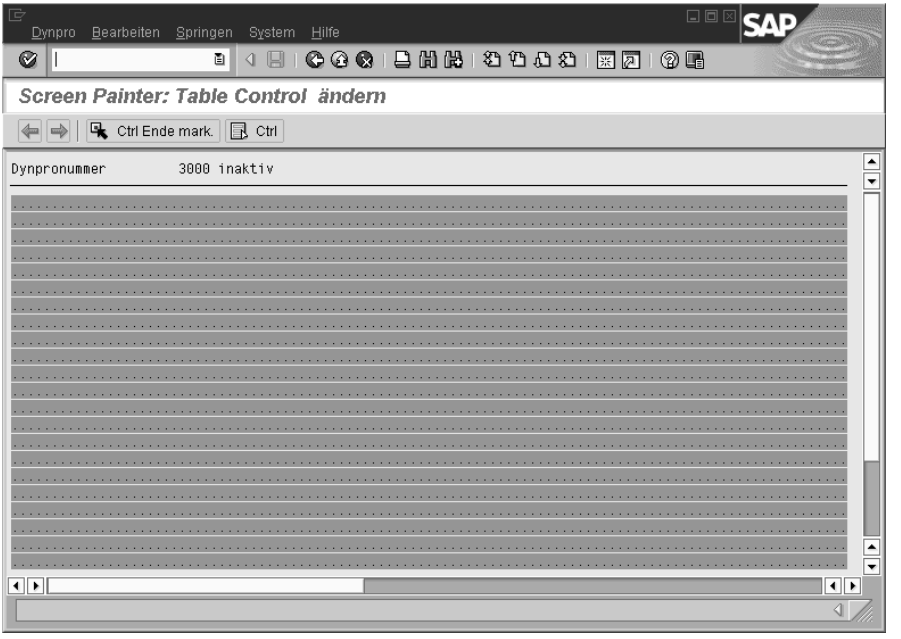

Abbildung 3.145 **© SAP AG**
Reservierte Fläche für ein Table View

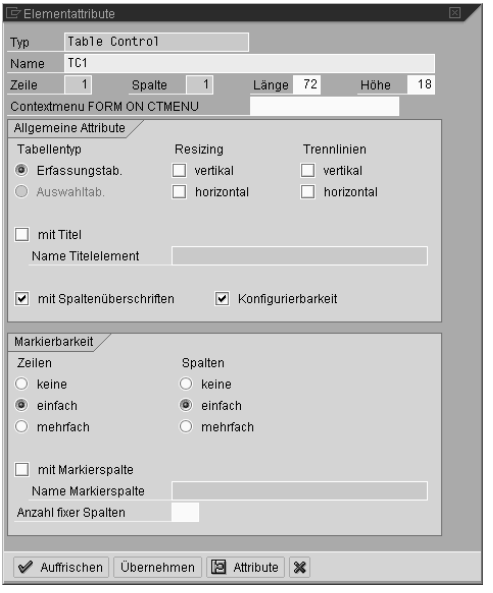

Abbildung 3.146 **© SAP AG**
Attribute für den Table View

In diesem Popup müssen Sie zwei Änderungen durchführen. Tragen Sie im Feld NAME den Wert `TC1` ein. Das entsprechende Control wurde bereits am Anfang des Beispiels im Top Include deklariert. Die zweite Änderung betrifft die Zeilenmarkierspalte. Im Rahmen MARKIERBARKEIT existiert ein Ankreuzfeld mit der Bezeichnung MIT MARKIERSPALTE. Dieses Feld ist standardmäßig aktiviert. Für dieses Beispiel ist es auszuschalten. Die Attribute können Sie durch die Schaltfläche ÜBERNEHMEN bestätigen. Im Layout-Editor erscheint nun ein leeres Control (Abbildung 3.147). Sie müssen dieses Dynpro nicht verlassen, die weitere Bearbeitung ist von diesem Dynpro aus möglich.

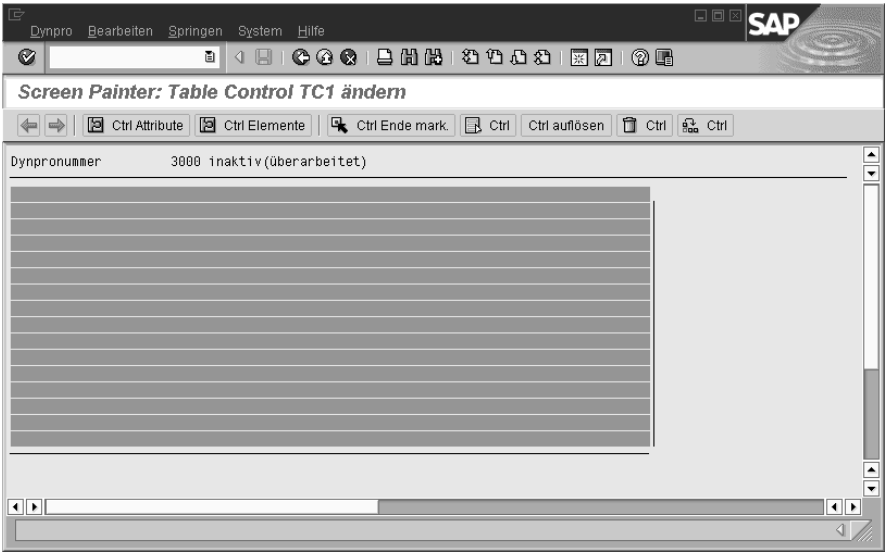

Abbildung 3.147 **© SAP AG**
Layout-Editor mit leerem Table View

Dem Table View müssen nun die darzustellenden Felder zugewiesen werden. Wählen Sie aus den angebotenen Funktionen die Drucktaste CTRLELEMENTE. Diese Funktion aktiviert ein Popup (Abbildung 3.148) in dem die Spalten des Table Views definiert werden.

In diesem Popup können Sie die gewünschten Felder manuell eintragen. Sie können aber auch das bereits bekannte Einfügewerkzeug benutzen, das über die Drucktaste DICT/PROGFELDER erreichbar ist. Wählen Sie, so wie bereits beim Step Loop beschrieben, die fünf Felder der internen Tabelle `itadir` aus. Nach der Auswahl im Einfügewerkzeug setzen Sie im Table View-Popup den Cursor in die erste Zeile der Spalte TABELLENSPALTE. Dann betätigen Sie die Drucktaste EINFÜGEN. Diese Taste ist nur als Symbol vorhanden! Alternativ zur Drucktaste können Sie auch die Tastenkombination Strg + ⇧ + F1 benutzen.

Abbildung 3.148 **© SAP AG**
Spalten des Table View definieren

Ebenfalls mit dem Einfügewerkzeug übernehmen Sie die Bezeichner aus der Tabelle `TADIR` in die Spalte ÜBERSCHRIFT ZUR TABELLENSPALTE. Dies muss aber für jedes Feld einzeln erfolgen. Sie können die Feldnamen auch per Hand eintragen, dann fragt das System aber in mehreren aufeinander folgenden Popups Eigenschaften zum Feld ab.

Nach dem Übernehmen der Felder sollten Sie wiederum die Ausgabelängen für `itadir-obj_name` und `itadir-devclass` ändern. Dies erreichen Sie durch Überschreiben der Werte in der Spalte VISLG. Abbildung 3.149 zeigt Ihnen das fertig ausgefüllte Popup.

Abbildung 3.149 **© SAP AG**
Fertig ausgefülltes Popup für Table View-Struktur

Nach Bestätigen der Eingaben im Popup gelangen Sie zurück in den Layout-Editor. Die zuvor leere Fläche des Controls enthält nun Platzhalter für die Überschriften und die Tabellenfelder. Bei dieser Gelegenheit können Sie die Abmessungen des Table View auf die wirklich erforderliche Größe verringern. Positionieren Sie dazu den Cursor auf die rechte untere Ecke und betätigen Sie die Drucktaste CTRLENDE MARK.

Ablauflogik

Die Ablauflogik ähnelt der des Step Loop-Beispiels. Es sind prinzipiell dieselben Anweisungen enthalten:

```
PROCESS BEFORE OUTPUT.
  LOOP AT itadir CURSOR i WITH CONTROL tc1.
  ENDLOOP.
*
PROCESS AFTER INPUT.
  MODULE exit_3000 AT EXIT-COMMAND.
  LOOP AT itadir.
    MODULE modify_itab_tc.
  ENDLOOP.
```

Die LOOP-Anweisung im PBO-Abschnitt ist durch den Zusatz `WITH CONTROL tc1` zu ergänzen. Erst dadurch wird die interne Tabelle mit dem Table Control verknüpft. Das Exit-Modul entspricht dem des Step Loop-Dynpros und soll hier nicht aufgelistet werden. Die Anweisung zum Zurückschreiben der geänderten Werte in die interne Tabelle unterscheidet sich von der des Step Loop-Dynpros. Obwohl auch im Table Control-Dynpro in der LOOP-Anweisung der CURSOR-Zusatz benutzt werden muss, kann zum Zeitpunkt PAI die Nummer des aktuellen Datensatzes nicht mehr dem Cursorfeld entnommen werden. Vielmehr ist ein spezielles Feld des Table Controls zu verwenden. Aus diesem Grund wird ein neues Modul notwendig:

```
MODULE modify_itab_tc INPUT.
  MODIFY itadir INDEX tc1-current_line.
ENDMODULE.          " MODIFY_ITAB_TC  INPUT
```

Nach dem Generieren des Dynpros kann die Anwendung erneut getestet werden. Nun sollte auch die Wahl des Table Control im Startbild zu einem korrekten Ergebnis führen, wie in Abbildung 3.150 dargestellt.

Die Bedienung der Anwendung ist sehr einfach gehalten. Vom Startbild aus wählen Sie mit den ersten beiden Drucktasten die Suchmaske. Innerhalb des Startbildes hat die Betätigung einer dieser Schaltflächen eine unmittelbare Wirkung. Die anderen beiden Tasten setzen lediglich den Inhalt einer Variablen. Dieser Wert wird erst ausgewertet, wenn eine der SUCHEN-Drucktasten innerhalb der Suchmasken betätigt wird, um entweder zum Step Loop oder zum Table Control zu verzweigen.

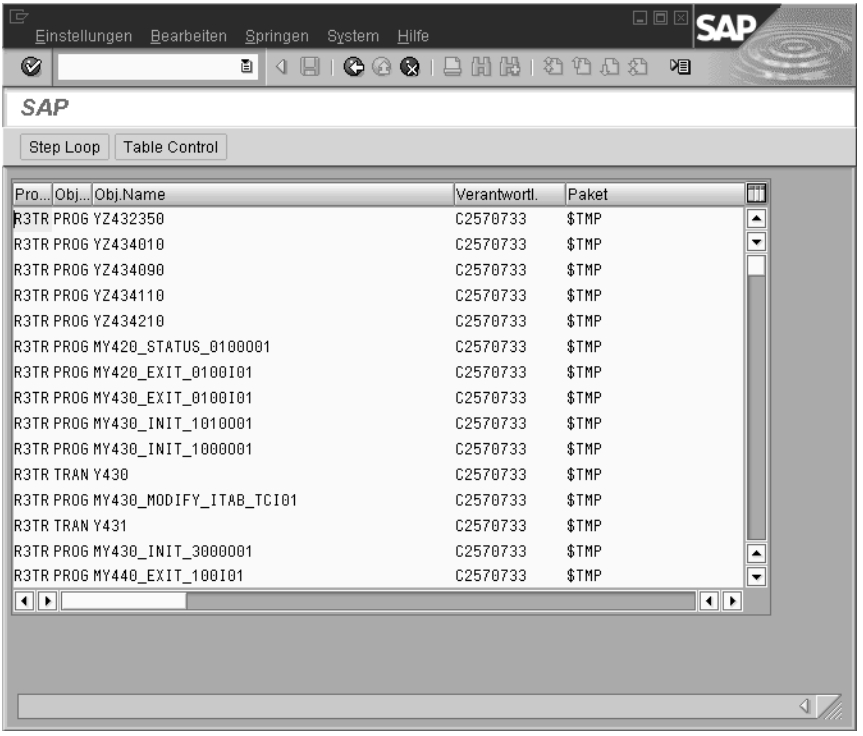

Abbildung 3.150 **© SAP AG**
Table View in der Anwendung

Tab Strips

Tab Strips fassen mehrere Subscreens in Form von Registerkarten zusammen. Es bietet sich in diesem Beispiel an, die beiden Eingabemasken in einem Tab Strip zu verknüpfen. Der Tab Strip wird in einem neuen Dynpro angelegt, das als alternatives Start-Dynpro dienen wird. Dabei wird zunächst die serverseitige Blätterfunktionalität programmiert.

Arbeiten im Layout-Editor

Legen Sie ein normales Dynpro mit der Nummer 200 an. Positionieren Sie den Cursor wieder in die linke obere Ecke und wählen Sie die Menüfunktion BEARBEITEN | ANLEGEN ELEMENT | TABSTRIP. Der Layout-Editor reserviert nun wieder den gesamten freien Platz für das anzulegende Control. Ebenso wie beim Table View markieren Sie durch einen Doppelklick die rechte untere Ecke. Nach diesem Doppelklick werden sie in einem automatisch aktivierten Popup (Abbildung 3.151) aufgefordert, einige wichtige Attribute des Tab Strip zu erfassen.

Abbildung 3.151 **© SAP AG**
Attribute des Tab Strip

Zu diesen Attributen gehört der Name, für den `TS1` zu verwenden ist, da ein derartiges Element bereits deklariert wurde. Einträge in andere Felder sind nicht erforderlich. Mit der Drucktaste ÜBERNEHMEN wird das Popup beendet. Im Layout-Editor erscheint ein bis auf zwei Registertasten leeres Tab Strip (Abbildung 3.152).

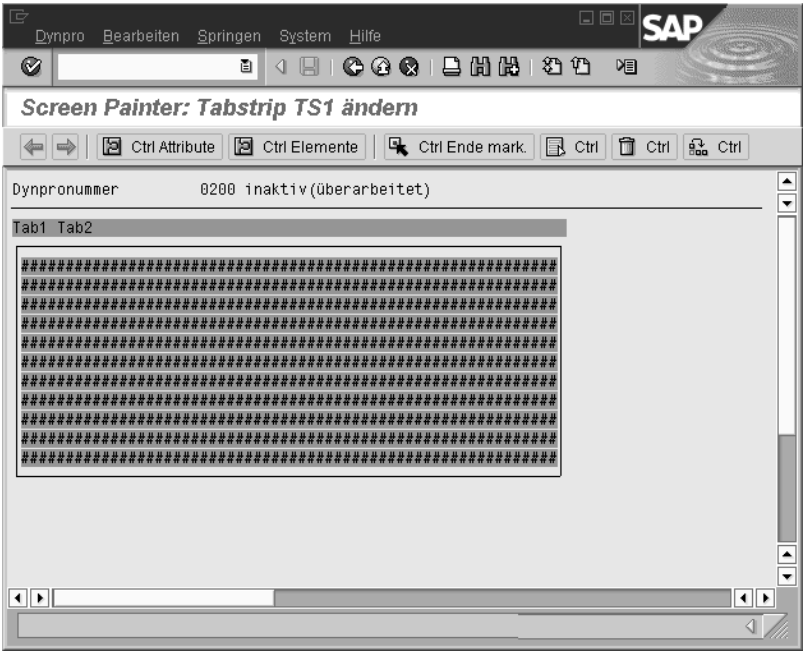

Abbildung 3.152 **© SAP AG**
Neu angelegtes Tab Strip

Das Tab Strip muss nun mit Elementen gefüllt werden. Dieser Vorgang verläuft ähnlich wie beim Table View. Falls Sie sich im Grundbild des Layout-Editors befinden, gelangen Sie mit einem Doppelklick in den markierten Bereich oder die Drucktaste GRAF.ELEMENT in den Bearbeitungsmodus für das Control. Falls Sie ein solches Element neu anlegen, befinden Sie sich unmittelbar nach dem Anlegen bereits in diesem Dynpro. Dort aktivieren Sie mit der Drucktaste CTRL ELEMENTE ein Popup (Abbildung 3.153), in dem Sie das Aussehen des Tab Strip definieren.

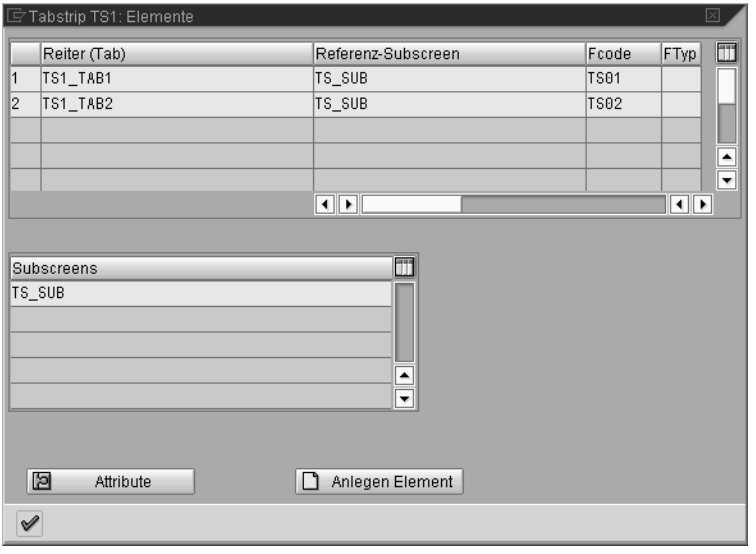

Abbildung 3.153 **© SAP AG**
Definition der Tab Strip-Elemente

In diesem Popup sind bereits zwei Reiter vorgegeben. Ihre Aufgabe besteht darin diesen Reitern einen Funktionscode, einen Text und den Namen eines Subscreen-Bereichs zuzuweisen. Funktionscode und Subscreen-Bereich können Sie direkt im dargestellten Popup eintragen, der Text kann über das Attribute-Popup gepflegt werden. Zum Aufruf dieses Popups setzen Sie den Cursor auf das zu pflegende Element und betätigen die Drucktaste ATTRIBUTE im Tab Strip-Popup. Als Texte können Sie z.B. „Anwender" und „Objekt" benutzen.

Im Teilbereich SUBSCREENS ist der Name des Subscreen-Bereiches (hier `TS_SUB`) einzutragen, der im Tab-Strip-Bereich zur Verfügung steht, um Subscreen-Dynpros darin einblenden zu können. Als Referenz-Subscreen für die einzelnen Register können nur Subscreens benutzt werden, die in dieser Tabelle definiert wurden. Da hier die serverseitige Blätter-Funktion benutzt werden soll, wird in diese beiden Felder ebenfalls der Subscreen-Bereich `TS_SUB` eingetragen. Beim Übernehmen der Daten mittels der gleichnamigen Drucktaste fordert der Layout-Editor zur Festlegung der Größe des Subscreen-Bereichs auf (Abbildung

3.154). Diese Werte sind so zu wählen, dass der freie Platz im Tab Strip komplett ausgenutzt wird. Dieser Subscreen muss vorher nicht explizit angelegt werden, die Auflistung im Popup und das Festlegen der Größe reichen aus. Die Arbeiten in Layout-Editor sind damit abgeschlossen.

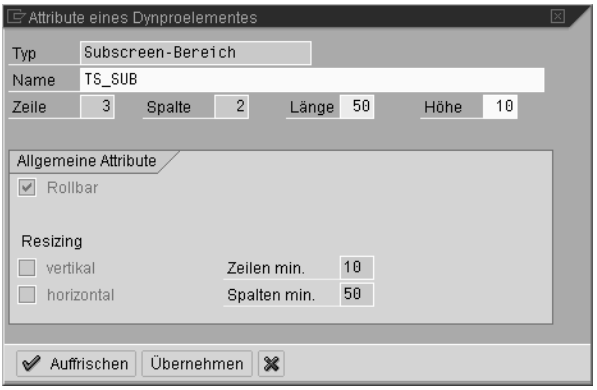

Abbildung 3.154 **© SAP AG**
Größe des Subscreen-Bereiches festlegen

Ablauflogik

Die Ablauflogik ist etwas komplexer, da sowohl Einträge in der eigentlichen Ablauflogik als auch im User-Command-Modul erforderlich sind. Hier zunächst die Ablauflogik.

```
PROCESS BEFORE OUTPUT.
  MODULE status_0200.
  CALL SUBSCREEN ts_sub INCLUDING 'SAPMY430' g_subscreen.
*
PROCESS AFTER INPUT.
  MODULE exit_200 AT EXIT-COMMAND.
  CALL SUBSCREEN ts_sub.
  MODULE user_command_0200.
```

Innerhalb eines Tab Strip steht ein Subscreen-Bereich zur Verfügung, der programmtechnisch ebenso behandelt wird wie ein ganz normaler Subscreen. Aus diesem Grund wird sowohl im Abschnitt PBO als auch PAI jeweils eine `CALL SUBSCREEN`-Anweisung erforderlich, die sich nicht von der des Dynpro 100 unterscheidet. Die Unterschiede treten im User-Command-Modul zu Tage:

```
CASE fcode.
    WHEN 'TS01'.
      g_subscreen  = '1000'.
      ts1-activetab = 'TS01'.
    WHEN 'TS02'.
```

```
      g_subscreen  = '1010'.
      ts1-activetab = 'TS02'.
    WHEN 'STEP'.
      target_screen = '2000'.
    WHEN 'TABC'.
      target_screen = '3000'.
    WHEN 'SEL1'.
      SELECT * FROM tadir
        INTO TABLE itadir
        WHERE devclass = tadir-devclass
          AND author   = tadir-author.
      IF sy-dbcnt > 0.
        LEAVE TO SCREEN target_screen.
      ENDIF.
    WHEN 'SEL2'.
      TRANSLATE tadir-obj_name USING '*%'.
      SELECT * FROM tadir
        INTO TABLE itadir
        WHERE pgmid = tadir-pgmid
          AND object = tadir-object
          AND obj_name LIKE tadir-obj_name.
      IF sy-dbcnt > 0.
        LEAVE TO SCREEN target_screen.
      ENDIF.
  ENDCASE.
ENDMODULE.                          " USER_COMMAND_0200  INPUT
```

Die relevanten Abschnitte, die Auswertung der Funktionscodes `TS01` und `TS02` befinden sich am Anfang des Moduls. Das Umschalten der beiden Eingabemasken erfolgt nun nicht mehr über zwei Drucktasten in der Drucktastenzeile sondern über die Registerkarten des Tab Strip. Die von diesen Registerkarten ausgelösten Funktionscodes werden auf dieselbe Art und Weise ausgewertet wie die durch Drucktasten ausgelösten. Allerdings muss zusätzlich durch eine Zuweisung zum Feld `activetab` des Tab Strip-Controls festgelegt werden, welche der Registerkarten optisch hervorgehoben werden soll.

Da die beiden Drucktasten der Drucktastenzeile nicht mehr benötigt werden, können Sie diese im Modul `status_0200` deaktivieren.

```
MODULE status_0200 OUTPUT.
  REFRESH excode.
  excode = 'SUB1'. APPEND excode.
  excode = 'SUB2'. APPEND excode.
  SET PF-STATUS 'STAT1' EXCLUDING excode.
  start_screen = sy-dynnr.
ENDMODULE.                              " STATUS_0200  OUTPUT
```

Das Exit-Modul unterscheidet sich nicht von dem des Dynpro 100.

Nach Programmierung der Ablauflogik und Aktivierung des neuen Dynpros müssen Sie eine zweite Transaktion (z.B. `Y431`) anlegen, die zwar ebenfalls das Programm `SAPMYZ430` startet, als Start-Dynpro allerdings das eben erzeugte Dynpro 200 benutzt. Abbildung 3.155 zeigt das neue Startbild.

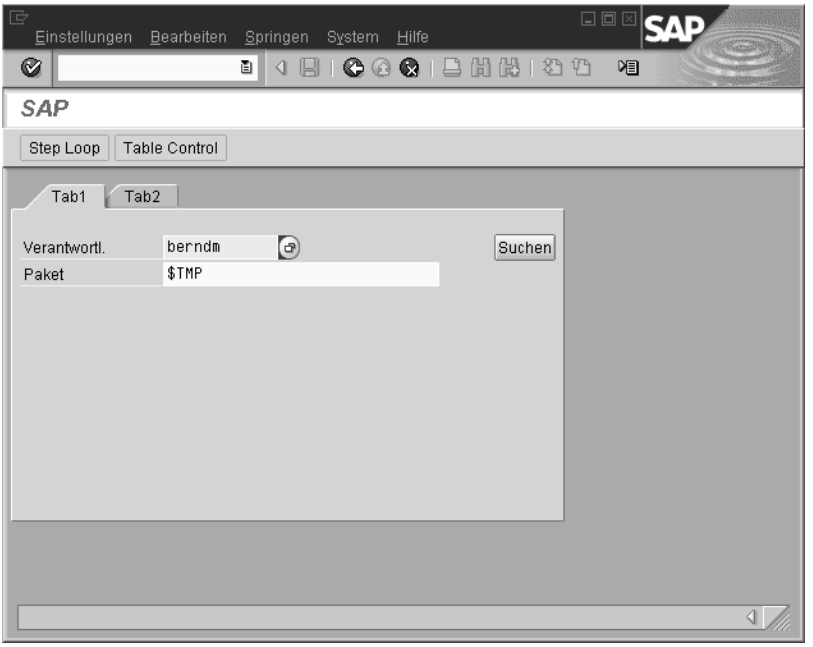

Abbildung 3.155 **© SAP AG**
Startbild mit Tab Strip

Clientseitiges Scrollen

Zum Abschluss soll anhand des aktuellen Beispiels auch noch das clientseitige Scrollen demonstriert werden. Dies erfordert, dass zu jedem Tab Strip ein eigener Subscreen-Bereich angelegt wird. Außerdem muss jeder in den Tab Strip-Bereichen vorhandene Subscreen innerhalb der Ablauflogik des Dynpros angesprochen werden.

Es bietet sich an, das clientseitige Scrollen in einem zusätzlichen Dynpro zu realisieren. Dieses wird zunächst wie das erste Tab Strip-Dynpro angelegt. Die Unterschiede zeigen sich dann bei der Pflege der Elemente des Tab Strip (siehe Abbildung 3.156). Im Bereich SUBSCREENS müssen zwei Subscreens (`T20` und `T21`) angelegt werden. Beide Subscreens müssen natürlich im Deklarationsteil der Anwendung deklariert werden. Jedem Tab Strip wird dann in der Spalte REFERENZ-SUBSCREEN einer der beiden Subscreens zugewiesen. Außerdem werden die beiden Funktionscodes zum Umschalten der Tab Strips mit dem Funktionstyp P für lokale GUI-Funktionen versehen.

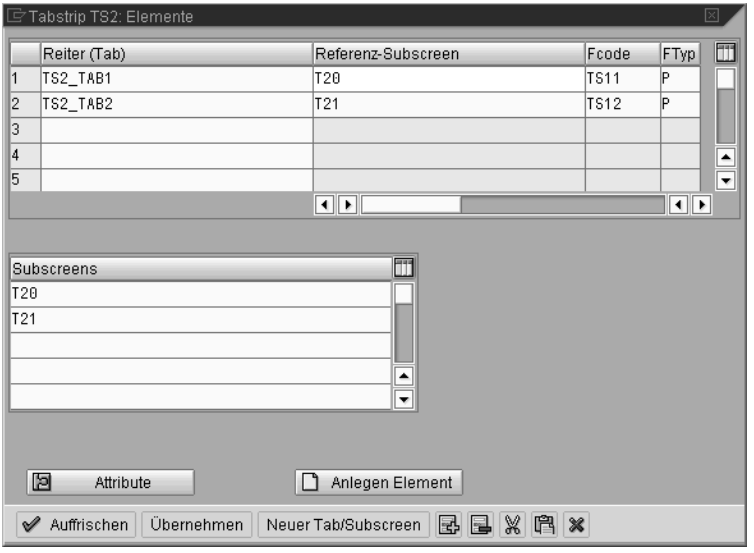

Abbildung 3.156 **© SAP AG**
Elemente des Tab Strip beim Blättern im Client

Innerhalb der Ablauflogik des Dynpros sind natürlich auch einige Änderungen erforderlich. Da zum Blättern keine Server-Zugriffe mehr stattfinden, müssen von vornherein beide Subscreens gefüllt werden. Innerhalb der beiden Dynpro-Zeitpunkte muss daher für jeden Subscreen ein `CALL SUBSCREEN`-Aufruf erfolgen.

```
PROCESS BEFORE OUTPUT.
  MODULE status_0200.
  CALL SUBSCREEN t20 INCLUDING 'SAPMY430' '1000'.
  CALL SUBSCREEN t21 INCLUDING 'SAPMY430' '1010'.

*
PROCESS AFTER INPUT.
  MODULE exit_200 AT EXIT-COMMAND.
  CALL SUBSCREEN t20.
  CALL SUBSCREEN t21.
  MODULE user_command_0200.
```

Übungen

❶ Die zwei Drucktasten, mit denen im Startbild die Darstellungsvariante der internen Tabelle festgelegt wird, haben in den Dynpros 2000 und 3000 keine Funktion. Ändern Sie diese beiden Dynpros so, dass dort direkt in die jeweils andere Darstellungsform gewechselt werden kann.

3.7.4 Spezielle Programmiertechniken in Dialoganwendungen

Ein Modul-Pool stellt die Programmmodule für ein oder mehrere Dynpros zur Verfügung. Außerdem werden im Modul-Pool globale Daten deklariert. Theoretisch könnte der gesamte Quellcode in einer einzigen Datei untergebracht werden. Im Editor lassen sich kürzere Dateien allerdings wesentlich einfacher handhaben. Es ist daher üblich, aber nicht zwingend erforderlich, Modul-Pools in mehrere Include-Dateien aufzusplitten. Man unterscheidet dazu sechs wesentliche Abschnitte, die dann auch durch sechs Include-Dateien bzw. Gruppen von Include-Dateien gebildet werden. Für die verschiedenen Dateigruppen existieren Vorschläge für den Namen der Include-Dateien.

- Datendeklaration;
- PBO-Module für Abarbeitung vor der Dynpro-Anzeige;
- PAI–Module für Abarbeitung nach der Dynpro-Anzeige;
- Include-Dateien für Unterprogramme (Forms);
- Module für Hilfe (F1);
- Module für Eingabemöglichkeit (F4).

Der eigentliche Modul-Pool nimmt nur die verschiedenen `INCLUDE`-Anweisungen auf, anderen Quelltext enthält er üblicherweise nicht. Der für jedes Programm erforderliche Programmkopf wird der Einfachheit halber in eines der Includes, das Top Include, verlagert. Dieses muss daher im Quelltext immer an erster Stelle stehen. Diese Sonderrolle spiegelt sich auch im Namen des Includes wider, der mit `TOP` endet.

Die Namen der Modul-Pools beginnen immer mit `SAPM`. Der nächste Buchstabe soll die Anwendungsgruppe kennzeichnen, für die der Modul-Pool vorgesehen ist, z. B. `F` für Finanzwesen. Ebenso wie bei Reports sind für Kundenentwicklungen an dieser Stelle die beiden Buchstaben `Y` und `Z` vorgesehen, alle anderen sind für die Verwendung bei SAP reserviert. Die weiteren Zeichen dienen dann der eigentlichen Kennzeichnung des Modul-Pools.

Ähnliche Konventionen gelten für die Namen der Include-Dateien. Sie beginnen mit dem um die ersten drei Zeichen gekürzten Namen des Modul-Pools, z. B. mit `MY410`. Es folgt ein Buchstabe, der auf die Art der Elemente im Include hindeutet. Ein `O` steht für Dateien, die PBO-Module enthalten, ein `I` hingegen für Includes mit PAI-Modulen. Dateien mit Unterprogrammen, also Form-Routinen erhalten ein `F`. Die letzten beiden Zeichen ermöglichen dann die Unterscheidung der einzelnen Dateien einer Gruppe. Ab der aktuellen Version 6.x werden die Namen noch weiter verfeinert. Dort wird vor den letzten drei Zeichen ggf. noch der Name des Moduls eingefügt, das als erstes (oder auch einziges) Modul in diesem Include enthalten ist.

Dieser Abschnitt soll nun wichtige Aktionen in Dialoganwendungen und die dazu erforderlichen ABAP-Kommandos beschreiben. Auch wenn diese Kommandos im Zusammenhang mit Dialoganwendungen erläutert werden, bedeutet dies nicht, dass sie ausschließlich dort benutzt werden dürfen. Einige der Anweisungen sind auch in Reports möglich und sinnvoll.

Dynamische Modifikation von Dynpro-Feldern

Im Screen Painter werden für die Dynpro-Felder Attribute vorgegeben. Einige dieser Attribute können zur Laufzeit dynamisch verändert werden, beispielsweise ist es möglich, einigen Feldern die Eingabeberechtigung zu entziehen, sie also in reine Anzeigefelder zu verwandeln oder die Felder ganz vom Dynpro zu löschen. Damit kann ein Dynpro mit relativ wenig Programmieraufwand an die konkreten Berechtigungen des Anwenders oder spezielle Eigenschaften des zu bearbeitenden Datensatzes angepasst werden.

Die Tabelle SCREEN

Die dynamischen Modifikationen finden zum Zeitpunkt PBO statt. Eine spezielle Variante der `LOOP`-Anweisung (der ABAP-Anweisung, nicht der gleichnamigen Anweisung der Ablauflogik) erlaubt die Bearbeitung aller Dynpro-Felder. Bei Ausführung eines Dynpros werden wichtige relevante Daten für jede Feldschablone nicht aber für die Schlüsselfelder in die interne Tabelle `SCREEN` gestellt. Diese Tabelle hat den in Tabelle 3.41 dargestellten Aufbau:

Feldname	Typ	Länge	Inhalt/Wirkung	Editieren sinnvoll
`SCREEN-NAME`	C	30	Feldname	
`SCREEN-GROUP1`	C	3	Bewertung der Modif-Gruppe 1	
`SCREEN-GROUP2`	C	3	Bewertung der Modif-Gruppe 2	
`SCREEN-GROUP3`	C	3	Bewertung der Modif-Gruppe 3	
`SCREEN-GROUP4`	C	3	Bewertung der Modif-Gruppe 4	
`SCREEN-REQUIRED`	C	1	Muss-Eingabe	X
`SCREEN-INPUT`	C	1	Eingabebereitschaft	X
`SCREEN-OUTPUT`	C	1	Feld wird angezeigt	X

Feldname	Typ	Länge	Inhalt/Wirkung	Editieren sinnvoll
SCREEN-INTENSIFIED	C	1	Hervorgehobene Darstellung	X
SCREEN-INVISIBLE	C	1	Passworteingabe	X
SCREEN-LENGTH	X	1	Länge (hexadezimal!)	X
SCREEN-ACTIVE	C	1	1 = Feld erscheint im Dynpro 0 = Feld erscheint nicht im Dynpro, überschreibt alle anderen Attribute	X
SCREEN-DISPLAY_3D	C	1	3D-Anzeige	X
SCREEN-VALUE_HELP	C	1	Werthilfe existiert	
SCREEN-REQUEST	C	1	Eingabe erfolgt	

Tabelle 3.41
Tabelle für Dynpro-Felder

In der LOOP-Schleife können einige Attribute geändert und mit dem Kommando MODIFY in die Tabelle SCREEN zurückgeschrieben werden. Dies ist nicht für alle Attribute sinnvoll, da einige nur Auskunft über spezielle Eigenschaften des Feldes liefern. Ein erstes Beispiel:

```
LOOP AT SCREEN.
  SCREEN-INPUT = '0'.
  MODIFY SCREEN.
ENDLOOP.
```

Mit diesen Anweisungen wird allen Feldern des Dynpros die Eingabebereitschaft entzogen. Als Kennzeichen für ein abgeschaltetes Attribut dient das Zeichen Null „0", eingeschaltet wird ein Attribut mit „1".

Falls nur ausgewählte Dynpro-Felder modifiziert werden sollen, kann in der LOOP-Schleife der Feldname abgefragt werden. Er entspricht dem Namen des Feldes, so wie er im Fullscreen bzw. der Feldliste des Dynpros zu finden ist:

```
LOOP AT SCREEN.
  IF SCREEN-NAME = 'KNA1-KUNNR'.
    SCREEN-INVISIBLE = '1'.
    MODIFY SCREEN.
  ENDIF.
ENDLOOP.
```

Falls Modifikationen für eine größere Anzahl von Dynpro-Feldern notwendig sind, kann auf die Modifikationsgruppen zurückgegriffen werden. Für jedes Dynpro-Feld können im Fullscreen-Editor vier Attribute gepflegt werden, die mit Modifikationsgruppe 1 bis 4 bezeichnet sind. In jedes der vier Felder kann eine beliebige dreistellige Zeichenkette eingetragen werden. Diese Werte können frei gewählt werden, es existieren keine Vorschriften oder Prüfungen. Es handelt sich einfach um vier zusätzliche Bezeichner für das Feld. Später kann dann über die vier Felder `SCREEN-GROUP1` bis `SCREEN-GROUP4` der Inhalt dieser Attribute ermittelt werden. Angenommen, in einem Dynpro existieren einige Felder, die stets zusammen ausgeblendet werden sollen. Im Fullscreen-Editor wird für diese Felder die Modifikationsgruppe 1 auf den Wert „001“ gesetzt. Das Ausblenden erfolgt dann mit den Anweisungen

```
LOOP AT SCREEN.
  IF SCREEN-GROUP1 = '001'.
    SCREEN-ACTIVE = '0'.
    MODIFY SCREEN.
  ENDIF.
ENDLOOP.
```

Die Modifikationsgruppe und der Wert für dieses Attribut sind natürlich beliebig wählbar. Sehr oft werden derartige Modifikationen in Abhängigkeit vom Transaktionscode durchgeführt. Neben einer Transaktion zum Pflegen von Daten existiert oft eine zum reinen Anzeigen. Beispielsweise verfügen alle bisher verwendeten Werkzeuge über einen derartigen Modus. Das Einstiegsbild bietet dann getrennte Menüfunktionen oder Drucktasten für Anzeigen und Ändern an. Der Transaktionscode ist im Systemfeld `SY-TCODE` zu finden.

Obwohl Step Loops aus mehreren Zeilen bestehen, liefert die `LOOP AT SCREEN`-Anweisung für den Step Loop nur einen einzigen Datensatz. Änderungen in diesem Datensatz beziehen sich auf den gesamten Step Loop, also alle im Dynpro sichtbaren Zeilen. Eine Ausnahme bildet der Aufruf von `LOOP AT SCREEN` innerhalb der `LOOP`-Schleife der Ablauflogik. Hier wird ebenfalls nur ein Datensatz für den Step Loop geliefert, der sich aber auf die gerade aktuelle Zeile im Step Loop bezieht. Eventuelle Änderungen werden in diesem speziellen Fall nur für diese Zeile wirksam.

Bearbeiten von Table Views

Table Views sind wesentlich komplexer aufgebaut als herkömmliche Dynpro-Felder oder Step Loops. Sie können daher nicht mit Hilfe des Kommandos `LOOP AT SCREEN` bearbeitet werden, auch wenn für jede Spalte des Table Views eine Zeile in der Tabelle `SCREEN` existiert.

Table Controls erfordern, im Gegensatz zu anderen Dynpro-Elementen, eine Deklaration im Modul-Pool. Neben dem Dynpro-Element existiert daher ein programminternes Datenobjekt mit relativ komplexer Struktur. Es besitzt den gleichen Namen wie das Table Control im Dynpro. Alle Informationen über den Table View (außer den konkreten Daten) sind in diesem Datenobjekt abgelegt.

Wenn das Verhalten des Table Views modifiziert werden soll, so müssen einige Felder in diesem Datenobjekt modifiziert werden. Andere Felder werden vom System benutzt, um Statusinformationen abzulegen. Sie übernehmen damit die Aufgabe von Systemfeldern.

Das Datenobjekt besteht aus einer Feldleiste. Einige einfache Felder in dieser Feldleiste steuern globale Eigenschaften des Table Views. Außerdem ist in der Feldleiste eine interne Tabelle enthalten, die Informationen über die einzelnen Spalten des Table Views liefert. Die Struktur für beide Elemente ist nicht, wie bei der Tabelle `SCREEN`, fest im System verankert, sondern wird durch zwei Typdeklarationen im Type-Pool `CXTAB` beschrieben. Bei der Deklaration eines Table Views werden diese Informationen automatisch ausgewertet.

Feldname	Typ	Inhalt / Wirkung	Editieren sinnvoll
FIXED_COLS	I	Anzahl feststehender Spalten am linken Rand.	X
LINES	I	Anzahl der Zeilen.	
TOP_LINE	I	Startzeile für Darstellung.	X
CURRENT_LINE	I	Enthält in LOOP-Schleifen über den Table View die aktuelle Zeile.	
LEFT_COL	I	Erste angezeigte verschiebbare Spalte.	
LINE_SEL_MODE	I	0 – keine Zeilenselektion. 1 – einfache Zeilenselektion. 2 – mehrfache Zeilenselektion.	X
COL_SEL_MODE	I	0 – keine Spaltenselektion. 1 – einfache Spaltenselektion. 2 – mehrfache Spaltenselektion.	X
LINE_SELECTOR	C(1)	Schaltfläche für Zeilenselektion anzeigen.	X
V_SCROLL	C(1)	X – vertikalen Scrollbalken anzeigen.	(funktionslos)
H_GRID	C(1)	X – Trennlinie zwischen Zeilen.	X
V_GRID	C(1)	X – Trennlinie zwischen Spalten.	X
COLS	CXTAB_COLUMN	Interne Tabelle für Spaltenbeschreibung.	

Tabelle 3.42
Struktur des Datentyps CXTAB_CONTROL für Table Views

Die Tabelle 3.42 zeigt die Struktur des Datentyps CXTAB_CONTROL. Dieser Datentyp beschreibt das gesamte Datenobjekt eines Table Control.

Für die Beschreibung der Struktur der internen Tabelle COLS existiert ebenfalls ein Datentyp. Er trägt den Namen CXTAB_COLUMN und wird in Tabelle 3.43 beschrieben. In diesem Datentyp wird das Feld SCREEN durch die Struktur der gleichnamigen internen Tabelle beschrieben. Es ergibt sich dadurch eine mehrstufige Namensbildung.

Feldname	Typ	Inhalt/Wirkung	Editieren sinnvoll
SCREEN	SCREEN	Attribute einer Spalte.	X
INDEX	I	Anzahl der Zeilen.	X
SELECTED	ICON-OLENG	X – Spalte ist selektiert.	X
VISLENGTH	ICON-OLENG		X
INVISIBLE	C(1)	X – Spalte unsichtbar.	X

Tabelle 3.43
Struktur des Datentyps CXTAB_COLUMN für Spaltenbeschreibungen in Table Views

Die folgenden Ausschnitte aus einem Programm demonstrieren die dynamische Modifikation eines Table Views. Sie beziehen sich auf das Beispiel zu den komplexen Elementen im Dynpro. Sie können das Dynpro 3000 dieses Beispiels mit einem zusätzlichen Modul ergänzen, um die Programmiertechnik zu testen. Es ist zu beachten, dass die interne Tabelle COLS keine Kopfzeile besitzt. Für den Zugriff auf die Tabelle muss daher ein Arbeitsbereich deklariert werden. Dazu muss auf den Datentyp CXTAB_COLUMN zurückgegriffen werden, der wiederum im Typpool CXTAB enthalten ist. Zunächst erfolgt die Deklaration der benötigten Datenobjekte, die im Top Include erfolgen muss:

```
TYPE-POOLS CXTAB.
DATA CL TYPE CXTAB_COLUMN.
```

Später werden in einem PBO-Modul die ersten drei Spalten des Table Views als feststehende und unveränderliche Spalten festgelegt sowie das Feld itadir-devclass ausgeblendet.

```
MODULE init_3000 OUTPUT.
  LOOP AT tc1-cols INTO cl.
    IF cl-index < 4.
      cl-screen-input = ' '.
    ELSE.
      IF ( cl-screen-name = 'ITADIR-DEVCLASS' ).
        cl-invisible = 'X'.
      ENDIF.
```

```
    ENDIF.

    MODIFY tc1-cols FROM cl.
  ENDLOOP.
  tc1-fixed_cols = 3.
ENDMODULE.
```

Verwendung von Icons

Eine andere Möglichkeit, das Aussehen von Dynpro-Feldern vor der Anzeige des Dynpros zu modifizieren, besteht in der Verwendung von Icons. In das Datenfeld, das mit dem Dynpro-Feld verbunden ist, wird ein spezieller Identifikator gestellt. Dieser wird durch das System als Verweis auf ein Icon erkannt und bei der Ausgabe des Feldes durch das Icon ersetzt. Das Dynpro-Feld darf dabei nur Ausgabecharakter haben. Dieser Mechanismus funktioniert übrigens auch im Zusammenhang mit dynamischen Texten für Funktionscodes. Um den Identifikator für ein Icon zu ermitteln, ist ein spezieller Funktionsbaustein aufzurufen. Er erwartet als Eingabeparameter den Namen eines Icons. Der Rückgabewert besteht aus einer speziellen Zeichenkette, die vom Dynpro als Verweis auf ein Icon erkannt wird. Bei Bedarf kann das Icon zusammen mit einem Text erscheinen. Außerdem kann ein Text für eine Quick-Info definiert werden.

Die dynamische Erzeugung von Icons und den dazu notwendigen Funktionsbaustein können Sie sehr einfach testen, in dem Sie ein globales Datenfeld (`p1`) ausreichender Länge (35 Zeichen) deklarieren und dieses Datenfeld im Fullscreen des Dynpros einbinden. Es erhält die Attribute ANZEIGEFELD und MIT ICON. Das automatisch gesetzte Attribut EINGABEFELD wird deaktiviert! Die Länge des Feldes und die im Dynpro vorzusehende Ausgabelänge richten sich unter anderem nach der Art des Icons und den auszugebenden Texten. Das Datenfeld muss neben dem Identifikator des Icons auch noch den eigentlichen Text und den Text für die Quick-Info aufnehmen können.

Das Zuweisen des Icons zum Dynpro-Feld erfolgt üblicherweise in einem PBO-Modul. Sehr oft verfügen Dynpros über ein Modul, in dem sämtliche Modifikationen der Oberfläche stattfinden. Das folgende Listing zeigt die Anwendung des erwähnten Funktionsbausteins. Die Icon-Namen können z. B. der Eingabehilfe des entsprechenden Feldes im Attribute-Popup des Screen Painters entnommen werden.

```
...
CALL FUNCTION 'ICON_CREATE'
     EXPORTING
         NAME                  = 'ICON_GREEN_LIGHT'
         TEXT                  = 'Text zum Icon'
         INFO                  = 'Quick-Info'
         ADD_STDINF            = 'X'
     IMPORTING
         RESULT                = P1
```

```
      EXCEPTIONS
          ICON_NOT_FOUND        = 01
          OUTPUTFIELD_TOO_SHORT = 02.
...
```

Datenveränderung erkennen

In fast jedem Dynpro steht mindestens ein Funktionscode zur Verfügung, mit dem das Dynpro sofort verlassen werden kann. Falls bereits Daten eingegeben wurden, kann es sinnvoll sein, den Anwender vor dem Verlassen des Dynpros über einen möglichen Datenverlust zu informieren. Im Systemfeld `SY-DATAR` (Data Received) wird durch das System ein „X" eingetragen, wenn im Dynpro Werte bearbeitet wurden. Dieses Feld wird vor jedem Durchlauf durch das Dynpro initialisiert. Eventuell muss sein Inhalt in einem programminternen Flag gesichert werden. Dieses Systemfeld ist die einzige Möglichkeit, um in `EXIT`-Modulen eine Datenveränderung zu erkennen. Der Transport von Daten aus Dynpro-Feldern in die programminternen Felder findet erst nach dem `EXIT`-Modul statt. Im `EXIT`-Modul ist daher keine Datensicherung möglich! Sollte dies erforderlich sein, darf im `EXIT`-Modul die Dynpro-Bearbeitung nicht beendet werden! Das in Kapitel 7 vorgestellte Anwendungsbeispiel demonstriert ausführlich, wie in solchen Fällen vorzugehen ist.

Eine weitere Variante, Datenänderungen zu vermerken, besteht in der Verwendung eines globalen Flags, das in einem speziellen Modul gesetzt wird. Der Zusatz `ON REQUEST` in einer `MODUL`-Anweisung sorgt dafür, dass dieses Modul nur aufgerufen wird, wenn Daten verändert wurden. In der Ablauflogik verbinden Sie die Felder, deren Inhalt überwacht werden soll, mit Hilfe der `FIELD`-Anweisungen mit einem Modul, in dem Sie ein globales Flag setzen:

```
* Ablauflogik
PROCESS AFTER INPUT.
...
  FIELD fieldname     "eine Anweisung für jedes Feld
    MODULE DATA_CHANGED ON REQUEST.
...

* Modul-Pool
MODULE DATA_CHANGED.
  FLG_DATA_CHANGED = 'X'.
ENDMODULE.
```

Das Flag ist zurückzusetzen, wenn das Dynpro mit neuen Daten gefüllt wird. Es ist allerdings, im Gegensatz zu `SY-DATAR`, nicht wirksam, wenn der Abbruch sofort bei der erstmaligen Abarbeitung des Dynpros erfolgt. Für eine sichere Erkennung von Datenänderungen müssen also beide Varianten benutzt werden.

Datenfeld an Cursorposition auswerten

In den LOOP-Anweisungen findet nur der Datentransfer statt. Wenn interne Tabellen bearbeitet werden sollen, ist mitunter auch die Anwendung zusätzlicher Funktionscodes zur Bearbeitung eines ausgewählten Datensatzes aus dem Dynpro möglich. Dabei kann es sich beispielsweise um das Einfügen neuer oder das Löschen vorhandener Sätze handeln. Aus Sicht des Anwenders sind diese Aktionen besonders einfach, wenn der zu bearbeitende Datensatz im Dynpro durch die Position des Cursors identifiziert wird und keinerlei zusätzliche Markierungsfelder o. Ä. erforderlich werden. Die Programmierung derartiger Funktionen erfordert es, innerhalb der PAI-Module, aber außerhalb der LOOP-Schleife zu ermitteln, in welchem Loop-Block der Cursor steht und aus dieser Angabe den Index des Datensatzes zu ermitteln. Die Nummer des Loop-Blocks kann sehr einfach mit der Anweisung

```
GET CURSOR LINE line.
```

ermittelt werden. Diese Anweisung stellt die Nummer des aktuellen Loop-Blocks in das als Parameter angegebene Feld. Dabei ist zu beachten, dass ein Loop-Block auf dem Dynpro aus mehreren Zeilen bestehen kann. Für alle Zeilen eines Blocks ermittelt die GET-Anweisung denselben Wert!

Die so ermittelte Nummer des Loop-Blocks ergibt zusammen mit der Nummer des ersten im Loop angezeigten Datensatzes (itab_offset im Beispiel 1 bzw. Wert des Loop-Cursors i im Beispiel 2) und vermindert um den Wert 1 die endgültige Nummer des Datensatzes.

Die Anweisung GET CURSOR verfügt über mehrere Optionen, die auch dann sinnvolle Ergebnisse liefern, wenn der Bildschirmcursor nicht in einem Step Loop-Bereich sondern in einem einfachen Dynpro-Feld steht. Die Anweisung liefert daher zusätzlich zum Rückgabeparameter in der Systemvariablen SY-SUBRC die Information, ob sich der Bildschirmcursor in einem Loop-Bereich befindet (SY-SUBRC = 0) oder nicht (SY-SUBRC = 4).

Die Anwendung der Anweisung ist recht einfach. Der nachfolgende Ausschnitt aus einem Programm zeigt die Verwendung. Im zugehörigen Status wurde die Funktionstaste F2 mit dem Funktionscode FT02 belegt. Dieser Funktionscode soll vor dem Datensatz, in dem der Bildschirmcursor steht, einen neuen Satz einfügen und eines der Felder des Datensatzes mit einem Wert füllen (Zeilennummer + 100). Das Feld cl enthält die mit HIDE verborgene Zeilennummer.

```
MODULE FCODE INPUT.
  CASE FCODE.
    WHEN 'FT02'.
      GET CURSOR LINE CL.
      IF SY-SUBRC = 0.
        CL = CL + I - 1.
        ITAB-Z = CL + 100.
```

```
        INSERT ITAB INDEX CL.
      ENDIF.
  ENDCASE.
ENDMODULE.
```

Listboxen füllen

Ab Version 4.x stehen für Dynpro-Eingabefelder auch Listboxen zur Verfügung, in denen Werte zur Auswahl angeboten werden. Dieser Wertevorrat kann automatisch durch das System oder manuell im Programm gesetzt werden. Voraussetzung ist, das entsprechende Eingabefeld mit dem Attribut LISTBOX zu versehen (siehe Abbildung 3.157).

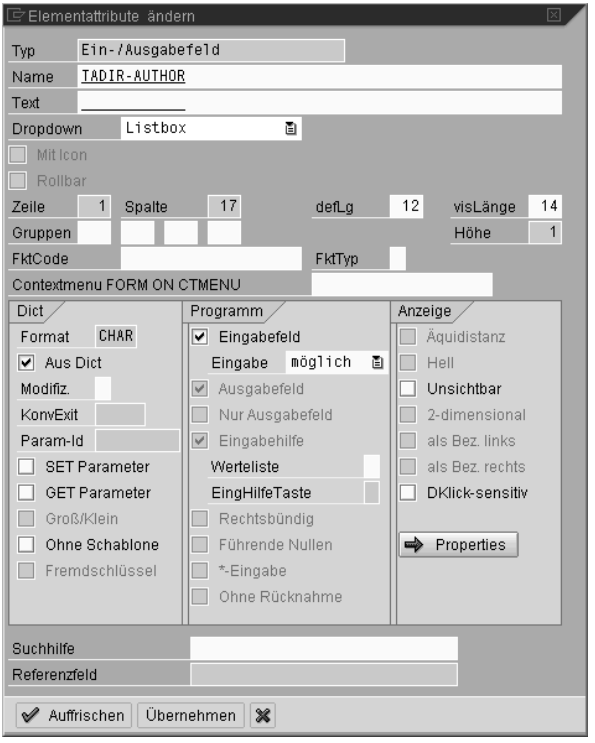

Abbildung 3.157 **© SAP AG**
Eingabefeld mit Listbox ergänzen

Zum programmgesteuerten Definieren des Wertevorrats wird der Funktionsbaustein VRM_SET_VALUE benutzt. Dieser Baustein erwartet als Eingabeparameter eine interne Tabelle des Typs VRM_VALUE. Die Definition dieses Datentyps ist im Type Pool VRM enthalten. Die erste Aufgabe besteht daher darin, den Type Pool einzubinden und die interne Tabelle zu deklarieren:

```
TYPE-POOLS vrm.
DATA possible_values TYPE vrm_value OCCURS 0 WITH HEADER LINE.
```

Zum Zeitpunkt PBO wird diese Tabelle dann gefüllt und an den Funktionsbaustein übergeben:

```
possible_values-key = 'KEY1'. possible_values-text = '1st item'.
APPEND possible_values.
possible_values-key = 'KEY2'. possible_values-text = '2nd item'.
APPEND possible_values.
possible_values-key = 'KEY3'. possible_values-text = '3rd item'.
APPEND possible_values.

CALL FUNCTION 'VRM_SET_VALUES'
  EXPORTING id     = 'TADIR-AUTHOR'
            values = possible_values[].
```

Der Funktionsbaustein erwartet als Eingabeparameter den Namen des Feldes, für den die Listbox gefüllt werden soll und die Tabelle mit den zu setzenden Werten. Da die Tabelle in diesem Beispiel über eine Kopfzeile verfügt, muss bei der Übergabe des Parameters durch das Anfügen der eckigen Klammern an den Tabellennamen verdeutlicht werden, das die Tabelle und nicht die Kopfzeile übergeben wird.

Programmverzweigungen aus Dynpros

Von einem Dynpro aus können verschiedene andere Elemente angesprungen werden. Diese Verzweigung findet in den meisten Fällen im PAI-Abschnitt statt. Aber auch im Abschnitt `PROCESS ON VALUE-REQUEST` sind Aufrufe anderer Programme oder Programmteile möglich und sinnvoll. Es bestehen folgende Verzweigungsmöglichkeiten:

- Fortsetzen mit anderem Dynpro;
- Ausführung eines Reports;
- Start einer anderen Transaktion;
- Beenden der Anwendung.

Zum Aufruf von Elementen existieren drei Kommandos mit einigen Varianten, `LEAVE`, `SUBMIT` und `CALL`. Sie unterscheiden sich unter anderem hinsichtlich der Steuerung des Programmflusses. Während mit `LEAVE` eine Anwendung sequenziell fortgesetzt wird, entspricht `CALL` eher einem Unterprogrammaufruf. Die Anweisung `SUBMIT` hingegen kann mit verschiedenen Zusätzen beide Verzweigungsarten realisieren.

Viele der in diesem Abschnitt besprochenen Kommandos können auch in Reports benutzt werden. Allerdings erfolgt dies in der Praxis wesentlich seltener als in den hier besprochenen Dialoganwendungen.

Verzweigung mit LEAVE

Mit LEAVE werden logisch aufeinander folgende Elemente, z.B. Dynpros, aufgerufen. Das aufgerufene Element erhält die vollständige Kontrolle über die weitere Ausführung des Programms. Wird die Abarbeitung eines solchen Elements beendet, ohne dass ein Folgeelement aufgerufen wird, entspricht dies der Beendigung der Anwendung.

Verschiedene Varianten der Anweisung LEAVE gestatten den Aufruf unterschiedlicher Elemente. Zum Aufruf eines Dynpros sind zumeist zwei Kommandos erforderlich. Zunächst ist mit

```
SET SCREEN screen.
```

die Nummer des aufzurufenden Dynpros zu setzen. Dieses Kommando kann nur entfallen, wenn die Nummer des neuen Dynpros im aktuellen Dynpro bereits als Folge-Dynpro gesetzt wurde. Der Sprung zum neuen Dynpro erfolgt mit

```
LEAVE SCREEN.
```

Ab Release 3.0 können beide Anweisungen in einer vereinigt werden. Diese lautet dann

```
LEAVE TO SCREEN screen.
```

Diese Art der Verzweigung ist endgültig. Sie ist nicht mit einem Unterprogrammaufruf zu verwechseln, nach Beendigung des aufgerufenen Dynpros erfolgt also kein automatischer Rücksprung zum rufenden Dynpro. Eine Sonderfunktion hat der Sprung zum Dynpro 0. Er beendet die Anwendung. Dieselbe Funktion kann auch mit

```
LEAVE PROGRAM.
```

erreicht werden. Der Sprung zu einer anderen Transaktion erfolgt mit

```
LEAVE TO TRANSACTION transactioncode.
```

Der Transaktionscode ist in Großbuchstaben anzugeben und in einfache Anführungsstriche einzuschließen. Diese Anweisung kann mit einem optionalen Zusatz

```
... AND SKIP FIRST SCREEN.
```

versehen werden. Dieser Zusatz bewirkt, dass das Start-Dynpro der gerufenen Transaktion nicht angezeigt wird. Die Ablauflogik des ersten Dynpros wird allerdings korrekt abgearbeitet. Die Felder dieses Dynpros müssen daher automatisch gefüllt werden, entweder durch GET/SET-Parameter in Verbindung mit entsprechenden Attributen im Dynpro oder durch Anweisungen in den PBO-Modulen. Sofern alle Muss-Eingabefelder korrekt gefüllt wurden, erhält der Anwender keine Möglichkeit zur Dateneingabe. Die Abarbeitung wird sofort mit

den PAI-Modulen fortgesetzt. Der aktuelle Funktionscode ist der für Datenfreigabe (Standard: Leerzeichen). Diese Form der Anweisung und das Überspringen der Dateneingabe im ersten Dynpro sind nur sinnvoll, wenn die gerufene Transaktion einen speziellen Aufbau besitzt. Darauf wird am Ende dieses Abschnitts genauer eingegangen.

Verzweigungen mit SUBMIT

Neben anderen Dynpros und Transaktionen können auch Reports aufgerufen werden. Dies erfolgt nicht mit der Anweisung `LEAVE`, sondern mit

```
SUBMIT report.
```

Zu diesem Kommando sind rund 20 verschiedene Zusätze verfügbar, mit denen vom Format der Ausgabeliste bis hin zum Drucken oder Archivieren Vorgaben an den Report übergeben werden können. Die meisten dieser Zusätze sind nur in Spezialfällen interessant. Wirklich bedeutsam sind an dieser Stelle zunächst nur die im Folgenden aufgeführten fünf Zusätze:

Der Zusatz

```
... AND RETURN.
```

bewirkt nach Ausführung des Reports die Rückkehr zum rufenden Programm. Diese wird an der Stelle nach der `SUBMIT`-Anweisung fortgesetzt. Der Report wird sofort gestartet, ohne dass vorher ein eventuell vorhandener Selektionsbildschirm abgearbeitet wird. Der Anwender hat somit keine Möglichkeit, die vom Report gesuchte Datenmenge einzuschränken. Der Programmierer kann aber dafür sorgen, dass dem Report trotzdem Parameter und Selektionen übergeben werden. Die einfachste Möglichkeit besteht im Benennen einer Variante mit

```
... USING SELECTION-SET variant
```

Des Weiteren ist es möglich, mit

```
... WITH parameter ...
```

oder

```
... WITH SELECTION-TABLE selection
```

Vorbelegungen für Parameter und Selektionen an den Report zu übergeben. Falls der Anwender den Selektionsbildschirm doch bearbeiten soll, kann dies durch den Zusatz

```
... VIA SELECTION-SCREEN
```

erreicht werden. Er bewirkt die Abarbeitung des Selektionsbildschirms. Falls von der rufenden Anwendung aus mit `WITH` Vorgabewerte für Parameter und Selektionen gesetzt wurden, kann der Anwender die übergebenen Vorschlagswerte im Selektionsbildschirm ändern, bevor er den Report ausführt.

Das Kommando CALL

Diese Anweisung führt die gerufene Anwendung oder das gerufene Element als Subprozess aus, nach Beendigung der gerufenen Anwendung wird die aufrufende Anwendung an der Unterbrechungsstelle fortgesetzt. Besonders verbreitet ist dieses Kommando, um in Dynpros Popups aufzurufen. Während der bisherigen Arbeit haben Sie bereits viele Popups, vor allem zur Eingabe einzelner Werte oder zur Bestätigung spezieller Aktivitäten kennen gelernt. All diese Popups werden mit einer Variante des Kommandos `CALL` aufgerufen.

Zum Aufruf von Dynpros dient die Anweisung

```
CALL SCREEN screen.
```

Im gerufenen Dynpro führt die Anweisung `LEAVE TO SCREEN 0.` jetzt nicht zur Beendigung der gesamten Anwendung, sondern nur zur Rückkehr zur Aufrufstelle. In der Grundform des Kommandos nimmt das neue Dynpro die gesamte Arbeitsfläche ein, ersetzt das alte also vollständig. Das gerufene Dynpro sollte ein normales Dynpro sein.

Ein Popup ist ein Dynpro, in dessen Attributen der Dynpro-Typ MODALES DIALOGFENSTER anstelle von NORMAL markiert ist. Der Aufruf erfordert die Angabe der Position, an der das Popup erscheinen soll. Diese Position wird relativ zum aktuellen Dynpro angegeben:

```
CALL SCREEN screen STARTING AT line column.
```

Das System sorgt bei dieser Form des Aufrufs dafür, dass die Größe des Popups entsprechend der in diesem Dynpro tatsächlich belegten Fläche erfolgt. Sehr lange Popups, deren Länge den auf dem Bildschirm zur Verfügung stehenden Raum überschreiten würden, werden automatisch auf die Bildschirmhöhe oder -breite begrenzt und erhalten einen Rollbalken. Dies kann aber nur eine Notlösung sein. Daher können Sie bei Bedarf die Endposition auch explizit angeben, was allerdings nur sinnvoll ist, wenn auch eine Startposition festgelegt wurde.

```
CALL SCREEN screen STARTING AT line column
                  ENDING AT line column.
```

Popups werden durch das System etwas anders behandelt als einfache Dynpros. Die Drucktastenzeile wird an den unteren Rand des Popups verlegt. Menü, Symbolleiste und das OK-Feld sind nicht verfügbar. Lediglich die Belegung aller Funktionstasten kann mit der rechten Maustaste eingeblendet werden. In Dynpros, die als Popup prozessiert werden sollen, sind daher alle wichtigen Funktionen auf Funktionstasten zu legen.

Mit `CALL SCREEN` können nur Dynpros der eigenen Anwendung aufgerufen werden. Dies würde zu einem unvertretbar hohen Programmieraufwand führen, wenn in mehreren Anwendungen Dynpros mit allgemeinem Charakter benötigt werden. Derartige allgemeine Dynpros werden daher als so genannte

Dialogbausteine programmiert, die aus allen Anwendungen heraus aufgerufen werden können. Details zu den Dialogbausteinen folgen in einem anderen Abschnitt. Hier soll nur der Aufruf dieser Bausteine demonstriert werden:

```
CALL DIALOG dialog.
```

Da diese Dialogbausteine nicht zur aktuellen Anwendung gehören, können sie auch nicht auf Datenfelder der laufenden Anwendung zugreifen. Falls Daten an den Dialogbaustein übergeben oder von diesem zurückgeliefert werden sollen, müssen die beteiligten Felder bei Aufruf angegeben werden. Dies erfolgt getrennt nach Export- und Import-Parametern. Export-Parameter sind die Elemente, die an den Baustein übergeben werden:

```
... EXPORTING field_1 FROM param_1 ... field_n FROM param_n.
```

Die Dialogbausteine besitzen eine definierte Schnittstelle. In dieser Schnittstelle werden alle Parameter aufgeführt, denen Werte zugewiesen oder von denen Werte gelesen werden können. Mit dem Zusatz `FROM` wird einem dieser Parameter der Schnittstelle (`field...`) ein Wert aus dem aktuellen Programm (`param...`) zugewiesen. Sollten die Namen der Felder im aufrufenden Programm und im Dialogbaustein identisch sein, kann der Zusatz `FROM` entfallen. Es muss dann nur noch der Name des zu exportierenden Feldes angegeben werden. Das Verfahren zur Rückgabe von Werten ist ähnlich. Mit dem Zusatz

```
... IMPORTING field_1 TO param_1 ... field_n TO param_n.
```

können Daten über die Parameter des Dialogbausteins an Elemente des rufenden Programms zurückgeliefert werden. Bezüglich identischer Namen gilt dieselbe Erleichterung wie bei `EXPORTING`.

Ein Dialogbaustein besteht nicht zwangsläufig aus einem einzigen Dynpro. Innerhalb eines Dialogbausteins kann zu weiteren Dynpros verzweigt werden. Daher ist auch für die Anweisung `CALL DIALOG` der Zusatz `AND SKIP FIRST SCREEN` zugelassen.

Auch der Sprung zu anderen Anwendungen ist mit `CALL` möglich. Mit der Anweisung

```
CALL TRANSACTION transactioncode.
```

wird die angegebene Transaktion als Subprozess – sozusagen als Unterprogramm – aufgerufen. Ebenso wie bei `LEAVE TO TRANSACTION` kann durch den Zusatz

```
... AND SKIP FIRST SCREEN
```

das erste Dynpro übersprungen werden, falls alle Muss-Eingabefelder mit gültigen Werten belegt sind. In der gerufenen Transaktion führen alle Kommandos, die diese Transaktion auf normale Weise beenden (z.B. `LEAVE TRANSACTION`, `LEAVE TO SCREEN 0`), zur aufrufenden Anwendung zurück.

Eingebettete List-Verarbeitung

Bisher wurde noch nicht darauf hingewiesen, dass Dialog- und List-Verarbeitung miteinander kombiniert werden können. Es ist möglich, in Dialoganwendungen Elemente der List-Verarbeitung einzubetten bzw. in Reports Dynpros aufzurufen. In der Praxis wird relativ häufig von dieser Möglichkeit Gebrauch gemacht. Beispielsweise schaltet das Kommando

```
LEAVE TO LIST-PROCESSING.
```

in einer Dialoganwendung vorrübergehend die List-Verarbeitung ein. Da diese Problematik recht komplex werden kann, ist ihr ein eigener Abschnitt (Verknüpfung von List- und Dialogverarbeitung) gewidmet.

3.7.5 Diskussion

Die in diesem Abschnitt vorgestellten Kommandos und die beschriebenen Zusammenhänge erläutern die Funktionsweise eines Dynpros. Dieses Wissen allein reicht aber nicht aus, um brauchbare Anwendungen zu erstellen. Das Dynpro-Konzept und einige der erwähnten Kommandos erfordern beim Entwurf einer Anwendung die Berücksichtigung eines Programmschemas und einiger ungeschriebener (SAP-interner) Empfehlungen. Sehr viele Anwendungen im SAP-System entsprechen einer relativ einfachen Grundform. Mitunter wird diese Grundstruktur zwar recht stark modifiziert, ohne aber vollständig mit ihr zu brechen. Von diesen Grundsätzen kann zwar abgewichen werden, dann bezahlen Sie die Freizügigkeit bei der Gestaltung der Anwendung aber mit einer komplizierten Programmstruktur und einer aufwändigen Wartung. Der ABAP-Einsteiger sollte sich zunächst an den folgenden Regeln orientieren. Die Grundgedanken bei der Anwendungsentwicklung sind folgende:

- Eine Dialoganwendung (und damit eine Transaktion) dient zur Bearbeitung eines Objekts bzw. der Erfüllung einer nicht weiter zerlegbaren Aufgabe. Das bedeutet natürlich nicht, dass nur eine Tabelle bearbeitet wird. Wenn allerdings eine Anwendung (beispielsweise `SE38`) zur Bearbeitung des Quelltexts eines Reports dient, dann werden direkt in dieser Anwendung keine anderen Elemente wie Textelemente oder Nachrichten bearbeitet. Dazu existieren eigene Transaktionen, die aber durchaus von der aktuellen Transaktion aus aufgerufen werden können.
- Eine Transaktion ist völlig eigenständig. Es darf keine programmtechnischen Vorbedingungen für den Aufruf einer Transaktion geben.
- Transaktionen sollten trotz ihrer Eigenständigkeit als Modul des Gesamtsystems gesehen werden. Insbesondere sollten sie so gestaltet werden, dass sie auch von anderen Anwendungen aus aufgerufen werden können. (z. B. Pflege der Textelemente über eigene Transaktion oder aus der `SE38` heraus). Da beim Aufruf einer Transaktion aus Programmen (`CALL TRANSACTION`) her-

aus keine Parameter mitgegeben werden können, existieren für viele Dialoganwendungen mehrere Transaktionscodes, die beispielsweise darüber entscheiden, ob in einem Programm ein Objekt angelegt, bearbeitet, gelöscht oder angezeigt wird. Dazu wird innerhalb einer Anwendung der Transaktionscode ausgewertet. Außerdem kann durch Auswertung des Systemfelds `SY-CALLD` ermittelt werden, ob eine Anwendung direkt oder mittels `CALL TRANSACTION` aufgerufen wurde.

- Eine Dialoganwendung besteht häufig aus zwei Teilen, jeder Teil aus mindestens einem Dynpro. Im ersten Teil (und damit im ersten Dynpro, dem Start-Dynpro) wird durch Eingabe der Werte für die Schlüsselfelder das konkrete zu bearbeitende Objekt ermittelt. In den nachfolgenden Dynpros wird das gewählte Objekt dann bearbeitet. Die Felder des ersten Dynpros werden üblicherweise mit `GET/SET`-Parametern verbunden. Beim Aufruf der Anwendung von einer anderen Transaktion aus werden die globalen Parameter belegt und das erste Dynpro der gerufenen Transaktion mit ... `AND SKIP FIRST SCREEN` unterdrückt. Damit kann ohne Eingabe der Schlüsselwerte durch den Anwender ein vorgegebenes Objekt bearbeitet werden.
- Bei der Planung einer Anwendung steht zunächst das Was und Wann im Vordergrund, nicht das Wie. Die reine Funktionalität einer Anwendung lässt sich relativ einfach durch Ergänzung der Ablauflogik der Dynpros erweitern. Änderungen der Programmstruktur hingegen sind oft kompliziert und führen zu unerwünschten Nebenwirkungen.
- Bei der Implementierung der Funktionalität sollten Sie die Eigenschaften der beiden Teile der Ablauflogik berücksichtigen. Im PBO-Teil erfolgen Initialisierungen bezüglich der Daten und des Dynpros, im PAI-Teil hingegen findet die eigentliche Prüfung und Auswertung der Daten statt.

3.8 Verknüpfung von List- und Dialoverarbeitung

Bisher wurde vorausgesetzt, dass zwischen Reports und dialogorientierten Anwendungen eine strenge Trennung besteht. Diese strikte Unterscheidung kann in ausgewählten Fällen überwunden werden. Es ist sowohl möglich, Listen in Dialoganwendungen einzubetten als auch in Reports Dynpros zu benutzen.

3.8.1 Listen in Dynpros

Die Gestaltung von Dynpros kann nur in relativ engen Grenzen dynamisch beeinflusst werden. Für einige Anwendungsfälle sind Dynpros daher nicht flexibel genug, um Werte darzustellen und zu bearbeiten. In diesen Fällen wird oft auf die interaktive List-Verarbeitung zurückgegriffen, die mittels der im Folgenden beschriebenen Anweisungen auch direkt in die Dynpro-Verarbeitung ein-

gebettet werden kann. Ein Beispiel für die Einbettung von Listen sind die Popups der Suchhilfen, in denen Listen dargestellt werden, aus denen sich der Anwender einen Eintrag aussuchen kann. Auch der bereits von Ihnen benutzte Object Browser, genauer gesagt dessen Objektliste, beruht auf der List-Verarbeitung. Innerhalb der Dialogverarbeitung kann mit der Anweisung

```
LEAVE TO LIST-PROCESSING.
```

vorübergehend die listenorientierte Arbeitsweise eingeschaltet werden. Alle Kommandos, die nur in der Listenverarbeitung erlaubt sind (z.B. `WRITE`) können jetzt ausgeführt werden. Die Darstellung der Liste auf dem Bildschirm erfolgt, wenn der PAI-Teil der Ablauflogik des aktuellen Dynpros komplett abgearbeitet oder explizit mit `LEAVE SCREEN` beendet wird. Ohne die eingebettete List-Verarbeitung würde an dieser Stelle meist der Sprung zum Folge-Dynpro erfolgen. Existiert aber eine eingeschobene Liste, wird zunächst diese Liste angezeigt und erst nach Beendigung der List-Verarbeitung das Folge-Dynpro aufgerufen. Beim Folge-Dynpro handelt es sich entweder um das mit dem Attribut FOLGE-DYNPRO oder mit der Anweisung `SET SCREEN` festgelegte Dynpro. Bei der praktischen Anwendung der eingebetteten Listen wird in den meisten Fällen immer wieder das Dynpro aufgerufen, in dessen PAI-Teil die Liste erzeugt wurde.

Die eingebettete Liste wird vom System ebenso wie eine mit einem eigenständigen Report erzeugte Liste ausgeführt. Das bedeutet unter anderem, dass einige Funktionscodes, z.B. die für das Beenden der List-Verarbeitung, automatisch durch das System ausgewertet werden. Innerhalb der List-Verarbeitung müssen also die für sie notwendigen Funktionscodes zur Verfügung stehen. Unbedingt erforderlich ist ein Funktionscode zum Beenden der List-Verarbeitung. Beim Aufruf der eingebetteten List-Verarbeitung wird allerdings nicht automatisch der Standardstatus für Listen aufgerufen, der diese Funktionscodes zur Verfügung stellen würde, sondern es ist weiterhin der Status des rufenden Dynpros aktiv. Ohne zusätzliche Programmierarbeit besteht daher die Gefahr, dass eine eingebettete Liste nicht beendet werden kann.

Um der Liste die erforderlichen Funktionscodes zur Verfügung zu stellen, existieren einige Möglichkeiten. Am einfachsten ist es, die notwendigen Funktionscodes im Status des Dynpros zu definieren. Wenn Sie beispielsweise die Funktionstaste F3 bzw. das entsprechende Symbol mit dem Funktionscode `BACK` belegen, reicht dies aus, um die Liste beenden zu können. Dies funktioniert aber nur, weil `BACK` zu den vom System automatisch ausgewerteten Funktionscodes gehört.

Die sicherste Variante besteht im Setzen des vordefinierten Status der List-Verarbeitung. Dies können Sie sehr einfach mit der Anweisung

```
SET PF-STATUS ' '.
```

erreichen. Da der angegebene Status nicht existiert, verwendet das System den zur Listenverarbeitung gehörenden Standardstatus. Die Standardfunktions-

codes (z.B. Beenden, Drucken, Suchen) fängt das System ab und verarbeitet sie. Da auf diese Weise die von Reports bekannte Funktionalität einer Liste zur Verfügung steht, reicht die Betätigung von F3, F12 oder F15 bzw. der entsprechenden Menüfunktionen aus, um zur Dialogverarbeitung zurückzukehren.

In vielen Fällen kann dieser Status allerdings nicht verwendet werden, da eine völlig andere Funktionalität der eingebetteten Liste im Vergleich zum normalen Reporting realisiert werden soll. In diesen Fällen wird es unter Umständen erforderlich, die Funktionscodes selbst auszuwerten und mit dem Kommando

```
LEAVE LIST-PROCESSING.
```

zur Dialogverarbeitung zurückzukehren.

Beim Beenden der List-Verarbeitung wird der PBO-Teil des Dynpros angesprungen, welches die Verarbeitung der Liste aufrief. Von dieser Vorgabe können Sie mit der Anweisung

```
LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN screen.
```

abweichen. Nach Beendigung der List-Verarbeitung springt das Programm zum angegebenen Dynpro.

Die individuelle Auswertung von Funktionscodes erfolgt nach dem Kommando `LEAVE TO LIST-PROCESSING` nicht mehr in der von Dynpros gewohnten Weise über die Ablauflogik eines Dynpros, sondern über die Zeitpunktanweisungen für Reports oder intern im System. Die auf die List-Verarbeitung bezogenen Zeitpunktanweisungen müssen direkt im Modul-Pool notiert werden.

Das folgende Beispiel (`SAPMY440`) veranschaulicht die in ein Dynpro eingebettete List-Verarbeitung. Sie finden im weiteren Verlauf des Textes nur den relevanten Ausschnitt des Modul-Pools. Die umgebende Anwendung erzeugen Sie selbst anhand der bisherigen Beispiele. Im Status des Dynpros sollen drei Drucktasten verfügbar sein, mit denen die Funktionscodes `FNC1`, `FNC2` und `FNC3` ausgelöst werden können. Diese Funktionscodes sollten dazu auf die Funktionstasten F6 bis F8 gelegt werden. Damit die Anwendung beendet werden kann, muss natürlich auch die Funktionstaste F15 als Exit-Funktionstaste verfügbar sein, wie bereits beschrieben. Dieser Funktionstaste ist der Funktionscode `FT15` zuzuweisen. Beachten Sie bitte, dass dieser Funktionscode ein Exit-Code sein muss und das Dynpro ein Exit-Modul besitzen muss. Als Folge-Dynpro wird die eigene Dynpro-Nummer benutzt.

Das Dynpro soll lediglich über ein Modul im PBO-Teil und zwei Module im PAI-Teil der Ablauflogik verfügen. Im PBO-Teil wird der oben beschriebene Status gesetzt. Der PAI-Teil verfügt über zwei Module, das bekannte Exit-Modul und das folgende PAI-Modul zur Auswertung des Funktionscodes. Nachfolgend das Listing der Ablauflogik:

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
*
PROCESS AFTER INPUT.
  MODULE exit_100 AT EXIT-COMMAND.
  MODULE user_command_0100.
```

Das User-Command-Modul enthält die wichtigsten Teile der Programmlogik:

```
MODULE user_command_0100 INPUT.
  CASE fcode.
    WHEN 'FNC1'.
      LEAVE TO LIST-PROCESSING.
      NEW-PAGE NO-TITLE NO-HEADING.
      SET PF-STATUS ' '.
      WRITE / 'Eingebettete List-Verarbeitung'.
      WRITE   'mit Standardstatus für Reports'.
      WRITE / 'Beenden mit F3 o. Ä.'.

    WHEN 'FNC2'.
      LEAVE TO LIST-PROCESSING.
      NEW-PAGE NO-TITLE NO-HEADING.
      WRITE / 'Eingebettete List-Verarbeitung,'.
      WRITE   'Status des Dynpros gilt weiter'.
      WRITE /
        'Beenden mit Funktionstaste F12 oder 3. Drucktaste'.
  ENDCASE.
ENDMODULE.                          " USER_COMMAND_0100  INPUT
```

In dem Include der PAI-Module wird manuell die folgende Zeitpunktanweisung und der dazugehörige Anweisungsblock eingetragen.

```
AT USER-COMMAND.
  IF sy-ucomm = 'FNC3' OR sy-ucomm = 'FT12'.
    LEAVE LIST-PROCESSING.
  ENDIF.
```

Über den Funktionscode `FNC1` wird die erste Liste angezeigt. Nach dem Einschalten der List-Verarbeitung werden der Standardtitel und die Listenüberschrift abgeschaltet. Dazu dient das Kommando `NEW-PAGE`. Es erzeugt in diesem Fall keinen Seitenvorschub, da vor Aufruf des Kommandos keine Ausgaben auf die Liste erfolgten. Es dient hier nur dazu, den Titel und die Überschrift abzuschalten. Als Bezeichner für den zu setzenden Status dient eine leere Zeichenkette. Ein derartiger Status existiert nicht, also wird der Standardstatus der Listenverarbeitung wirksam. Die durch diesen Status zur Verfügung gestellten Funktionen werden durch das System selbst ausgewertet, zusätzliche Programmierarbeiten sind nicht erforderlich. Beispielsweise kann die Liste problemlos gedruckt werden, da das Drucken zur Standardfunktionalität eines Reports gehört.

Bei der zweiten Version der List-Erzeugung, ausgelöst durch den Funktionscode FNC2, wird auf das Setzen eines neuen Status verzichtet. Daher steht weiterhin der Status des Dynpros zur Verfügung. Die Beendigung der List-Verarbeitung mit der Funktionstaste F3 ist nun nicht mehr möglich, da der Status nicht den automatisch vom System ausgewerteten Funktionscode bereitstellt. Das muss mittels der verschiedenen, individuell zu programmierenden Zeitpunktanweisungen der Listenverarbeitung erfolgen. Eine dieser Zeitpunktanweisungen folgt daher als Beispiel für diese Programmierweise unmittelbar an das Modul. Das Auslösen eines beliebigen, nicht durch das System zu verarbeitenden Funktionscodes löst das Ereignis AT USER-COMMAND aus. Eine entsprechenden Abfrage führt bei der dritten Drucktaste und der Funktionstaste F12 zum Beenden der List-Verarbeitung.

In diesem Beispiel wird das Dynpro angezeigt, um durch die Drucktasten die Auswahl zwischen den beiden unterschiedlichen Listen zu ermöglichen. Es besitzt damit eigene Funktionalität. In echten Anwendungen ist eine derartige Auswahl meist nicht erforderlich. Das Dynpro bildet in diesem Fall nur den Rahmen, der es überhaupt erst ermöglicht, eine List-Verarbeitung in eine Dialoganwendung einzufügen. Um den modularen Charakter des Dynpros innerhalb der Anwendung zu wahren, sollten Dialog und List-Verarbeitung getrennt werden. Innerhalb eines Dynpros, in dem eine List-Verarbeitung ausgeführt wird, findet daher meist keine Dialogverarbeitung statt. Dies ist keine zwingende Forderung, sondern lediglich ein Hinweis, der aus praktischer Erfahrung resultiert. So strukturierte Anwendungen sind einfacher zu programmieren und zu warten als Anwendungen, die in einem Dynpro mehrere Funktionen vereinen. Die Anzeige des Dialogbildschirmes ist in solchen Fällen nicht notwendig, eher störend. Im ABAP-Befehlssatz existiert daher eine Anweisung, die es ermöglicht, den Dialog in einem Dynpro zu überspringen und sofort die PAI-Verarbeitung auszuführen. Diese Anweisung lautet

```
SUPPRESS DIALOG.
```

Anhand einer Anwendung (SAPMY450), die durch leichte Änderungen des letzten Beispiels entsteht, kann diese Anweisung demonstriert werden. Im PBO-Teil des Dynpros ist lediglich ein Modul erforderlich, das die eben vorgestellte Anweisung SUPPRESS DIALOG ausführt. Das Setzen eines Status ist nicht erforderlich. Da der eigentliche Dialog übersprungen wird, kann auch kein Funktionscode ausgelöst werden. Im PAI-Abschnitt des Dynpros ist daher ebenfalls nur ein Modul erforderlich, in dem die Listenerstellung ausgeführt wird.

```
MODULE user_command_0100 INPUT.
  LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
  NEW-PAGE NO-TITLE NO-HEADING.
  SET PF-STATUS ' '.
  WRITE / 'Eingebettete List-Verarbeitung'.
  WRITE   'mit Standardstatus für Reports'.
  WRITE / 'Beenden mit F3 o. Ä.'.
ENDMODULE.
```

Es erfolgt keinerlei Auswertung irgendwelcher Funktionscodes, da diese ohne Dialog nicht erzeugt werden können. Ohne den Zusatz `AND RETURN TO SCREEN 0` zur `LEAVE`-Anweisung würde das Dynpro nach Verlassen der List-Verarbeitung erneut ausgeführt, da nach Beenden der List-Verarbeitung immer wieder der PBO-Teil des rufenden Dynpros ausgeführt wird. Das Beenden des Dynpros ist daher nur über die demonstrierte Zeitpunktanweisung möglich. Je nachdem, wie das Dynpro aufgerufen wird, beendet das Kommando der Zeitpunktanweisung in Verbindung mit dem `AND RETURN`-Zusatz die gesamte Anwendung (Aufruf des Dynpros mit `LEAVE TO SCREEN`) oder kehrt nur zu einem anderen Dynpro zurück (Aufruf mit `CALL SCREEN`). In den meisten Fällen ist die letzte Variante die empfehlenswerte, um eine Liste in eine Dialoganwendung einzubinden.

3.8.2 Eingabehilfe

Der praktische Einsatz der eingebetteten List-Verarbeitung erfolgt oft, um eine Eingabehilfe zu realisieren. Zum Abschluss dieses Unterabschnittes soll daher demonstriert werden, wie ein interaktiver Report als Eingabehilfe für ein Dynpro-Feld benutzt werden kann. Da nicht alle Schritte der Programmentwicklung detailliert beschrieben werden, stellt das Erstellen des Programms auch eine gute Übung zur Bedienung der Entwicklungsumgebung und der verschiedenen Werkzeuge dar.

Zunächst wird ein Dynpro erzeugt und in dieses eine einfache List-Verarbeitung eingebunden. Diese Liste zeigt zunächst die Nummern aller verfügbaren Farbattribute an. Ein Doppelklick auf eine Nummer blendet in der Liste einige weitere Zeilen mit Beispielen zu dieser Farbe und anderen Attributen ein bzw. aus.

In einem zweiten Abschnitt wird ein weiteres Dynpro mit einem einzigen Eingabefeld erzeugt. Für dieses Eingabefeld wird eine Eingabehilfe programmiert, die das im ersten Teil erzeugte Dynpro benutzt.

Um das komplette Beispiel zu erstellen, führen Sie folgende Arbeitsschritte aus:

❶ Erstellen Sie einen Modul-Pool. Den Namen können Sie im Rahmen der bestehenden Konventionen beliebig wählen. Der Modul-Pool soll ein Top Include besitzen, in dem die folgenden Deklarationen eingetragen werden:

```
DATA: BEGIN OF itab OCCURS 20,
  index TYPE i,
  col   TYPE i,
  detail,
  text(20),
END OF itab.

DATA: g_init    VALUE 'X',  " Flag: erster Start des Dynpros
      g_line   TYPE i,      " oberste angezeigte Zeile
```

```
                              " der Liste
      g_page   TYPE i,        " oberste angezeigte Seite
                              " der Liste
      g_int     TYPE i,       " Hilfsfeld
      g_col(20).              " globales Feld zur Rückgabe der
                              " gewählten Zeile, für Teil 2
```

❷ Legen Sie in diesem Modul-Pool ein Dynpro 100 an. Der Fullscreen des Dynpros bleibt leer. Eine Zuweisung zum OK-Code-Feld ist nicht erforderlich. Die Ablauflogik ist im folgenden Listing zu sehen:

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
*
PROCESS AFTER INPUT.
  MODULE user_command_0100.
```

❸ Erzeugen Sie die beiden Module gemäß folgendem Listing:

```
MODULE status_0100 OUTPUT.
* Status für Liste setzen
  SET PF-STATUS 'STAT1'.

* Aufbau der Hilfstabelle bei erstem Aufruf
  IF g_init = 'X'.

* interne Tabelle initialisieren
    CLEAR itab.
    REFRESH itab.

* globale Felder initialisieren
    CLEAR: g_init, g_int, g_line, g_page.

* einen Datensatz für jede Farbe
    WHILE g_int < 8.
      itab-index = g_int + 1.
      itab-col = g_int.
      itab-detail = ' '.
      itab-text = 'Color'.
      itab-text+7(5) = g_int.
      APPEND itab.
      ADD 1 TO g_int.
    ENDWHILE.
  ENDIF.

* Eingabe in Dynpro unterdrücken
  SUPPRESS DIALOG.

ENDMODULE.                              " STATUS_0100  OUTPUT
```

```
MODULE user_command_0100 INPUT.
  LEAVE TO LIST-PROCESSING.
  NEW-PAGE NO-TITLE NO-HEADING.

* jede Zeile der Itab ausgeben
  LOOP AT itab.
    WRITE: / itab-text INTENSIFIED OFF.
    HIDE itab.

* falls Flag gesetzt, dann Beispiele für Farbe und Attribute
    IF itab-detail = 'X'.
      WRITE: /5 'Normal         ' COLOR = itab-col
                                  INTENSIFIED OFF
                                  INVERSE OFF.
      HIDE itab.

      WRITE: /5 'Intensiv       ' COLOR = itab-col
                                  INTENSIFIED ON
                                  INVERSE OFF.
      HIDE itab.
      WRITE: /5 'Invers         ' COLOR = itab-col
                                  INTENSIFIED OFF
                                  INVERSE ON.
      HIDE itab.

      WRITE: /5 'Intensiv Invers' COLOR = itab-col
                                  INTENSIFIED ON
                                  INVERSE ON.
      HIDE itab.
      WRITE /.

    ENDIF.
  ENDLOOP.

* wegen LINE-SELECTION
  CLEAR itab.

* wenn Neuaufbau der Liste nach Zeilenselektion,
* dann zur alten Position springen
  SCROLL LIST TO PAGE g_page LINE g_line.
ENDMODULE.                    " USER_COMMAND_0100  INPUT
```

❹ Der im `INIT`-Modul aufgerufene Status `STAT1` muss erzeugt werden. Er sollte im Oberflächeneditor mit der Menüfunktion ZUSÄTZE | VORLAGE ABGLEICHEN von einem Liststatus abgeleitet werden. In diesem Status wird eine neue Funktion `SELE` angelegt, die auch über eine Funktions- und Drucktaste erreichbar sein muss. Als Text kann „Übernehmen“ benutzt werden. Außerdem

wird der Funktionscode für ABBRECHEN (Funktionstaste F12) in FT12 geändert. Der letztgenannte Funktionscode muss ein Exit-Funktionscode sein.

❺ Im Modul-Pool werden die folgenden beiden Zeitpunktanweisungen programmiert. Zunächst AT LINE-SELECTION:

```
AT LINE-SELECTION.
  IF NOT itab IS INITIAL.

* merken aktuelle Position der ersten angezeigten Zeile
    g_page = sy-cpage.
    g_line = sy-staro.

* Flag für Detailanzeige umschalten
    IF itab-detail <> ' '.
      itab-detail = ' '.
    ELSE.
      itab-detail = 'X'.
    ENDIF.

* Datensatz in Tabelle ändern
    MODIFY itab INDEX itab-index.

* Rücksprung zum PBO-Teil des Dynpros -> Neuaufbau der Liste
    LEAVE LIST-PROCESSING.
  ENDIF.
```

Als zweites AT USER-COMMAND wird notiert:

```
AT USER-COMMAND.
  CASE sy-ucomm.
    WHEN 'FT12'.
      LEAVE TO SCREEN 0.
  ENDCASE.
```

❻ Erzeugen Sie eine Transaktion, die das Dynpro 100 als Startdynpro benutzt. Testen Sie die Anwendung. Sie erzeugt eine kurze Liste mit den Nummern der in Listen verfügbaren Farben. Ein Doppelklick auf eine dieser Zeilen blendet vier zusätzliche Zeilen mit Beispielen zur Farbdarstellung ein.

❼ Erzeugen Sie ein zweites Dynpro (200). Im Fullscreen dieses Dynpros wird ein Eingabefeld für das globale Datenfeld g_col erzeugt. Die Ablauflogik des zweiten Dynpros enthält drei Modulaufrufe:

```
PROCESS BEFORE OUTPUT.
  MODULE status_0200.
*
PROCESS AFTER INPUT.
  MODULE exit_0200 AT EXIT-COMMAND.
```

```
PROCESS ON VALUE-REQUEST.
  FIELD g_col MODULE get_color.
```

❽ Die vom Dynpro 200 aufgerufenen Module sind anzulegen:

```
MODULE status_0200 OUTPUT.
  SET PF-STATUS 'STAT1'.
ENDMODULE.                              " STATUS_0200  OUTPUT

MODULE exit_0200 INPUT.
  LEAVE TO SCREEN 0.
ENDMODULE.                                " EXIT_0200  INPUT

MODULE get_color INPUT.
  g_init = 'X'.
  CALL SCREEN 100 STARTING AT 5 5 ENDING AT 30 10.
ENDMODULE.                               " GET_COLOR  INPUT
```

❾ In der Zeitpunktanweisung `AT USER-COMMAND` wird die Auswertung des eben angelegten Funktionscodes nachgetragen. Dazu ist lediglich ein zweiter `WHEN`-Zweig erforderlich.

```
WHEN 'SELE'.
  IF NOT itab IS INITIAL.
    g_col = itab-text.
    LEAVE TO SCREEN 0.
  ELSE.
    MESSAGE ID 'YZ4' TYPE 'I' NUMBER '000'.
  ENDIF.
```

❿ Erzeugen Sie eine zweite Transaktion, die das Dynpro 200 als Start-Dynpro benutzt, und testen Sie die Anwendung. Neben dem Eingabefeld erscheint eine kleine Schaltfläche, über die das Dynpro 100 in Form eines Popups aufgerufen werden kann. Dort können Sie einen Eintrag markieren und mit der Schaltfläche ÜBERNEHMEN in das Eingabefeld übernehmen.

Diskussion

Im Schritt 1 werden zunächst die im Programm benötigten Daten deklariert. Die interne Tabelle, genauer gesagt, das Flag `detail` steuert später die Anzeige der zusätzlichen Zeilen mit den Beispielen zu den Ausgabeformaten. Die beiden Felder `index` und `col` werden, ebenso wie der Text, nur mitgeführt, um das Programm etwas zu vereinfachen. Die dort enthaltenen Informationen lassen sich prinzipiell auch zur Laufzeit aus dem Index (Datensatznummer) der internen Tabelle ableiten.

Der Aufruf einer Liste in einer Dialoganwendung erfordert ein Dynpro. Dieses wird in den Schritten 2 und 3 angelegt. Im Init-Modul wird die interne Tabelle mit acht Datensätzen gefüllt. Das Löschen der Tabelle vor dem Füllen ist im ers-

ten Teil des Beispiels noch nicht unbedingt notwendig. Auf die Gründe dafür wird später eingegangen. Neben dem Aufbau der Liste werden einige globale Felder initialisiert, um einen definierten Grundzustand zu erreichen. Sie speichern später die aktuelle Zeile der Liste.

Am Ende des Init-Moduls wird die Dialogverarbeitung mit `SUPPRESS DIALOG` unterdrückt. Die Anwendung setzt daher sofort mit der Abarbeitung des einzigen PAI-Moduls fort. In diesem Modul wird die List-Verarbeitung eingeschaltet und die Liste aufgebaut. Für jede Zeile der internen Tabelle erscheint in der Liste mindestens eine Zeile. Falls in der internen Tabelle das Flag `detail` eingeschaltet ist, werden vier zusätzliche Zeilen mit den unterschiedlichen Attributkombinationen ausgegeben. Bei der Erstellung der internen Tabelle wird dieses Flag zurückgesetzt, die Liste enthält also nur acht einfache Zeilen. Für jede ausgegebene Zeile wird der gesamte Datensatz mit `HIDE` in den verborgenen Datenbereich gestellt.

Nach Darstellung der Liste kann der Anwender zwei Aktivitäten ausführen. Die Betätigung der Funktionstaste F12 oder des entsprechenden Symbols löst den Funktionscode `FT12` aus. Dieser wird von der Zeitpunktanweisung `AT USER-COMMAND` verarbeitet und führt zum Beenden der Anwendung, da zum Dynpro 0 gesprungen wird. An dieser Stelle darf nicht der Standardfunktionscode verwendet werden, der im Menu Painter vorgeschlagen wird. Dieser würde vom System ausgewertet werden und lediglich die List-Verarbeitung, aber nicht die gesamte Anwendung beenden. Nach der List-Verarbeitung wird aber wieder das rufende Dynpro aktiviert, wodurch die Liste sofort wieder erzeugt würde. Ein geordnetes Beenden der Anwendung ist nur durch die selbst programmierte Auswertung eines Funktionscodes möglich.

Der interessanteste Teil der Anwendung ist der Zeitpunkt `AT LINE-SELECTION`, der im Schritt 5 programmiert wird. Nach einer Zeilenselektion wird der mit `HIDE` in den verborgenen Datenbereich gestellte Datensatz automatisch in die Kopfzeile `itab` zurückgeholt, falls eine der Datenzeilen der Liste selektiert wurde. Dies wird durch die einleitende `IF`-Anweisung erkannt.

Die Darstellung der Liste am Bildschirm wird im Wesentlichen durch das Flag `detail` in den Datensätzen der internen Tabelle gesteuert. Dieses wird in der Kopfzeile der Tabelle umgeschaltet. Anschließend wird der in der Kopfzeile enthaltene Datensatzindex benutzt, um den Inhalt der internen Tabelle zu aktualisieren. Die Verwendung des Index vereinfacht die Anwendung. Falls der Index des Datensatzes nicht direkt aus den zurückgeholten Daten ermittelt werden kann, müsste zunächst der Satz in der internen Tabelle gesucht werden. Im verborgenen Datenbereich muss daher auf jeden Fall ein eindeutiger Schlüssel für jeden Datensatz abgespeichert werden. Nach Modifizieren der internen Tabelle wird die List-Verarbeitung durch ein entsprechendes Kommando beendet. Der PBO-Teil des Dynpros 100 wird erneut prozessiert. Das führt natürlich zur erneuten Anzeige der Liste. Diese wird entsprechend den eben gesetzten oder zurückgesetzten Flags komplett neu aufgebaut.

Bei einem Neuaufbau wird eine Liste beginnend mit der ersten Zeile angezeigt. Da auch diese kleine Demoliste durchaus länger als eine Bildschirmseite werden kann, würde sich der Anwender nach einem Doppelklick unter Umständen auf einer anderen Seite der Liste wiederfinden als vor der Zeilenselektion. Um dies zu vermeiden, werden bei `AT LINE-SELECTION` die aktuelle Zeilen- und Seitennummer in globalen Feldern gesichert. Nach dem Aufbau der Liste wird mit der Anweisung `SCROLL` zur jeweiligen Zeile der Liste gesprungen.

Die eingebettete List-Verarbeitung wird sehr oft benutzt, um dem Anwender eine Eingabehilfe bereitzustellen, in der er aus einer bestimmten Menge von Werten einen auswählen kann. Mit Hilfe des eben erzeugten Programms kann das prinzipielle Verfahren sehr einfach demonstriert werden. Zunächst wird im Schritt 6 der obigen Anleitung ein Dynpro 200 mit einem Eingabefeld erzeugt. Die einzige Aufgabe dieses Dynpros besteht darin, die Eingabehilfe für ein Datenfeld zu demonstrieren. Davon abgesehen verfügt es über keinerlei echte Funktionalität. Das Datenfeld, das mit dem Eingabefeld zusammenarbeitet, muss sowohl vom Dynpro 100 als auch 200 erreichbar sein, da es zur Datenübergabe zwischen den beiden Dynpros benutzt wird. Das Dynpro 200 wird im Dialogmodus ausgeführt. Es benötigt daher einen Status, in dem ein Funktionscode zum Beenden der Anwendung verfügbar sein muss. Der Einfachheit halber kann hier `STAT1` benutzt werden, auch wenn hier einige Funktionscodes enthalten sind, die nicht benötigt werden.

Die Eingabehilfe wird durch einen weiteren Zeitpunkt in der Ablauflogik des Dynpros 200 realisiert. Er wird prozessiert, wenn im Eingabefeld die Funktionstaste F4 betätigt wird. Die Anweisungen zum Aufruf der Eingabehilfe sind denkbar einfach. Zunächst wird das Flag gesetzt, mit dem die Initialisierung des Dynpros 100 ausgelöst werden kann. Bei mehrmaligem Aufruf der Eingabehilfe wird das Dynpro 100 zwar immer wieder neu aufgerufen, die interne Tabelle `itab` liegt aber im globalen Datenbereich der Anwendung und bleibt bis zum Verlassen der Transaktion bestehen. Um bei jedem Start der Eingabehilfe die Grundform der Liste zu erhalten, wird per Flag `g_init` der Neuaufbau der Hilfstabelle veranlasst.

Anschließend wird das Dynpro 100 mit `CALL` aufgerufen. Mit den Zusätzen `STARTING AT` und `ENDING AT` wird die Ausführung des Dynpros und damit der Liste in einem Popup erzwungen. Das Dynpro 100 ist vollständig lauffähig. Im Popup wird die Liste angezeigt, per Doppelklick kann die Detailanzeige für einen Datensatz eingeblendet werden, mit F12 wird der Rücksprung zum Dynpro 200 eingeleitet. Da das Dynpro 100 mit der Anweisung `CALL` aufgerufen wurde, führt die Anweisung `LEAVE TO SCREEN 0` nicht zum Beenden der gesamten Anwendung, sondern nur zum Rücksprung auf das rufende Dynpro.

Im Dynpro 100, genauer zum `AT USER-COMMAND`-Zeitpunkt, muss nur noch die Datenübergabe an das rufende Dynpro realisiert werden. Es würde ausreichen, unmittelbar vor `LEAVE` das globale Datenfeld `g_col` mit dem Text aus der Kopfzeile von `itab` zu füllen, die ja auch bei `AT USER-COMMAND` mit den Daten aus

dem verborgenen Datenbereich gefüllt wird. Um dem Anwender die Möglichkeit zu geben, ohne Datenübergabe abzubrechen, wird für die Datenübernahme ein zusätzlicher Funktionscode (`SELE`) in den Status des Dynpros 100 aufgenommen und zum Zeitpunkt `AT USER-COMMAND` ausgewertet. Beim Auslösen des Funktionscodes wird das globale Feld `g_col` vor dem Rücksprung mit einem Wert der selektierten Zeile gefüllt. Das Dynpro wird mit diesem Funktionscode allerdings nur dann beendet, wenn eine korrekte Zeile gewählt wurde. Wie im Listing angedeutet, sollte der Anwender eine Fehlernachricht erhalten, wenn keine Zeile ermittelt werden konnte. Da die Nachrichtenklasse und die Nachricht aller Wahrscheinlichkeit nach nicht existieren, werden Sie lediglich ein leeres Popup ohne Fehlermeldung erhalten. Auch wenn Nachrichtenklasse und Nachricht existieren sollten, wird zwischen Nachrichtentext und Fehlerursache kein Zusammenhang bestehen.

3.8.3 Dynpros in Reports

Der Aufruf von Dynpros ist nicht zwangsweise davon abhängig, ob das Programm ein Modul-Pool ist oder nicht. Der Programmtyp hat vor allem Einfluss auf den Start der Anwendung, nicht aber auf die in ihr enthaltenen Elemente. Auch Reports können daher Dynpros und Module enthalten und ausführen. Auf diese Weise können spezielle Probleme schneller gelöst werden als unter alleiniger Verwendung der Dialoganwendungen. Folgende Vorteile bestehen:

- Im Report kann relativ einfach ein Selektionsbild zur Vorselektion von Daten erstellt und geändert werden. Dazu muss kein Dynpro bearbeitet werden, einige `PARAMETERS`- oder `SELECTION`-Anweisungen reichen aus.
- Eventuell vorhandene logische Datenbanken können eingesetzt werden, um die Datenselektion zu vereinfachen.
- Selektierte Datensätze können in einem interaktiven Report zur Auswahl (z. B. per Doppelklick) angeboten werden. Die Bearbeitung erfolgt in einem echten Dynpro. Ein solcher Report ist leichter und schneller zu programmieren als eine ebenfalls für diesen Zweck verwendbares Dynpro mit einem Step Loop oder Table View.

Wegen der Vermischung von List- und Dialogverarbeitung sind derartige Anwendungen in der Bedienung etwas komplizierter, sie werden daher oft für systemnahe Aufgaben und einen kleinen Benutzerkreis verwendet.

Wenn in einem Report ein Dynpro aufgerufen werden soll, erfolgt dies mit dem Kommando

```
CALL SCREEN screen.
```

Die zum Dynpro gehörenden Module müssen im aktuellen Programm verfügbar sein.

Innerhalb eines Reports kann ein Dynpro mit zwei verschiedenen Methoden erzeugt werden. Am einfachsten ist es, den Navigationsmechanismus zu benutzen. Nach einem Doppelklick auf die Dynpro-Nummer in der oben angeführten Anweisung erzeugt das R/3-System das gewünschte Dynpro mit allen Elementen. Es ist natürlich auch möglich, das Dynpro von der Objektliste des Object Browser aus anzulegen.

Die Demonstration des Verfahrens ist anhand des Programms `YZ434200` relativ einfach möglich. Der Aufruf des Funktionsbausteins zum Anzeigen des Quelltextes soll ersetzt werden durch einen kleinen Dialog, in dem die Bezeichnung des Programms geändert werden kann.

Der eigentliche Report muss dazu zunächst mit einigen zusätzlichen Deklarationen ergänzt werden. Außerdem ist es sinnvoll, ihn etwas einfacher zu gestalten, um die wesentlichen Punkte besser hervorheben zu können. Nachfolgend das Listing des geänderten Reports: Sie können ihn sehr einfach durch Kopieren des oben erwähnten Programms und Modifizieren einiger Zeilen erstellen. Im Deklarationsteil müssen zwei zusätzliche Datenfelder eingeführt werden. Es handelt sich um das Feld `l_text`, das temporär den Programmtitel enthält sowie um das Feld `fcode`, das im Dialog den Funktionscode aufnehmen muss.

```
REPORT  yz436030 NO STANDARD PAGE HEADING.
DATA: trdir    TYPE trdir,        " Table with ABAP-Programs
      textpool TYPE textpool.     " Structure for texts



* internal table for texts
DATA: BEGIN OF itext OCCURS 20.
        INCLUDE STRUCTURE textpool.
DATA: END OF itext.

DATA:                              "new
  fcode   LIKE sy-ucomm,           "new
  l_text  LIKE itext-entry.        "new

FORMAT COLOR 4.

SELECT * FROM trdir
  WHERE cnam = sy-uname            " User
    AND appl <> 'S'                " no generated programs
    AND subc IN ('1', 'M', 'F') . " no includes

* write program name
  WRITE: /(60) trdir-name.
```

```
* read texts
  CLEAR itext.
  REFRESH itext.
  READ TEXTPOOL trdir-name INTO itext LANGUAGE sy-langu.

* find description
  CLEAR itext.
  itext-id = 'R'.
  READ TABLE itext.

* write description
  IF sy-subrc = 0.
    WRITE: /(60) itext-entry.
    HIDE: trdir, itext-entry.
  ENDIF.
  WRITE /.
ENDSELECT.

CLEAR trdir.

AT LINE-SELECTION.

*....if-block new
  IF NOT trdir IS INITIAL.
    l_text = itext-entry.
    CALL SCREEN 100.
    MODIFY CURRENT LINE FIELD VALUE itext-entry.
    CLEAR trdir.
  ENDIF.
```

Generieren Sie das Programm, damit im Dynpro die programminternen Felder vom Einfügewerkzeug gefunden werden können.

Ein Doppelklick auf die Dynpro-Nummer sorgt dafür, dass vom System ein neues Dynpro erzeugt wird. In diesem Dynpro erzeugen Sie ein Eingabefeld für das Datenfeld `l_text` sowie ein Anzeigefeld für `trdir-name`. Da das Feld `l_text` sehr lang ist, sollte die sichtbare Länge im Dynpro eingeschränkt werden. Außerdem sollten Sie für dieses Feld die Groß-/Kleinschreibung ermöglichen. Die Ablauflogik und die beiden Module geben Auskunft über die Funktion des Dynpros und indirekt auch über den Aufbau des Status, der natürlich ebenfalls erzeugt werden muss. Die Ablauflogik entspricht dem automatisch generierten Vorschlag.

```
PROCESS BEFORE OUTPUT.
  MODULE status_0100.
*
PROCESS AFTER INPUT.
  MODULE user_command_0100.
```

Im Status-Modul wird lediglich der Status gesetzt.

```
MODULE STATUS_0100 OUTPUT.
  SET PF-STATUS 'STAT1'.
ENDMODULE.                 " STATUS_0100  OUTPUT
```

Die drei im User-Command-Modul ausgewerteten Funktionscodes ermöglichen es, das Dynpro mit und ohne Sichern des editierten Wertes zu beenden oder den alten Inhalt wieder herzustellen, ohne das Dynpro zu verlassen. Den Status `STAT1` erzeugen Sie bitte in eigener Verantwortung so, dass die drei Funktionscodes erzeugt werden können. Da alle Funktionscodes im User-Command-Modul ausgewertet werden und kein Exit-Modul existiert, muss für die Funktionscodes ausnahmsweise keine Exit-Eigenschaft gesetzt werden.

```
MODULE user_command_0100 INPUT.
  CASE fcode.

*.....save and exit
    WHEN 'FT11'.
      CLEAR itext.
      REFRESH itext.
      READ TEXTPOOL trdir-name INTO itext LANGUAGE sy-langu.
      CLEAR itext.
      itext-id    = 'R'.
      itext-key   = space.
      READ TABLE itext.
      itext-entry = l_text.
      IF sy-tabix = 0.
        APPEND itext.
      ELSE.
        MODIFY itext INDEX sy-tabix.
      ENDIF.
      INSERT TEXTPOOL trdir-name FROM itext
        LANGUAGE sy-langu.
      LEAVE TO SCREEN 0.

*.....exit without save
    WHEN 'FT03'.
      LEAVE TO SCREEN 0.

*.....recall text
    WHEN 'UNDO'.
      l_text = itext-entry .
  ENDCASE.
ENDMODULE.                         " USER_COMMAND_0100  INPUT
```

Das Zurückschreiben des geänderten Wertes in die Datenbank ist etwas kompliziert, da zunächst wieder der Textpool in eine interne Tabelle gelesen, in dieser

der entsprechende Eintrag gesucht und überschrieben sowie die gesamte interne Tabelle in den Textpool zurückgeschrieben werden muss. Würde nur der eine überarbeitete Eintrag für den Programmtitel in den Textpool geschrieben werden, so gingen alle anderen Einträge (Textelemente etc.) verloren.

3.9 Funktions- und Dialogbausteine

Im Zusammenhang mit den bisher behandelten Kommandos wurden Funktions- und Dialogbausteine erwähnt und gelegentlich ohne nähere Erläuterung benutzt. Vor allem Funktionsbausteine sind sehr leistungsfähige Hilfsmittel zum Modularisieren von Anwendungen.

Funktionsbausteine werden innerhalb eines so genannten *Function-Pools* abgelegt. Ein Function-Pool umfasst einen oder mehrere logisch zusammengehörende Funktionsbausteine. Der Function-Pool stellt für alle in ihm enthaltenen Funktionsbausteine globale Datendeklarationen und allgemein verwendbare Unterprogramme (Form-Routinen) bereit. Insofern bestehen Ähnlichkeiten zu einem Modul-Pool. Die Inhalte der im globalen Datenbereich (im Rahmenprogramm) deklarierten Datenfelder oder internen Tabellen bleiben solange erhalten, wie das rufende Programm noch aktiv ist. Man spricht daher auch von einem lokalen Gedächtnis.

Funktionsbausteine ähneln Unterprogrammen (Forms). Sie kapseln Programmcode und Daten. Der Zugriff auf Daten kann nur über die Schnittstelle des Funktionsbausteins erfolgen. Zwischen Funktionsbausteinen und herkömmlichen Unterprogrammen bestehen aber auch wesentliche Unterschiede:

- Funktionsbausteine müssen in einem Programm enthalten sein, das einen speziellen Typ besitzt. Diese speziellen Programme werden Function-Pool genannt. Funktionsbausteine können über globale Daten diese Programms verfügen. Ein Function-Pool mit Funktionsbausteinen wird auch als Funktionsgruppe bezeichnet.
- Funktionsbausteine können aus anderen Programmen heraus über ihren Namen aufgerufen werden. Die Angabe des Function-Pool ist dabei nicht notwendig. Funktionsbausteine müssen daher einen systemweit eindeutigen Namen besitzen.
- Funktionsbausteine können mittels einer eigenen Pflegetransaktion SE37 unabhängig vom Function-Pool gepflegt werden.
- Funktionsbausteine besitzen eine exakt festgelegte und dokumentierbare Schnittstelle. Diese Schnittstelle kann unter Berücksichtigung spezieller Bedingungen erweitert werden, ohne dass an bestehenden Programmen, aus denen der Funktionsbaustein aufgerufen wird, etwas geändert werden muss.

- Eine Testumgebung erleichtert die Prüfung und Wartung von Funktionsbausteinen. Für jeden Baustein können Testdaten aufbewahrt werden, mit denen nach Überarbeitung eines Funktionsbausteins das Verhalten überprüft werden kann.

3.9.1 Erzeugen von Funktionsbausteinen

Entsprechend der bisherigen Verfahrensweise beschreibt dieser Abschnitt das Erzeugen einer Funktionsgruppe und eines einfachen Funktionsbausteins. Außerdem wird die Testumgebung demonstriert. Der darauf folgende Abschnitt vermittelt dann Detailkenntnisse.

Beim Erzeugen eines neuen Funktionsbausteins muss angegeben werden, zu welcher Funktionsgruppe er gehören soll. Falls noch keine geeignete Funktionsgruppe existiert, muss sie angelegt werden. Wie auch bei anderen Entwicklungsobjekten, können Sie Funktionsgruppen und Funktionsbausteine auf unterschiedliche Weise erzeugen. Letztlich führen aber alle Methoden zum selben Werkzeug, dem Editor für Funktionsbausteine. Dieser ist direkt über den Transaktionscode `SE37` sowie vom Hauptmenü der SAP Workbench aus über die Menüfunktion ENTWICKLUNG | FUNCTION BUILDER erreichbar.

Die einfachste Variante besteht wiederum im Einsatz des Object Navigator. Im Startbild des Object Navigator bietet das Kontextmenü des Pakets die Funktion ANLEGEN | FUNKTIONSGRUPPE an. Der Aufruf dieser Funktion führt zum Einblenden eines Popups, in dem Sie den Namen der Funktionsgruppe sowie einen beschreibenden Kurztext eingeben müssen. Der Name kann frei gewählt werden, wobei der erste Buchstabe den bereits erläuterten Konventionen genügen sollte (A-X für SAP, Y und Z für Kundenentwicklungen bzw. reservierter Namensraum). Der Name des Function-Pool wird vom System aus der Zeichenkette `SAPL`, gefolgt vom Namen der Funktionsgruppe, gebildet. In diesem Beispiel wird als Funktionsgruppenname `YZZ1` benutzt. Das `Y` ist fest vorgegeben, die `1` steht für die erste Funktionsgruppe. Das zweite und dritte Zeichen, hier das `ZZ`, sollte bei Ausführung dieses Beispiels durch mehrere Programmierer zur Unterscheidung der User genutzt werden. Abbildung 3.158 zeigt dieses Popup.

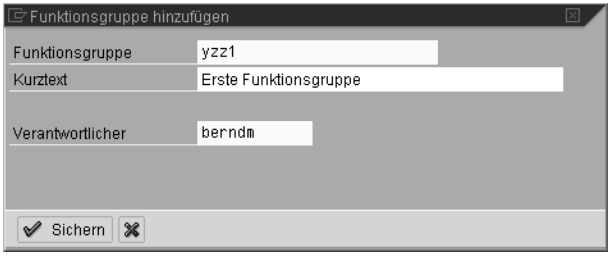

Abbildung 3.158 **© SAP AG**
Anlegen einer neuen Funktionsgruppe

Nach Bestätigen der Eingabe baut der Object Browser für existierende Funktionsgruppen die Objektliste auf. Sie enthält von Anfang an zwei Include-Dateien. Abbildung 3.159 zeigt die Objektliste einer Funktionsgruppe. Die Namensbildung der Include-Dateien erfolgt ähnlich wie bei den Include-Dateien von Modul-Pools. Der Name beginnt mit einem L, dann folgt der Name der Funktionsgruppe. Den Abschluss bilden drei Zeichen, die fast immer vom System vorgegeben werden. Es ist bei der Bearbeitung von Funktionsgruppen allerdings wesentlich seltener als bei anderen Programmarten erforderlich, eigene Include-Dateien anzulegen, da viele automatisch durch das System erzeugt werden. In einer neu angelegten Funktionsgruppe finden Sie auf jeden Fall ein `Top Include` (hier `LYZZ1TOP`), das Deklarationen aufnimmt und ein Include mit Verwaltungsinformationen. Dieses besitzt immer die Endung `UXX`. Dieses Include kann nicht manuell bearbeitet werden. Jeder Funktionsbaustein wird in einem eigenen Include abgelegt, dessen Name eine Endung der Form `Unn` erhält, wobei nn für eine zweistellige Zahl steht. Diese Includes werden beim Anlegen eines Funktionsbausteins automatisch erzeugt.

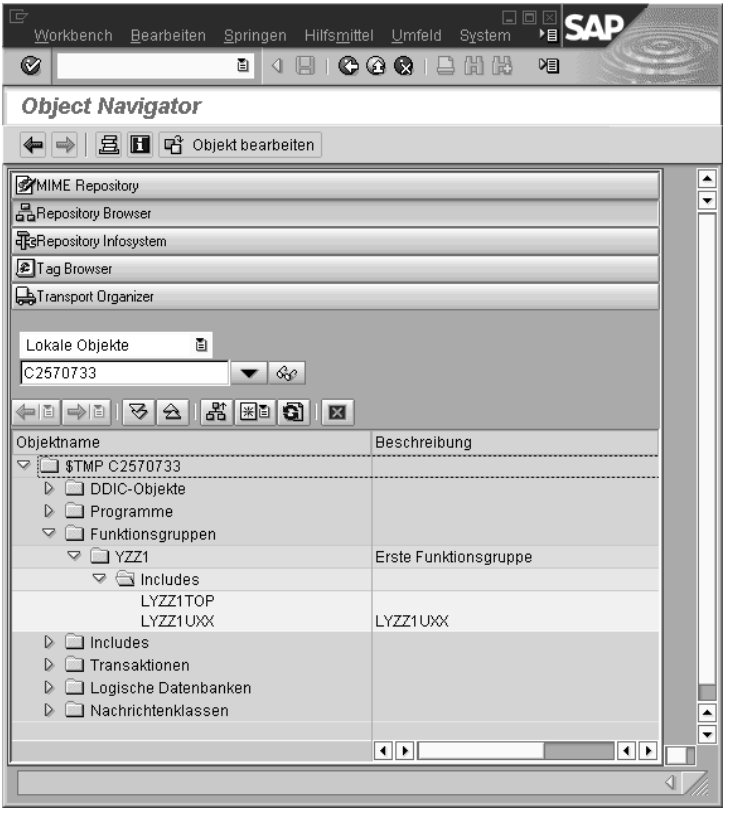

Abbildung 3.159 **© SAP AG**
Objektliste einer neu angelegten Funktionsgruppe

Das Top Include enthält den Programmkopf und nimmt später die Datendeklarationen auf. Datenfelder im Top Include sind innerhalb der Funktionsgruppe global gültig. Die Deklarationen können Sie entweder manuell eintragen oder per Navigationsmechanismus von Funktionsbausteinen aus erzeugen. Erfolgt innerhalb einer Anwendung erstmalig ein Zugriff auf eine Funktionsgruppe, werden nur bei diesem ersten Aufruf eines Funktionsbausteins alle globalen Felder der Funktionsgruppe initialisiert. Alle weiteren Änderungen dieser Felder sind lediglich durch die diversen Funktionsbausteine der Funktionsgruppe möglich. Die Werte der globalen Elemente bleiben so lange erhalten, bis die rufende Anwendung beendet wird.

Von der Objektliste des Object Navigator aus können die einzelnen Funktionsbausteine sehr einfach angelegt werden. Der Cursor wird auf den Namen der Funktionsgruppe platziert. Danach kann über das Kontextmenü mit der Menüfunktion ANLEGEN | FUNKTIONSBAUSTEIN eine neue Funktion erzeugt werden. Dies erfolgt wieder durch Eintragen des Namens des anzulegenden Funktionsbausteins in einem Popup (siehe Abbildung 3.160).

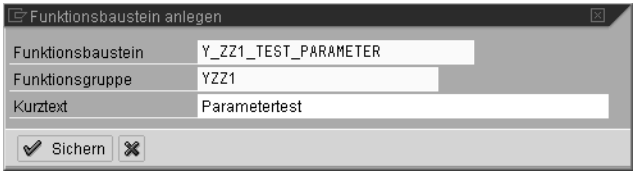

Abbildung 3.160 **© SAP AG**
Popup zum Erzeugen eines Funktionsbausteins

Für den ersten anzulegenden Baustein wird der Name `Y_ZZ1_TEST_PARAMETER` gewählt. Die Namen von Funktionsbausteinen müssen im gesamten R/3-System eindeutig sein. Es ist daher üblich, aber nicht zwangsweise erforderlich, die Namen von Funktionsbausteinen mit dem Namen der Funktionsgruppe zu beginnen. Dies vermeidet zunächst Verwechslungen bei ähnlich klingenden Namen. Außerdem können die eigentlichen Bezeichner der Funktionsbausteine eindeutig auf deren Funktion verweisen, ohne dass zur Gewährleistung der eindeutigen Namensgebung Kunstgriffe oder ungewöhnliche Abkürzungen erforderlich werden. Funktionsbausteine, die in Kundensystemen erzeugt werden, müssen mit den Buchstaben „Y“ oder „Z“ beginnen und an der zweiten Stelle des Namens einen Unterstrich „_“ besitzen.

Der erste Funktionsbaustein soll die Parameterschnittstelle und deren Eigenschaften demonstrieren. Nach Bestätigung des Eintrages durch Betätigen des Anlegen-Symbols im Popup erfragt das System weitere Angaben zum Funktionsbaustein. Es existieren zwei wichtige Teilwerkzeuge zur Pflege eines Funktionsbausteins. Das eine ist der eigentliche Editor, in dem der Quelltext bearbeitet wird. Das zweite Werkzeug bietet in Form eines Tab Strips einige Dynpros an, in denen Verwaltungsinformationen und die Schnittstellenbeschreibung gepflegt werden. Zunächst erfragt das System die Verwaltungsinformationen (siehe Abbildung 3.161).

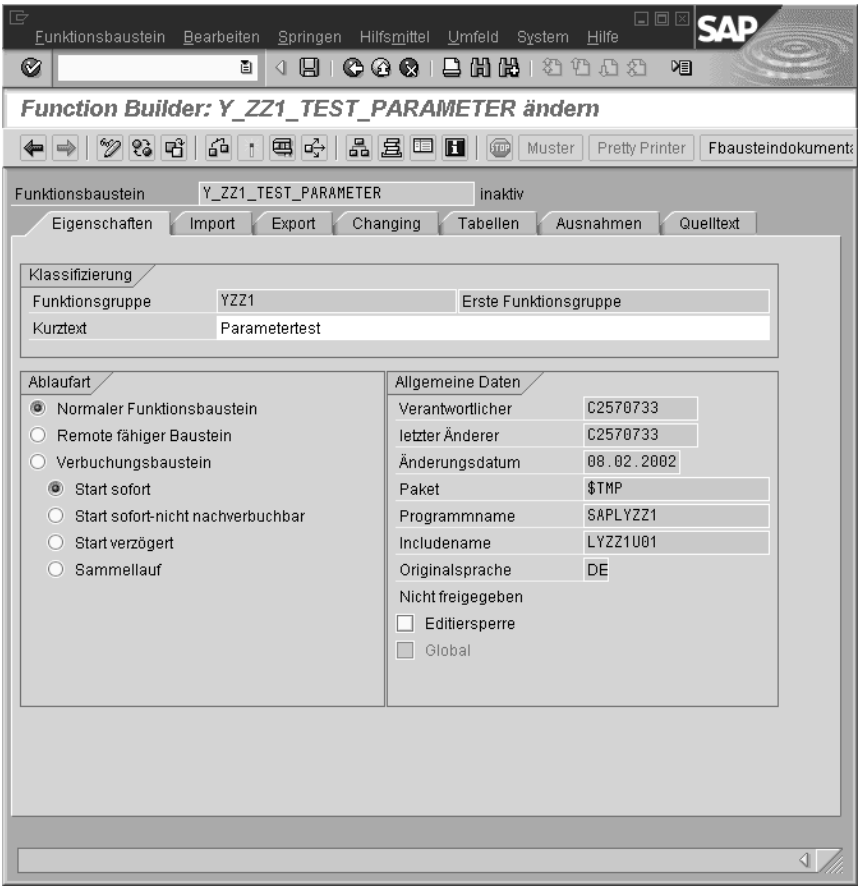

Abbildung 3.161 **© SAP AG**
Verwaltungsinformationen eines Funktionsbausteins

Diese Angaben sind die Kurzbeschreibung, die Anwendung und die Ablaufart. Die Kurzbeschreibung ist obligatorisch. Das Feld ANWENDUNG ist von untergeordneter Bedeutung. Es ist daher kein Muss-Eingabefeld und kann frei bleiben. In der Gruppe für die ABLAUFART ist der Eintrag NORMAL als Standardvorgabe markiert. Dieser Eintrag muss unbedingt gesetzt werden bzw. gesetzt bleiben. Über die Hintergründe informiert der nächste Abschnitt. Die in diesem Dynpro zu erfassenden Informationen entsprechen in ihrer Bedeutung etwa den Attributen von Reports oder Dynpros. Allerdings werden sie bei Funktionsbausteinen als Verwaltungsinformationen bezeichnet.

Die Angaben im Verwaltungs-Dynpro werden abgespeichert. Das System ergänzt einige Statusfelder im Dynpro, z. B. den Namen der Include-Datei, in welcher später der Quelltext für den neuen Funktionsbaustein abgelegt wird. Jeder

Funktionsbaustein erhält eine eigene Include-Datei. Diese Dateien sollten nie direkt, beispielsweise mit dem Standardeditor (SE38), bearbeitet werden. Insbesondere dürfen sie nicht manuell gelöscht werden! Nur die Bearbeitung von Funktionsgruppen mit dem Object Browser bzw. von Funktionsbausteinen mit der SE37 gewährleistet die Konsistenz der Funktionsgruppe! Im aktuellen Dynpro können Sie durch Auswahl der jeweiligen Registerkarten die Parameterschnittstelle oder über eine Drucktaste den Quelltext pflegen.

Im Gegensatz zu einfachen Unterprogrammen (Form-Routinen), bei denen die Schnittstelle zur Datenübergabe bei der Deklaration des Unterprogramms durch Notieren einiger Anweisungen festgelegt wird, müssen Sie die Parameter eines Funktionsbausteins in einigen speziellen Eingabemasken definieren. Die Parameterschnittstelle ist in fünf Teilbereiche gegliedert, für die jeweils eine eigene Registerkarte zur Verfügung steht. Es handelt sich zunächst um drei Eingabemasken für einfache Parameter (Felder oder Feldleisten), eine für Tabellen und eine für Ausnahmen. Im Rahmen dieses Beispiels sollen zunächst nur einfache Parameter benutzt werden. Diese sind in Import-, Export- und Changing-Parameter unterteilt. Abbildung 3.162 zeigt die Eingabemaske für Import-Parameter.

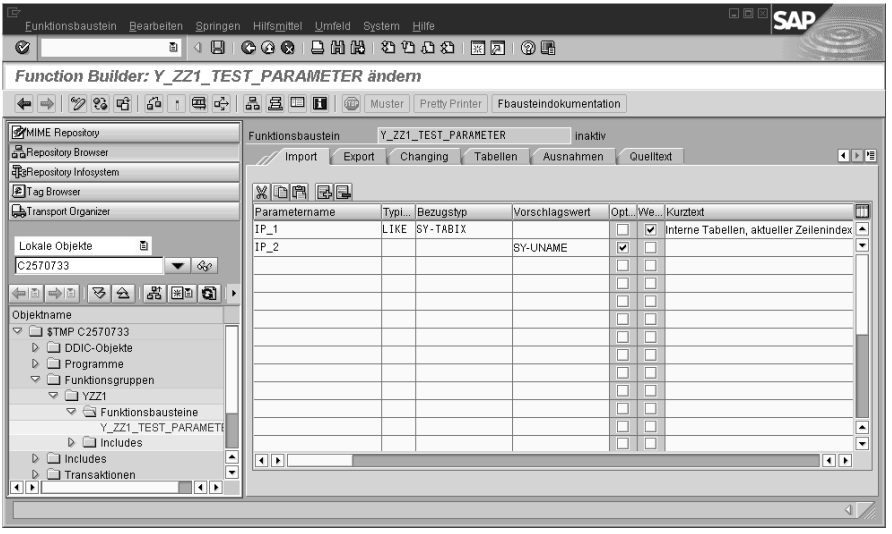

Abbildung 3.162 **© SAP AG**
Pflege der Import-Parameter eines Funktionsbausteins

Jeder Parameter wird innerhalb des Funktionsbausteins über einen eindeutigen Namen gekennzeichnet. Dieser Name wird im Feld IMPORT-PARAMETER eingetragen. Im Feld BEZUGSTYP kann der Datentyp des Parameters näher spezifiziert werden. In diesem Fall wird durch den Inhalt der Spalte TYPISIERUNG festgelegt, auf welche Weise der angegebene Bezugstyp ausgewertet werden soll. Möglich sind Bezüge mit LIKE, TYPE und TYPE REF TO. Letztere Variante dient zur Über-

gabe von Objekten und wird daher nur bei Nutzung der objektorientierten Variante von ABAP erforderlich. Die Angaben zur Typisierung sind optional, alle nicht näher deklarierten Parameter werden als CHAR-Felder behandelt. Einem Parameter kann ein Vorschlagswert (Festwert oder Systemfeld) zugewiesen werden. Falls dem Parameter beim Aufruf des Funktionsbausteins kein Wert zugewiesen wird, gilt dieser Vorgabewert. Die Schnittstelle eines Funktionsbausteins kann nachträglich erweitert werden. Damit bereits programmierte Aufrufe dieses Bausteins nicht nachträglich geändert werden müssen, können bzw. müssen die zusätzlichen Parameter mit einem Flag als optionale Parameter gekennzeichnet werden. Parameter mit Vorschlagswerten sind automatisch optionale Parameter. Für derartige Parameter ist beim Aufruf des Funktionsbausteins eine Wertübergabe möglich, aber nicht unbedingt notwendig. Das Flag WERTÜBERGABE in der letzten Spalte erzwingt die Zwischenspeicherung des übergebenen Wertes. Dieses Flag muss markiert werden, wenn Sie einen Import-Parameter innerhalb des Funktionsbausteins temporär ändern möchten.

In den drei Registerkarten sollten zum ersten Test die in Tabelle 3.44 aufgelisteten Parameter erzeugt werden.

Parametertyp	Parameter	Struktur/Feld	Typ	Vorschlag	Optional	Wertübergabe
Import	IP_1	SY-TABIX				X
Import	IP_2			SY-UNAME	X	
Changing	CP					
Export	EP_1	SY-TABIX				
Export	EP_2					

Tabelle 3.44
Parameter des neuen Funktionsbausteins

Nach Eintragen der Parameter werden die Einstellungen abgespeichert. Mit der Registerkarte QUELLTEXT wird der Editor zur Bearbeitung des Quelltextes aufgerufen. Die Bedienung des Editors entspricht im Wesentlichen der des bereits oft verwendeten Editors der SE38, allerdings unterscheiden sich einige der Pulldown-Menüs voneinander, da sie auf das jeweilige Entwicklungsobjekt bezogene Funktionen enthalten. Für den Funktionsbaustein erzeugt das System zunächst einen aus den Anweisungen FUNCTION und ENDFUNCTION bestehenden Rahmen. In diesen Rahmen wird als Kommentar die Beschreibung der Schnittstelle eingefügt. Aus den Kommentaren zur Schnittstelle gehen alle im Parameterbildschirm gesetzten Eigenschaften der Parameter hervor. Änderungen dieses Kommentars haben natürlich keinen Einfluss auf die Gestaltung der Schnittstelle.

Im Funktionsbaustein können nun einige Anweisungen notiert werden. In diesem Beispiel reichen einige Manipulationen mit und Zuweisungen zu den Parametern aus. Nachfolgend finden Sie den kompletten Quelltext des Funktionsbausteins einschließlich der automatisch erzeugten Kommentare.

```
FUNCTION y_zz1_test_parameter.
*"----------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(IP_1) LIKE  SY-TABIX
*"     REFERENCE(IP_2) DEFAULT SY-UNAME
*"  EXPORTING
*"     REFERENCE(EP_1) LIKE  SY-TABIX
*"     REFERENCE(EP_2)
*"  CHANGING
*"     REFERENCE(CP)
*"----------------------------------------------------------------

  ep_1 = ip_1 * 5.
  ep_2 = ip_2.
  cp   = cp + ep_1.

  ip_1 = 9999. " keine Auswirkung auf rufendes Programm

ENDFUNCTION.
```

In diesem Beispiel werden den Ausgabeparametern neue Werte zugewiesen. Diese neuen Werte ergeben sich direkt oder indirekt aus den Import-Parametern, so dass deren Auswertung demonstriert werden kann. Der Test des Funktionsbausteins ist ähnlich wie der Test der Reports direkt aus der Entwicklungsumgebung heraus möglich. Bevor Funktionsbausteine ausgeführt werden können, müssen sie allerdings – ähnlich wie Reports – aktiviert werden. Das kann im Normalfall für einen einzelnen Funktionsbaustein direkt aus dem Editor heraus erfolgen. Da in diesem Beispiel auch die Funktionsgruppe neu erstellt und noch nicht aktiviert wurde, müssen Sie im Repository Browser zur Objektliste der Funktionsgruppe zurückkehren und dort sowohl die Funktionsgruppe als auch den Funktionsbaustein aktivieren. Dazu setzen Sie den Cursor auf den Namen der Funktionsgruppe und lösen anschließend die Menüfunktion FUNKTIONSBAUSTEIN | AKTIVIEREN aus. Gleichberechtigt dazu können Sie auch das entsprechende Symbol oder die Tastenkombination Strg F3 verwenden.

Die Aktivierung von Funktionsbausteinen ordnet sich der allgemeinen Verfahrensweise zur Verwaltung aktiver und inaktiver Programmquellen unter. Das bedeutet, dass nach jeder Änderung eines Funktionsbausteins eine Aktivierung erfolgen muss, damit diese Änderungen wirksam werden. Darin liegt eine wesentliche Änderung gegenüber älteren Versionen, in denen noch keine inaktiven Quelltexte verwaltet werden konnten. In diesen Systemen mussten Funktions-

bausteine ebenfalls aktiviert werden, was aber eher mit einer Freigabe vergleichbar war. Diese Aktivierung musste nur einmal durchgeführt werden, Änderungen am Quelltext erforderten keine erneute Aktivierung.

Nach erfolgreicher Aktivierung kann der Funktionsbaustein getestet werden. Setzten Sie dazu in der Objektliste des Repository Browser den Cursor auf den Namen des Funktionsbausteins und rufen Sie die Menüfunktion ENTWICKLUNGSOBJEKT | AUSFÜHREN auf. Sie gelangen damit zur Testumgebung für Funktionsbausteine. Die Taste F8 bzw. das AUSFÜHREN-Symbol in der Drucktastenzeile führen zum selben Ergebnis. Aus dem Quelltext des Bausteins heraus erreichen Sie die Testumgebung mit der Menüfunktion HILFSMITTEL | TESTUMGEBUNG. Im ersten Dynpro der Testumgebung (Abbildung 3.163) werden für alle Import- und Changing-Parameter des zu testenden Bausteins Eingabefelder bereitgestellt und mit in der Schnittstelle deklarierten Vorgabewerten gefüllt. Die Breite des Eingabefelds richtet sich nach dem Datentyp. Für Parameter ohne Typangabe (hier für den Changing-Parameter `cp`) wird ein sehr breites Eingabefeld bereitgestellt. Damit weicht das System bei Parametern von Funktionsbausteinen von der Standardlänge 1 für Datenfelder ohne Typangabe ab.

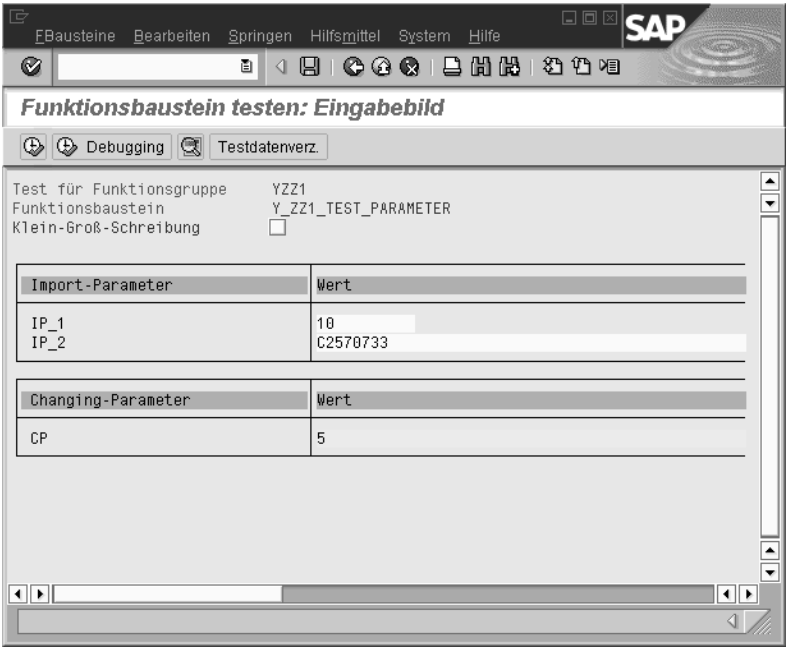

Abbildung 3.163 **© SAP AG**
Testumgebung für Funktionsbausteine: Eingabebild

Für alle mit einem Vorgabewert belegten Parameter wird dieser im Eingabebildschirm eingetragen. Diese Vorgabe kann natürlich überschrieben werden. Die Dynpros der Testumgebung nutzen die eingebettete List-Verarbeitung mit interaktiven Elementen. So kann die Oberfläche auf recht einfache Weise an die Schnittstellen der unterschiedlichsten Funktionsbausteine angepasst werden. Nach Eingabe der Testwerte wird mit der Funktionstaste F8 oder dem entsprechenden Symbol der Test eingeleitet. Nach Beendigung der Verarbeitung werden in einem zweiten Dynpro der Testumgebung die Rückgabeparameter angezeigt (siehe Abbildung 3.164). Die erfolgreiche Verarbeitung der Parameter durch den Funktionsbaustein lässt sich leicht nachvollziehen. Beachten Sie bitte, dass Sie den sichtbaren Bildausschnitt mittels der Rollbalken bis zum rechten Rand verschieben müssen, wenn Sie das Ergebnis des Changing-Parameters sehen wollen.

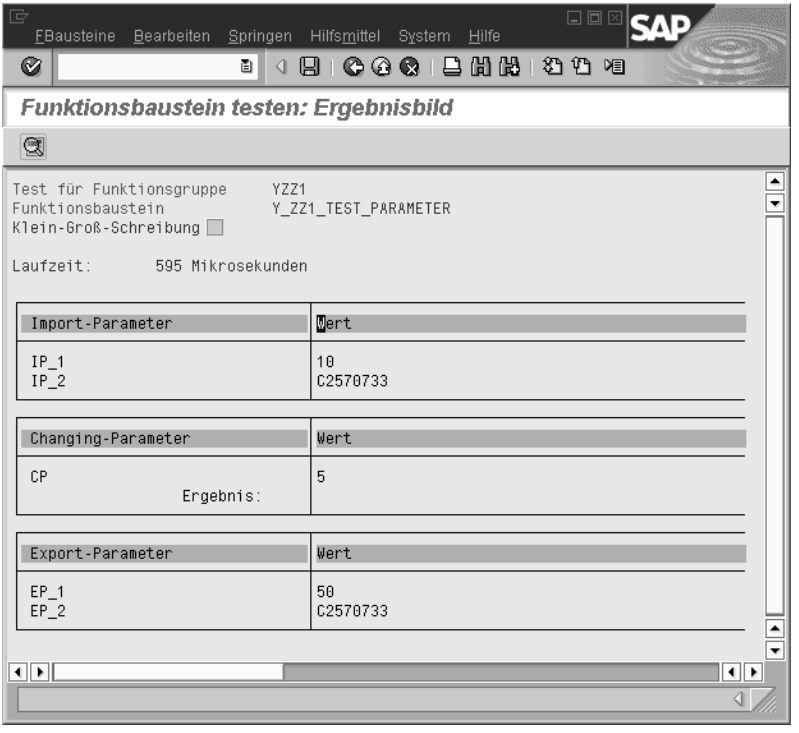

Abbildung 3.164 **© SAP AG**
Testumgebung für Funktionsbausteine: Ergebnisbild

Im Ergebnisbild erscheinen die aktuellen Werte der Parameter. Auch die beiden Export-Parameter, die im Eingabebild fehlten, erscheinen im Ergebnisbild. Innerhalb des Funktionsbausteins können die Import-Parameter zwar, wie im Beispiel demonstriert, in bestimmten Fällen neu belegt werden, dies hat aber keinen Einfluss auf die Felder im rufenden Programm. Derartige Neuzuweisungen sind nur innerhalb des Funktionsbausteins wirksam und können eingesetzt werden, um lokale Datenfelder einzusparen. Allerdings spricht dies nicht für einen guten Programmierstil.

Anstelle der Testumgebung kann natürlich auch eine kleine Anwendung dazu dienen, den eben erzeugten Funktionsbaustein aufzurufen. Besonders eignet sich dazu ein separater Report oder eines der Demoprogramme zur eingebetteten List-Verarbeitung.

```
REPORT  yz438010.
DATA: x TYPE i VALUE 2,
      y TYPE i VALUE 3,
      n TYPE i,
      s(20).

CALL FUNCTION 'Y_ZZ1_TEST_PARAMETER'
  EXPORTING
    ip_1 = x
    ip_2 = 'MEIER'
  IMPORTING
    ep_1 = n
    ep_2 = s
  CHANGING
    cp   = y.

WRITE: / n,
       / s,
       / y.
```

Zu beachten ist hier vor allem die Verwendung der Begriffe Import und Export. Daten, die ein Funktionsbaustein erhält, also importiert, müssen im rufenden Programm gesendet, also exportiert werden. Die Import-Parameter des Funktionsbausteins erscheinen beim Aufruf daher unter der Anweisung `EXPORTING`, während die im Funktionsbaustein als Export-Parameter deklarierten Parameter mit `IMPORTING` gekennzeichnet sind.

Allen formalen Parametern, die Werte an das rufende Programm zurückliefern, müssen Datenfelder als aktuelle Parameter zugewiesen werden. Alle Parameter, über die lediglich Werte an den Baustein übergeben werden, können auch mit Direktwerten versorgt werden.

3.9.2 Funktionsbausteine im Detail

Funktionsbausteine werden in ABAP-Anwendungen für völlig unterschiedliche Aufgaben eingesetzt. Entsprechend vielfältig sind die Eigenschaften, die für Funktionsbausteine gesetzt werden können. Die Programmierweise eines Bausteins muss ebenfalls dem späteren Einsatzzweck angepasst werden. Bevor einige Beispiele Programmierung und Einsatz von Funktionsbausteinen veranschaulichen, erhalten Sie in diesem Abschnitt zunächst das notwendige Grundlagenwissen.

Ablaufart

Beim Anlegen eines neuen Bausteins muss eine von mehreren verfügbaren Ablaufarten gewählt werden (siehe Abbildung 3.161, Verwaltungsinformationen eines Funktionsbausteins). Diese Funktionstypen haben entscheidenden Einfluss auf die Art der Abarbeitung eines Funktionsbausteins. Ein mit dem Typ NORMAL versehener Baustein ist die Urform von Funktionsbausteinen. Er wird aus Anwendungen heraus aufgerufen und sofort ausgeführt. Das rufende Programm wird erst fortgesetzt, wenn der Funktionsbaustein komplett abgearbeitet wurde. Bei dieser Form des Aufrufs muss sich der gerufene Funktionsbaustein innerhalb des aktuellen R/3-Systems befinden. Der so genannte *RFC-Mechanismus* (RFC, Remote Function Call) ermöglicht es, Funktionsbausteine in anderen Systemen aufzurufen und dort auszuführen. Auf diese Weise können verschiedene SAP-Systeme miteinander kommunizieren und Daten austauschen. Dabei können die beteiligten Systeme unterschiedlichen SAP-Versionen und Releaseständen angehören. Unter bestimmten Voraussetzungen ist auch die Kommunikation mit Nicht-SAP-Systemen möglich. Funktionsbausteine, die von anderen Systemen aus gerufen werden sollen, müssen den Typ REMOTE FUNCTION CALL UNTERSTÜTZT aufweisen.

Alle anderen Funktionstypen sind für so genannte *Verbucher* vorgesehen. Verbucher sind spezielle Funktionsbausteine, die sowohl bei ihrer Deklaration als auch beim späteren Aufruf spezielle Angaben erfordern. Datenbankänderungen, also das Schreiben von Daten in die Datenbank, erfordern eine gewisse Zeit. Falls viele Tabellen von den Änderungen betroffen sind und diese Änderungen weitere Aktionen auslösen (z.B. Aktualisieren von Indizes oder Matchcodes), kann dieser Zeitraum so groß werden, dass keine komfortable Dialogverarbeitung mehr möglich ist, da die Anwendung bis zum Ende des Schreibvorganges warten müsste. Komplexere Datenbankänderungen werden daher nicht synchron zum restlichen Programm ausgeführt, sondern in asynchron laufende Verbuchungsbausteine ausgelagert. Diese Bausteine werden erst ausgeführt, wenn das rufende Programm einen `COMMIT WORK` auslöst. Sie werden beim Aufruf mit dem Zusatz

```
... IN UPDATE TASK.
```

gekennzeichnet. Beim Aufruf eines so gekennzeichneten Bausteins werden die übergebenen Parameter vom System automatisch gesichert, ohne dass der Funktionsbaustein ausgeführt wird. Der Baustein kann also mehrfach mit unterschiedlichen Werten für die Parameter aufgerufen werden. Erst bei einem COMMIT WORK führt das System die Verbucher aus, wobei es ihnen die korrekten Parameter übergibt. Diese Form der Ausführung ist sehr praktisch, da die Parameterübergabe wie bei anderen Funktionsbausteinen auch erfolgen kann. Es ist sogar möglich, Verbuchungsbausteine zunächst als normale Funktionsbausteine zu entwerfen und zu testen und erst später in echte Verbucher umzuwandeln.

Da Verbucher asynchron zur rufenden Anwendung laufen, können sie keine Werte zurückliefern. Sie dürfen daher keine Export-Parameter besitzen. Bei Abarbeitungsfehlern können sie aber Fehlermeldungen (Nachrichten) auslösen, die zwar nicht in der rufenden Anwendung ausgegeben werden können, aber in diversen Fehlerprotokollen erscheinen.

Die verschiedenen Funktionstypen für Verbucher unterscheiden sich vor allem durch den Abarbeitungszeitpunkt. Verbucher der Typen VERBUCHUNG MIT START SOFORT und START SOFORT NICHT NACHVERBUCHBAR werden sofort ausgeführt. Das rufende Programm wartet dabei nicht auf die Beendigung, sondern läuft weiter. Beide Prozesse laufen also parallel. Man spricht bei dieser Form der Verbuchung auch von einer V1-Verbuchung. Da die zu verbuchenden Daten durch das System intern aufbewahrt werden, kann ein fehlgeschlagener Verbuchungsvorgang normalerweise wiederholt werden. Wenn Sie eine nochmalige Verbuchung verbieten (verhindern) möchten, dann ist die Verwendung des zweiten erwähnten Funktionstyps erforderlich.

Die letzten beiden Funktionstypen VERBUCHUNG MIT START VERZÖGERT und SAMMELLAUF definieren Verbucher, die zeitverzögert laufen. Der Startzeitpunkt kann dabei in Zeiten mit geringer Systembelastung verlagert werden. Diese Form der Verbuchung wird auch als V2-Verbuchung bezeichnet.

Bei allen asynchronen Verbuchungen sollte bedacht werden, dass der genaue Verbuchungszeitpunkt nicht vorherbestimmt werden kann. Es ist daher möglich, dass ein Datensatz von einer Transaktion geschrieben wurde, eine danach gestartete Anwendung diesen Datensatz aber nicht findet, weil der Verbucher noch nicht gelaufen ist. Derartige Auswirkungen müssen bei der Programmierung eigener Anwendungen bedacht werden, wobei oft Kompromisse bezüglich Aktualität der Daten und Performance der Anwendung erforderlich sind.

Parameter

Mittels der Parameter erfolgt der Datenaustausch zwischen Funktionsbaustein und rufender Anwendung. Dabei werden die Parameter über ihren Namen identifiziert. Beim Aufruf eines Funktionsbausteins werden die Übergabewerte (aktuelle Parameter) direkt den namentlich benannten formalen Parametern der

Schnittstelle zugewiesen. Die Reihenfolge der Parameter beim Aufruf des Funktionsbausteins ist daher bedeutungslos.

Die Parameter eines Funktionsbausteins entsprechen einer von drei Arten: Man unterscheidet einfache Datenfelder bzw. Feldleisten, Tabellen und Ausnahmen. Die einfachen Datenfelder bestehen aus den Untergruppen Import-, Export- und Changing-Parameter. Die Parameter können unterschiedliche Eigenschaften und Attribute besitzen, die als erstes beschrieben werden sollen. Allerdings sind nicht alle Attribute für alle Arten von Parametern verfügbar. Die Attribute werden im Parameterbildschirm gesetzt.

Die formalen Parameter der Schnittstelle eines Funktionsbausteins können mit einer Typbeschreibung versehen werden. In diesem Fall wird zur Laufzeit geprüft, ob der Typ bzw. die Eigenschaften des aktuellen Parameters zum formalen Parameter passen. Ist dies nicht der Fall, erzeugt die Steuerlogik einen Laufzeitfehler.

Die Parameter eines Funktionsbausteins sind normalerweise nur lokal, also innerhalb des Funktionsbausteins, gültig. Mit der Menüfunktion BEARBEITEN | SCHNITTSTELLE | PARAMETER GLOBALIS. kann die Schnittstelle eines Funktionsbausteins global für die gesamte Funktionsgruppe bekannt gemacht werden. In der Funktionsgruppe dürfen dann keine anderen globalen Daten mit identischem Namen existieren. Die Inhalte der globalen Schnittstelle eines Funktionsbausteins sind im Gegensatz zu den anderen globalen Daten nur vom Aufruf bis zum Verlassen des Bausteins bekannt. Die globale Gültigkeit einer Schnittstelle kann mit der Menüfunktion BEARBEITEN | SCHNITTSTELLE | PARAMETER LOKALIS. wieder aufgehoben werden.

Globale Schnittstellen erleichtern in bestimmten Fällen den Datenzugriff, vor allem, wenn die Funktionsgruppe Dynpros enthält, von denen aus auf Parameter eines Funktionsbausteins zugegriffen werden soll. Sie widersprechen aber den gewohnten Regeln bezüglich des Gültigkeitsbereiches von Daten und können schnell zu unübersichtlichen Programmen und damit zu unerwünschten Nebenwirkungen führen. Es ist sicherer, auf diese Schnittstellenvariante zu verzichten und stattdessen die Inhalte der Parameter in globale Datenfelder der Funktionsgruppe umzukopieren.

Parametername

Der *Parametername* dient zur Identifikation des Parameters. Er wird gemäß den Namenskonventionen für Datenfelder gebildet. Es ist mitunter der besseren Übersicht wegen üblich, aber nicht unbedingt erforderlich, durch ein Präfix auf die Art des Parameters (Import, Export, Changing, Tabelle) hinzuweisen. Da es Changing-Parameter erst ab Release 3.0 gibt, für rekursive Aufrufe allerdings Datenfelder erforderlich sind, die sowohl als Import- als auch als Export-Parameter fungieren müssen, können Im- und Export-Parameter mit identischem Namen angelegt werden. Sie müssen in diesem Fall einen übereinstimmenden Datentyp haben. Außerdem muss das Flag WERTÜBERGABE gesetzt sein.

Typisierung

Der Inhalt dieses Feldes legt fest, auf welche Weise der im Feld BEZUGSTYP eingetragene Bezeichner zur Typisierung eines Parameters benutzt werden soll. Möglich sind die Angaben `LIKE`, `TYPE` und `TYPE REF TO`.

Bezugstyp

Parameter von Funktionsbausteinen können typisiert werden. Zur Laufzeit muss dann ein Wert des angegebenen Typs als Parameter übergeben werden. Die Typisierung erfolgt entweder durch einen `LIKE`-Bezug auf ein existierendes Feld oder eine echte Typisierung mit `TYPE`. Die jeweilige Variante ist im Attribut TYPISIERUNG zu pflegen.

Ab Release 3.0 sind unter ABAP Typdeklarationen möglich. Diese können direkt in Programmen erfolgen, sie können aber auch in so genannte Typ-Pools ausgelagert werden. Diese Typ-Pools können mit der Anweisung

```
TYPE-POOL type-pool.
```

in eine Anwendung eingebunden werden. Falls dies im Rahmenprogramm einer Funktionsgruppe geschehen ist, können Parameter auch mit Verweis auf Datentypen aus diesem Typ-Pool deklariert werden.

Vorschlagswert

Einfache Parameter können mit einem Vorschlagswert versehen werden. Auf diesen Wert wird dann zurückgegriffen, wenn beim Aufruf des Funktionsbausteins in einem Programm keine explizite Zuweisung zum Parameter erfolgt. Der Vorschlagswert kann entweder eine Konstante des jeweiligen Datentyps oder eines der Systemfelder (`SY-...`) sein. Es ist zwar möglich, hier ein programminternes Feld z.B. aus dem globalen Datenbereich der Funktionsgruppe einzutragen, dies hat aber bei der Ausführung des Funktionsbausteins keinerlei Auswirkungen.

Optional

Alle Parameter, bei denen dieses Flag gesetzt wurde, müssen beim Aufruf des Funktionsbausteins nicht unbedingt mit einem Wert versorgt werden. Ihre Verwendung ist damit optional. Wird die Schnittstelle eines bereits in Programmen verwendeten Bausteins nachträglich geändert und werden die neuen Parameter als optional gekennzeichnet, dann müssen die bereits existierenden Aufrufe nicht geändert werden. Alle Parameter mit einem Vorschlagswert sind automatisch optionale Parameter. Fehlt beim Aufruf eines Bausteins die Zuweisung zu einem obligatorischen Parameter, reagiert das System mit einem Laufzeitfehler.

Wertübergabe

Herkömmliche Programmiersprachen wie C oder Pascal kennen bei der Übergabe von Parametern an ein Unterprogramm die Übergabe als Wert- oder als Referenzparameter. Bei Wertparametern wird der Wert des aktuellen Parameters in einen lokalen Datenbereich kopiert, auf den dann der formale Parameter zeigt. Falls im Funktionsbaustein der Wert des formalen Parameters geändert wird, hat dies keinen Einfluss auf den Wert des aktuellen Parameters. Bei der Übergabe per Referenz verweist der formale Parameter direkt auf den Inhalt des aktuellen Parameters, Änderungen des formalen Parameters haben damit eine direkte Auswirkung auf den aktuellen Parameter. Mittels des Flags REFERENZ kann in Funktionsbausteinen für einfache Parameter die Art der Parameterübergabe festgelegt werden. Tabellen werden immer als Referenzparameter übergeben.

Funktional notwendig sind Referenzparameter für Funktionsbausteine unter ABAP nicht. Das Konzept der Import- und Export-Parameter gewährleistet ohnehin die Übergabe von Werten an das rufende Programm. Allerdings spart die Übergabe per Referenz Speicherplatz und verbessert die Performance etwas, da kein Kopieren von Daten notwendig ist. Tabellen wurden aus diesem Grund schon immer nur als Referenz übergeben.

Eine Besonderheit ist bei Referenzparametern zu beachten, falls im Funktionsbaustein eine Ausnahme ausgelöst wird. Dies entspricht in etwa dem Auslösen einer Nachricht in Dynpros. Die Bearbeitung des Funktionsbausteins wird sofort beendet und das rufende Programm nach der Aufrufstelle fortgesetzt. Alle Wertzuweisungen zu Referenzparametern werden im Moment der Zuweisung sofort wirksam. Sie bleiben also erhalten, auch wenn nach der Zuweisung der Funktionsbaustein mit einer Ausnahme beendet wird. Die Übertragung der Wertparameter erfolgt hingegen nur bei ordnungsgemäßer Beendigung des Funktionsbausteins!

Langtext

Zu jedem Funktionsbaustein kann ein Langtext gepflegt werden. Dieser Text dient als Dokumentation für Programmierer, welche diesen Baustein benutzen möchten. Der eigentliche Text wird mit dem bereits erwähnten SAPScript-Editor gepflegt.

Nach Beschreibung der Attribute folgt eine Erläuterung der verschiedenen Parametertypen.

Import-Parameter

Import-Parameter dienen zur Übergabe von Daten an den Funktionsbaustein. Sie können im Funktionsbaustein neu belegt werden, falls es sich um Wertparameter handelt, da bei der Zuweisung zu einem Wertparameter der Wert des aktuellen Parameters nicht geändert wird. Die Zuweisung von neuen Werten zu

Import-Referenzparametern hingegen löst einen Laufzeitfehler aus, da eine solche Zuweisung den Wert des aktuellen Parameters ändern würde.

Beim Aufruf des Funktionsbausteins können den Import-Parametern sowohl Direktwerte als auch Datenfelder zugewiesen werden, unabhängig davon, ob es sich um einen Wert- oder Referenzparameter handelt. Dies ist, verglichen mit der Verfahrensweise anderer, kompilierbarer Programmiersprachen, ungewöhnlich. Dort werden Direktwerte durch den Compiler direkt in den erzeugten Maschinencode geschrieben. Für sie wird kein Speicherplatz im Datenbereich reserviert, wodurch eine Übergabe per Referenz nicht möglich ist.

Für Import-Parameter können alle oben genannten Attribute gesetzt werden.

Export-Parameter

Über die Export-Parameter liefert der Funktionsbaustein Ergebnisse an das rufende Programm zurück. Den Export-Parametern werden im Funktionsbaustein Werte zugewiesen. Nach Verarbeitung des Funktionsbausteins überträgt das System die Inhalte der Export-Parameter in die aktuellen Parameter und ändert dadurch deren Wert. Innerhalb eines Funktionsbausteins können die Export-Parameter auch gelesen und damit auf der rechten Seite von Zuweisungen verwendet werden. Dies ist aber erst dann sinnvoll, wenn dem Export-Parameter ein korrekter Wert zugewiesen wurde. Vorher ist dessen Inhalt undefiniert, denn es findet keine Datenübertragung vom aktuellen zum Export-Parameter statt.

Export-Parameter sind automatisch optionale Parameter, sie müssen also nicht ausgewertet werden. Beim Aufruf eines Funktionsbausteins können alle nicht benötigten Export-Parameter auskommentiert oder gelöscht werden. Für Export-Parameter können keine Vorgabewerte in der Schnittstellendefinition gesetzt werden.

Changing-Parameter

Die Changing-Parameter weisen die Eigenschaften von Import- und Export-Parametern auf. Sie übernehmen den Wert des aktuellen Parameters und stellen ihn für den Funktionsbaustein bereit. Falls eine Zuweisung zu einem Changing-Parameter erfolgt, wird nach der Abarbeitung des Funktionsbausteins der Wert in den aktuellen Parameter übertragen. Ebenso wie für Import-Parameter können auch für Changing-Parameter alle erwähnten Attribute verwendet werden.

Tabellen-Parameter

Über den Parametertyp TABELLEN werden interne Tabellen an den Funktionsbaustein übergeben. Diese Übergabe erfolgt stets als Referenzparameter, Änderungen des Funktionsbausteins werden also sofort in der Tabelle wirksam. Ab Release 3.0 können Tabellen als optional gekennzeichnet werden, bis zum Releasestand 2.2 war dies nicht möglich. Tabellen-Parameter müssen auf jeden Fall mit Bezug auf eine Struktur oder einen Datentyp angelegt werden.

Ausnahmen

Während der Abarbeitung eines Funktionsbausteins können unvorhergesehene Ereignisse eintreten, über die das rufende Programm unterrichtet werden muss. Beispiele dafür sind Lese- oder Schreibversuche auf Datensätze, die bereits von anderen Programmen gesperrt sind oder gar nicht existieren. Da ein Funktionsbaustein von unterschiedlichen Anwendungen in einem jeweils anderen Kontext aufgerufen werden kann, ist es in Funktionsbausteinen oft nicht möglich, sinnvoll auf solche Ereignisse zu reagieren. Fehlerzustände können daher über so genannte Ausnahmen an das rufende Programm gesendet werden. Die möglichen Ausnahmen werden in der Schnittstelle des Funktionsbausteins definiert. Es handelt sich dabei um Namen, welche die Ausnahmen beschreiben. Andere Angaben oder Attribute sind nicht erforderlich. Die Verwendung möglichst eindeutiger Bezeichner für Ausnahmen (z.B. `NOT_FOUND`, `RECORD_LOCKED`) erleichtert dem Anwender eines Funktionsbausteins die Auswertung der Ausnahmen. Die programmtechnische Behandlung von Ausnahmen erläutert der nächste Abschnitt.

Ausnahmebehandlung

Aus verschiedenen Gründen sollen Funktionsbausteine oft nicht selbst auf unerwartete Ereignisse reagieren, sondern dies dem aufrufenden Programm überlassen. Funktionsbausteine müssen das rufende Programm daher auf geeignete Weise über Fehler oder unerwartete Ereignisse informieren. Diese Information soll so transparent wie möglich sein, der Übergabemechanismus zudem flexibel. Außerdem muss sichergestellt werden, dass Ausnahmesituationen auch wirklich bearbeitet werden und nicht unentdeckt bleiben. All diese Aufgaben übernimmt die Ausnahmebehandlung, deren Funktionalität weit über die eines einfachen Rückgabewertes hinausreicht.

In der Schnittstelle eines Funktionsbausteins werden die bereits erwähnten Ausnahmen aufgeführt. Sie sind innerhalb des Funktionsbausteins als vordefinierte Bezeichner verfügbar. Die ABAP-Kommandos, welche Ausnahmen auslösen, verwenden diese Bezeichner. Beim Aufruf eines Funktionsbausteins kann der Programmierer entscheiden, welche der möglichen Ausnahmen er selbst behandelt. Dazu weist er den zu behandelnden Ausnahmen einen nummerischen Wert zu.

Wenn ein Funktionsbaustein eine Ausnahme auslöst, die im rufenden Programm behandelt werden soll, wird der Funktionsbaustein beendet und die Nummer der Ausnahme in das Feld `SY-SUBRC` gestellt. Im rufenden Programm kann dann dieses Systemfeld ausgewertet werden. Das Zuweisen einer Nummer zur Ausnahme reicht aus, um dem Funktionsbaustein zu signalisieren, dass diese Ausnahme durch das rufende Programm behandelt wird. Dieser kann aber nicht überprüfen, ob das rufende Programm auch wirklich auf die Ausnahme reagiert.

Wie ein Funktionsbaustein auf Ausnahmen reagiert, die nicht durch das rufende Programm ausgewertet werden, hängt von der Art der Auslösung im Funktionsbaustein ab. Ausnahmen müssen explizit ausgelöst werden; sie entstehen nicht automatisch beim Auftreten irgendwelcher Probleme. Dazu stehen zwei Kommandos zur Verfügung. Mit

```
RAISE exception.
```

wird eine Ausnahme ausgelöst. Falls das rufende Programm diese Ausnahme behandelt, verläuft die Behandlung wie beschrieben. Wenn die Ausnahme allerdings nicht behandelt wird, erzeugt das System einen Laufzeitfehler. Etwas weniger rigoros ist die nächste Variante der Ausnahmeerzeugung. Dazu erhält die Anweisung MESSAGE einen Zusatz:

```
MESSAGE ... RAISING exception.
```

Auf eine nicht behandelte Ausnahme reagiert der Funktionsbaustein nicht mehr mit einem Laufzeitfehler, sondern mit dem Aussenden der in der MESSAGE-Anweisung angegebenen Nachricht. Für eine vom rufenden Programm behandelte Ausnahme läuft hingegen der bereits geschilderte Vorgang ab. Allerdings werden neben SY-SUBRC einige weitere Systemfelder gefüllt. Die für die MESSAGE-Anweisung relevante Nachrichtenklasse wird in das Feld SY-MSGID übertragen. Analog dazu werden die Felder SY-MSGTY mit dem Nachrichtentyp und SY-MSGNO mit der eigentlichen Nachrichtennummer gefüllt. Einer MESSAGE-RAISING-Anweisung können mit dem Zusatz WITH bis zu vier Parameter übergeben werden, die anstelle einiger Platzhalter im Fehlertext erscheinen. Diese Parameter werden, falls vorhanden, in die Systemfelder SY-MSGV1 bis SY-MSGV4 übertragen. Auf diese Weise verfügt das rufende Programm über alle Informationen, um die Fehlermeldung gegebenenfalls selbst aufrufen zu können.

Es ist nicht notwendig, beim Aufruf eines Funktionsbausteins den Ausnahmen einen eindeutigen Wert zuzuweisen. Mehrere Ausnahmen können ein und denselben Wert erhalten. Nach dem Auslösen einer dieser Ausnahmen kann das rufende Programm dann aber nicht mehr unterschieden, welche Ausnahme im Funktionsbaustein ausgelöst wurde. Wenn diese Unterscheidung nicht erforderlich ist, reicht es auch aus, die zu behandelnden Ausnahmen ohne Wertzuweisung hinter EXCEPTIONS zu notieren. Der später für diese Ausnahmen zurückgegebene Wert ist unbestimmt, aber ungleich 0. Alle nicht zu behandelnden Ausnahmen erhalten keinen Wert. Sie müssen entweder auskommentiert oder aus dem Aufruf gelöscht werden. Ähnlich arbeitet der Zusatz OTHERS. Mit dieser vom System vordefinierten Ausnahme werden beim Aufruf alle nicht einzeln aufgeführten Ausnahmen eines Funktionsbausteins benannt. Auch diese Ausnahme kann mit und ohne Wertzuweisung benutzt werden.

Mitunter ist es unerwünscht, dass Funktionsbausteine selbst Nachrichten mit MESSAGE auslösen. Dies kann unterbunden werden, auch wenn das Aussenden einer Nachricht im Funktionsbaustein durch eine einfache MESSAGE-Anweisung ohne den Zusatz RAISING erfolgt. Dazu wird beim Aufruf des Bausteins die

ebenfalls vordefinierte Ausnahme ERROR_MESSAGE verwendet. Wenn sie gesetzt ist, führen alle E- und A-Nachrichten im Funktionsbaustein automatisch zum Auslösen der Ausnahme ERROR_MESSAGE. Alle anderen Nachrichten werden unterdrückt.

Das folgende Programm soll die Verwendung der unterschiedlichen Ausnahmen detaillierter demonstrieren. Zunächst wird in der bereits vorhandenen Funktionsgruppe ein neuer Funktionsbaustein mit dem Funktionstyp NORMAL erzeugt. Er ist relativ einfach. Der Aufbau der Schnittstelle geht aus dem Kommentar im Quelltext hervor. Als Eingabeparameter dient ein Parameter mit Namen A, der vom Feld SY-TABIX abgeleitet wird.

Die Ausnahmen werden innerhalb der Schnittstellendefinition in einer eigenen Registerkarte gepflegt. In dieser Maske steht nur eine einzige Eingabespalte zur Verfügung, in der die Ausnahmen aufgelistet werden.

Der Quelltext des zweiten Bausteins ist relativ einfach. Abhängig vom Wert der Eingabeparameters werden verschiedene Ausnahmen mit unterschiedlichen Varianten des MESSAGE-Kommandos ausgelöst.

```
FUNCTION y_zz1_test_exception.
*"----------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(A) LIKE  SY-TABIX
*"  EXCEPTIONS
*"      A1
*"      A2
*"      A3
*"----------------------------------------------------------------
  CASE a.
    WHEN 1.
      RAISE a1.
    WHEN 2.
      MESSAGE ID 'YZ4' TYPE 'I' NUMBER '002' RAISING a2.
    WHEN 3.
      MESSAGE ID 'YZ4' TYPE 'E' NUMBER '003' RAISING a3.
    WHEN 4.
      MESSAGE ID 'YZ4' TYPE 'I' NUMBER '000'.
    WHEN 5.
      MESSAGE ID 'YZ4' TYPE 'E' NUMBER '000'.
  ENDCASE.
ENDFUNCTION.
```

Im folgenden Report wird der Baustein mit mehreren Varianten der Ausnahmebehandlung aufgerufen. Die Nummer der auszulösenden Ausnahme wird dem Rückgabewert SY-SUBRC gegenübergestellt.

```
REPORT  yz438020.
* gemeinsames Abfangen mehrerer Ausnahmen
WRITE: / 'Gemeinsames Abfangen' COLOR 5.
DO 3 TIMES.
  CALL FUNCTION 'Y_ZZ1_TEST_EXCEPTION'
       EXPORTING
            a  = sy-index
       EXCEPTIONS
            a1
            a2
            a3.

  WRITE: / 'Ausnahme:', (4)sy-index, 'SY-SUBRC',
            (4)sy-subrc.
ENDDO.

* selektives Abfangen einer Ausnahme + Message im FB
WRITE: /, / 'Selektives Abfangen' COLOR 5.
DO 4 TIMES.
  CALL FUNCTION 'Y_ZZ1_TEST_EXCEPTION'
    EXPORTING
      a  = sy-index
    EXCEPTIONS
      a1 = 11
      a2 = 12
      a3 = 13.

  WRITE: / 'Ausnahme:', (4)sy-index, 'SY-SUBRC', (4)sy-subrc.
ENDDO.

* Verwendung von OTHERS und ERROR-MESSAGE
WRITE: /, / 'OTHERS und ERROR_MESSAGE' COLOR 5.
DO 5 TIMES.
  CALL FUNCTION 'Y_ZZ1_TEST_EXCEPTION'
    EXPORTING
      a             = sy-index
    EXCEPTIONS
      a2            = 1
      error_message = 33
      OTHERS        = 99.

  WRITE: / 'Ausnahme:', (4)sy-index, 'SY-SUBRC', (4)sy-subrc.
ENDDO.
```

In der ersten `DO`-Schleife werden nacheinander die drei echten Ausnahmen und die Nachricht mit dem Typ `I` ausgelöst. Die drei Ausnahmen werden durch das rufende Programm zwar behandelt, allerdings wird keine individuelle Num-

mer für die Ausnahmen vergeben. Die Ausgaben im Report zeigen, dass für alle drei Ausnahmen in SY-SUBRC ein Wert von 1 zurückgeliefert wird. Auf diese Weise kann ermittelt werden, dass im Funktionsbaustein eine Ausnahme erzeugt wurde, aber nicht welche. Erst die Zuweisung von Werten zu den Ausnahmen ermöglicht diese Unterscheidung, wie die zweite Schleife zeigt. Die zugewiesenen Werte sind beliebig. Normalerweise werden jedoch laufende Nummern eingesetzt und nicht – wie im Beispiel – willkürliche Werte. Innerhalb der Schleife gibt der Funktionsbaustein noch eine Nachricht aus. Diese erscheint wegen des Typs I als Popup, das bestätigt werden muss. Dieser Nachrichtentyp bewirkt keinen Programmabbruch. Als Rückgabewert erscheint im Report daher 0.

Die Anweisungen der dritten Schleife demonstrieren die Zusätze OTHERS und ERROR_MESSAGE. Die zweite Ausnahme wird individuell behandelt. Für sie erscheint im Report der Rückgabewert 1. Die Anweisung OTHERS weist allen nicht explizit aufgeführten Ausnahmen, hier also A1 und A3, den Wert 99 zu. Das Aussenden von Fehlernachrichten aller Typen wird durch ERROR_MESSAGE verhindert. Es erscheint weder das Info-Popup noch die mit dem Parameter 5 ausgelöste echte Fehlernachricht. Allerdings stellt Letztere den Wert der Ausnahme ERROR_MESSAGE in das Feld SY-SUBRC.

Einige weitere Merkmale der Ausnahmebehandlung können nicht im obigen Beispiel vorgeführt werden, da sie zu einem Programmabbruch führen. Dies geschieht zum Beispiel dann, wenn Ausnahmen ausgelöst werden, die das rufende Programm nicht abfängt. Rufen Sie daher den Funktionsbaustein in einem weiteren Report, den Sie selbst anlegen, in folgender Form auf. Vor jedem Start des Reports ist er abzuspeichern!

```
CALL FUNCTION 'YZ31_TEST_EXCEPTION'
     EXPORTING
         a  = 1.  " Nacheinander 1 bis 3 einsetzen
```

Der Aufruf von Funktionsbausteinen

Funktionsbausteine werden mit der Anweisung

```
CALL function ...
```

aufgerufen. Der Name des Funktionsbausteins ist in einfache Anführungsstriche einzuschließen. Er ist in Großbuchstaben einzutragen, andernfalls wird der Funktionsbaustein nicht gefunden! In der Anweisung müssen alle zu übergebenden Parameter aufgeführt werden. Außerdem ist die Bezeichnung der einzelnen Parametergruppen mit den Anweisungen EXPORTING, IMPORTING, CHANGING, TABLES und EXCEPTIONS erforderlich. Da die Schnittstellen von Funktionsbausteinen sehr umfangreich sein können, stellt der ABAP-Programmeditor eine Hilfsfunktion zum Einfügen der gesamten Anweisung bereit. Mit der Menüfunktion BEARBEITEN | MUSTER kann eine Hilfsfunktion aufgerufen werden, die einige

komplexe Anweisungen in den Quelltext einfügen kann. In einem Popup (Abbildung 3.165) erfragt sie zunächst Daten zum einzufügenden Objekt.

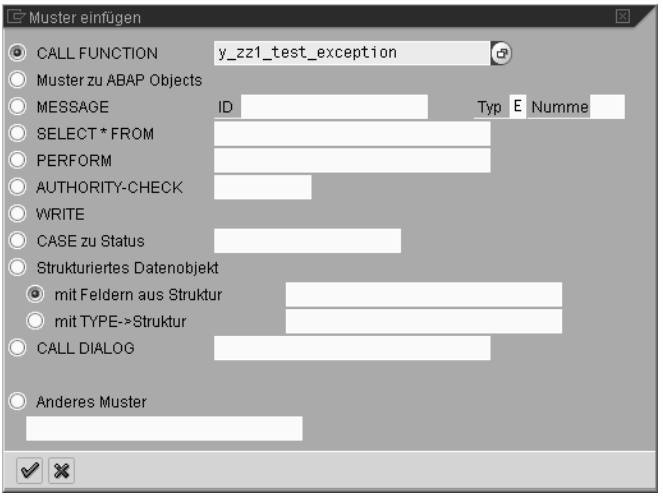

Abbildung 3.165 **© SAP AG**
Popup zum Einfügen eines Funktionsbaustein-Aufrufs in ein Programm

In diesem Popup markieren Sie das Auswahlfeld CALL FUNCTION. Im zugehörigen Eingabefeld müssen Sie den Namen des einzufügenden Bausteins eintragen. Falls die korrekte Schreibweise nicht bekannt ist, steht eine Eingabehilfe bereit, die Sie über die Taste F4 aufrufen können.

Nach Bestätigung der Eingabe mit der ↵-Taste oder dem entsprechenden Symbol fügt das System ein vollständiges Kommando zum Aufruf des Bausteins in den Quelltext ein. Dieser Aufruf enthält alle Parameter des Funktionsbausteins. Jeder der Parameter steht in einer eigenen Zeile. Alle optionalen Parameter werden zwar in den Quelltext übernommen, aber auskommentiert. Sie sind bei Bedarf durch Entfernen des Kommentarzeichens und Zuweisen eines aktuellen Parameters zu aktivieren. Bei Parametern mit Vorgabewerten werden diese in die Zuweisung eingetragen. Da solche Parameter automatisch optional sind, werden sie natürlich auch auskommentiert. Die Aufnahme in den Aufruf ist nur dann erforderlich, wenn ein vom Vorgabewert abweichender Wert übergeben werden soll. Auskommentierte Zeilen können im Quelltext verbleiben. Dadurch sind spätere Änderungen durch Entfernen des Kommentarzeichens und Zuweisen eines Wertes sehr einfach möglich.

Unbedingt erforderliche Parameter stehen in echten, nicht als Kommentar gekennzeichneten Anweisungszeilen. Da die Zuweisung eines Wertes aber noch fehlt, würde beim Start eines solchen Programms oder beim separaten Syntaxcheck ein Fehler angezeigt.

Alle im Funktionsbaustein deklarierten Ausnahmen stehen ebenfalls im automatisch generierten Aufruf. Das System versieht sie mit einer laufenden Nummer. Falls eine Ausnahme nicht behandelt werden soll, kann sie auskommentiert werden. Die zugewiesenen Nummern kann der Programmierer problemlos ändern.

Durch das automatische Auskommentieren ergeben sich mitunter kleine Fehler. Wenn in einer der Parametergruppen alle Parameter optional sind, muss die jeweilige Gruppenbezeichnung (z.B. `EXPORTING` oder `TABLES`) manuell auskommentiert werden, da eine solche Anweisung ohne nachfolgende Parameter zu einem Syntaxfehler führt.

Neben dem einfachen Aufruf, wie er in den bisherigen Beispielen erfolgte, existieren für die Anweisung `CALL FUNCTION` noch einige Zusätze, die spezielle Funktionstypen aufrufen bzw. für eine besondere Ausführungsart des aufgerufenen Funktionsbausteins sorgen. An dieser Stelle soll nur der Zusatz `IN UPDATE TASK` erwähnt werden. Er ist erforderlich, um Verbuchungsfunktionsbausteine aufzurufen.

Dokumentation

Da Funktionsbausteine oft für den anwendungsübergreifenden Einsatz entworfen werden, kommt der Dokumentation des Bausteins selbst, der Funktionsgruppe als Ganzes und den einzelnen Parameter eine erhebliche Bedeutung zu. Ein potenzieller Anwender soll aus den Dokumentationen zu den erwähnten Bestandteilen ausreichend Informationen zur Anwendung des Funktionsbausteins erhalten und nicht den Quelltext analysieren müssen. Alle Funktionsbausteine, die zur übergreifenden Benutzung gedacht sind, müssen daher gründlich dokumentiert werden. Die Werkzeuge dazu können Sie an mehreren Stellen der Entwicklungsumgebung aufrufen. In diesem Abschnitt steht wieder der Object Browser im Mittelpunkt.

Die Dokumentation einer Funktionsgruppe kann von deren Objektliste aus erfolgen. Ein Doppelklick auf den Namen der Funktionsgruppe ruft ein Popup auf, in dem einige wichtige Eigenschaften der Funktionsgruppe gepflegt werden können. Zu diesen Eigenschaften gehört auch die Dokumentation, die über die Drucktaste am unteren Rand des Popups erreichbar ist (Abbildung 3.166).

Das System ruft nun einen speziellen Editor (SAP-Script-Editor, Abbildung 3.167) auf, in dem Sie eine ausführliche Dokumentation erfassen können. Im Gegensatz zum SAP-Script-Editor früherer Versionen ähnelt die Bedienung dieses Editors den üblichen Windows-Anwendungen.

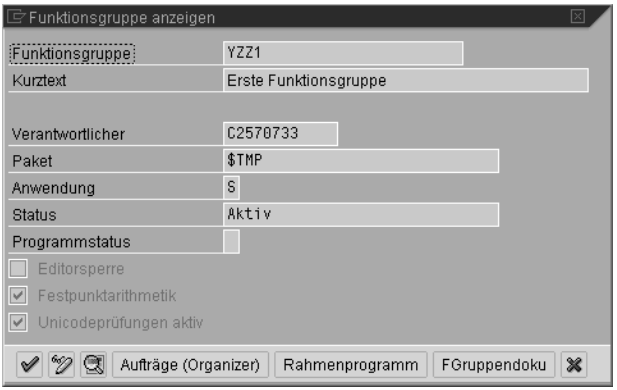

Abbildung 3.166 **© SAP AG**
Editor zur Pflege der Dokumentation einer Funktionsgruppe

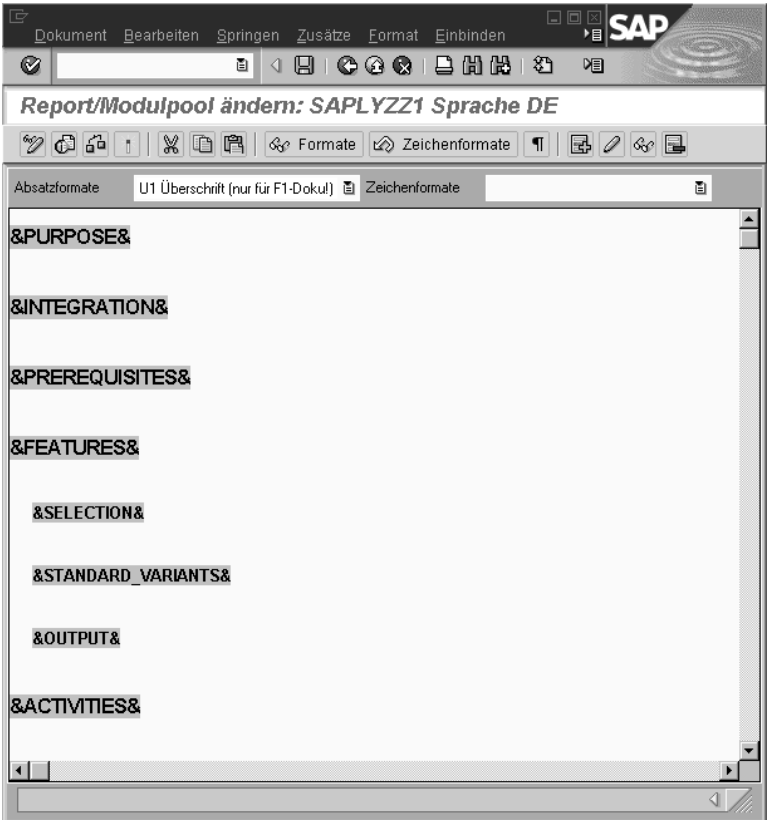

Abbildung 3.167 **© SAP AG**
Dokumentation der Funktionsgruppe bearbeiten

Auch die einzelnen Funktionsbausteine können eine separate Dokumentation erhalten. Diese wird über eine Registerkarte im Tab Strip des Verwaltungs-Dynpros aufgerufen. In dieser Eingabemaske kann jedem Parameter und jeder Ausnahme ein kurzer beschreibender Text zugewiesen werden. Außerdem können Sie den Kurztext zum Funktionsbaustein pflegen. Ein Doppelklick auf eines der Eingabefelder führt wieder zum SAP-Script-Editor, mit dem ein längerer Text erfasst werden kann.

3.9.3 Dialogbausteine

Dialogbausteine ähneln gewöhnlichen Dialogtransaktionen, die gekapselt wurden und nur noch über eine genau definierte Schnittstelle zum Datenaustausch erreichbar sind. Insofern entsprechen sie den Funktionsbausteinen. Anstelle des Function-Pools werden ein oder mehrere Dialogbausteine einem Modul-Pool zugeordnet. Dieser Modul-Pool enthält dieselben Elemente wie jener einer gewöhnlichen Dialoganwendung, also eine Oberfläche, Dynpros und Module. Allerdings existiert keine Analogie zum Begriff Funktionsgruppe.

Abgesehen von der Definition der Schnittstelle des Dialogbausteins entspricht die Programmierung der einer normalen Dialoganwendung. Zunächst sollte daher ein Modul-Pool erzeugt werden. Auch die Programmierung der Funktionalität, also das Anlegen von Dynpros und der dazugehörigen Module, ist möglich. Anschließend kann vom Grundbild der Workbench aus mit ENTWICKLUNG | PROGRAMMIERUMFELD | DIALOGBAUSTEINE oder dem Transaktionscode SE35 das Pflegewerkzeug für Dialogbausteine aufgerufen werden. Dort wird der Name erfasst und mittels der Drucktaste ANLEGEN der Baustein erzeugt. Im nachfolgenden Dynpro sind die obligatorische Kurzbeschreibung, der Name des Modul-Pools und die Nummer des Startdynpros einzutragen. In weiteren Dynpros wird die Schnittstelle definiert. Dieser Vorgang entspricht dem Erzeugen der Schnittstelle von Funktionsbausteinen, allerdings ist die Schnittstelle von Dialogbausteinen einfacher gehalten.

Genau genommen stellen Dialogbausteine nur eine exakt definierte Aufrufmöglichkeit für Dynpros dar, die auch auf andere Weise, z. B. mit `CALL SCREEN`, ausgeführt werden könnten. Sie werden daher wesentlich seltener eingesetzt als Funktionsbausteine.

3.9.4 BAPIs

Der Zugriff externer Programme auf die Ressourcen des R/3-Systems gestalten sich relativ kompliziert, obwohl mit dem RFC-Aufruf von Funktionsbausteinen eine einfache technische Lösung existiert. Der Grund liegt in der Komplexität des R/3-Systems und der Tatsache, dass viele der Anwendungen in sich abgeschlossen sind. Bedingt durch das Dynpro-Konzept sind die innere Funktionalität und die Visualisierung der Anwendung untrennbar miteinander ver-

knüpft. Es existieren kaum klar definierte Schnittstellen, über die eine externe Anwendung spezifische Ressourcen einer Anwendung nutzen kann. Für einige Programmiertechniken, z.B. einige Internet-Anbindungen oder die Ankopplung von Programmen anderer Hersteller, sind derartige Schnittstellen aber unbedingt erforderlich. Ab der Version 3.1 stellt SAP so genannte BAPIs (Business Application Component Interfaces) bereit. BAPIs sind technisch gesehen nichts weiter als RFC-fähige Funktionsbausteine. Sie sollen dialoglose, auf so genannte Business-Objekte bezogene Anwendungsfunktionalität bereitstellen. Ziel des Einsatzes der BAPIs ist es unter anderem, die visuelle Oberfläche getrennt von der Anwendungsfunktionalität zu entwickeln. BAPIs können sowohl innerhalb des R/3-Systems in ABAP-Anwendungen als auch von externen Anwendungen genutzt werden. Aus verschiedenen Gründen müssen BAPIs im Business Object Repository als Methoden eines Business Objects definiert werden.

Da die Entwicklung von BAPIs bei SAP in vollem Gange ist, sei an dieser Stelle bezüglich des aktuellen Entwicklungsstandes und der Dokumentation an die SAP verwiesen. Die jeweils aktuelle Beschreibung finden sie auf dem WWW-Server der SAP-AG.

Damit BAPIs im erwähnten Sinne eingesetzt werden können, müssen sie einigen Bedingungen genügen. Diese Anforderungen haben auch Auswirkung auf die Anwendungen, die mit Hilfe der BAPIs programmiert werden. Sie sollen hier stichpunktartig beschrieben werden.

Statuslosigkeit

Die BAPIs sollen eine in sich abgeschlossene Aufgabe nach dem Prinzip „Alles oder nichts" ausführen. Sie sollen weder von der vorherigen Ausführung anderer BAPIs abhängig sein noch einen temporären Systemzustand hinterlassen, der Voraussetzung für die Abarbeitung anderer BAPIs ist. BAPIs dürfen daher auch keine Datenfelder im globalen Datenbereich ihrer Funktionsgruppe ändern.

In der Praxis bringt das einige Probleme mit sich. Herkömmliche Anwendungen lesen beispielsweise einen Datensatz und sperren ihn gleichzeitig für die Bearbeitung durch Dritte. Der Datensatz wird durch den Anwender modifiziert und in die Tabelle zurückgeschrieben. Dabei wird die Sperre aufgehoben. Bei dieser Form der Verarbeitung können unter normalen Voraussetzungen keine Probleme auftreten. Da BAPIs keinen Dialog besitzen, kann die eben geschilderte Aufgabe nicht durch einen einzigen BAPI übernommen werden. Es werden zwei BAPI-Aufrufe erforderlich, ein lesender und ein schreibender. Da wegen der geforderten Statuslosigkeit keine Datenbanksperren gesetzt werden dürfen, ist nicht sichergestellt, dass der schreibende Zugriff auf die Datenbank wirklich ausgeführt werden darf. Möglicherweise wird derselbe Datensatz gleichzeitig durch eine andere Anwendung bearbeitet. Denkbar ist auch, dass er durch eine herkömmliche Anwendung momentan gesperrt ist. Dies bedingt entweder einen Verzicht auf Funktionalität, eine völlig andere Programmierweise oder aber Kompromisse bezüglich der Einhaltung der Forderung zur Statuslosigkeit.

Unveränderbarkeit

Im Gegensatz zu den üblichen R/3-Transaktionen, die von Release zu Release teilweise starken Änderungen unterliegen, müssen BAPIs eine über mehrere Releasezyklen hinweg konstante Schnittstelle bieten. Das gilt sowohl für die eigentliche technische Schnittstelle als auch für die Funktionalität. Andernfalls müssten Anwendungen, die BAPIs verwenden, nach jedem Upgrade geprüft und angepasst werden.

3.10 Allgemein verwendbare Funktionsbausteine

Viele der im R/3-System existierenden Funktionsbausteine erfüllen anwendungsübergreifende Aufgaben. Sie entsprechen damit einer Standard-Bibliothek, die in vielen anderen Programmiersprachen zur Verfügung stehen. Einige wichtige Funktionen innerhalb des R/3-Systems werden nicht durch spezielle ABAP-Befehle, sondern durch Funktionsbausteine ausgeführt. Dieser Abschnitt beschreibt einige der wichtigsten Programmiertechniken, die auf dem Einsatz von Funktionsbausteinen beruhen.

Die bisher beschriebenen Beispiele dienten lediglich zur Demonstration einiger Eigenschaften von Funktionsbausteinen. In diesem Abschnitt sollen einige praxisnahe Anwendungsbeispiele für den Einsatz von Funktionsbausteinen vorgestellt werden.

3.10.1 Standardisierte Dialoge

In einem Programm werden sehr häufig diverse Standardabfragen erforderlich. In der Funktionsbibliothek des SAP-Systems sind bereits eine große Anzahl derartiger Dialoge vorhanden. Sie entsprechen den Anforderungen des SAP Style Guide. Die Funktionsbausteine können und sollen in eigenen Anwendungen benutzt werden, um dem Anwender des SAP-Systems eine einheitliche Oberfläche anzubieten. Obwohl die Abfragen letztlich durch ein Dynpro gebildet werden, bietet die Realisierung als Funktionsbaustein eine größere Flexibilität. Die zur Verfügung stehenden Funktionsbausteine werden in verschiedene Aufgabenbereiche eingeteilt. Sie gehören zudem zu unterschiedlichen Funktionsgruppen (`SPO1` bis `SPO6` und `STAB`). Folgende Bereiche werden unterschieden:

- Sicherheitsabfragen,
- Auswahl von Alternativen,
- Eingeben von Daten,
- Dialoge zum Drucken,
- Anzeigen von Text,
- Anzeigen von Tabellen.

Der einfachste der Bausteine ist POPUP_TO_CONFIRM_LOSS_OF_DATA. Die Verwendung dieses Bausteins bietet sich beispielsweise an, wenn die Bearbeitung eines Dynpros in einem Exit-Modul ohne Datensicherung abgebrochen werden soll. Mit derartigen Abfragen dürften Sie während Ihrer bisherigen Arbeit oft in Berührung gekommen sein. Der Funktionsbaustein ist sehr einfach anzuwenden. Das Listing zeigt einen Ausschnitt aus einem realen Programm.

```
...
IF ( ( flg_action = c_yes ) OR ( sy-datar = 'X' ) ).
  CALL FUNCTION 'POPUP_TO_CONFIRM_LOSS_OF_DATA'
    EXPORTING
      textline1    = text-015          " Wollen Sie wirklich
                                       " abbrechen?
"     textline2    = ' '
      titel        = text-021          " Achtung!
"     start_column = 25
"     start_row    = 6
    IMPORTING
      answer       = l_answer.

  IF l_answer = 'J'.
    LEAVE TO SCREEN 0.
  ENDIF.
ELSE.
  LEAVE TO SCREEN 0.
ENDIF.
...
```

Aus diesem Beispiel wird die Anwendung des Bausteins ersichtlich. Drei der Parameter sind zwingend erforderlich. Dabei handelt es sich um einen Text (textline1), den Titel, der im Kopf des Popups erscheint (titel), sowie den Rückgabeparameter (answer). Über den Parameter answer liefert der Baustein die Information zurück, welche der beiden Drucktasten im Popup betätigt wurde. Dieser Wert ist entweder „J“ oder „N“. Im Baustein wird die Antwort NEIN voreingestellt, so dass versehentliches Bestätigen des Popups mit der ↵-Taste nicht zu einem Programmabbruch führt. Die Analyse dieses sehr einfachen Bausteins, sowohl des Quelltextes als auch der anderen Bestandteile wie Schnittstellendefinition oder Dokumentation, liefert interessante Einblicke in die Programmierung und festigt die vermittelten Kenntnisse.

Einige andere Funktionsbausteine stellen wesentlich komplexere Hilfsmittel bereit, erfordern aber auch eine etwas aufwändigere Programmierung. Das nachfolgende, deutlich kompliziertere Programm verwendet zwei Funktionsbausteine zur Eingabe von Werten und zur Auswahl von Datensätzen aus einer Tabelle. Diese Funktionsbausteine ermöglichen den dynamischen Aufbau eines Popups zur Laufzeit eines Programms, also ohne den Einsatz des Screen Painter. Zur einfachen Demonstration werden die Anweisungen zunächst als Unterpro-

gramm in einen Report eingebunden. Dieses Unterprogramm ist unverändert als Eingabehilfe zum Zeitpunkt `PROCESS ON VALUE-REQUEST` verwendbar. Dazu wird das Unterprogramm einfach von einem POV-Modul aus aufgerufen. Die Ausgabe des selektierten Wertes wird dann durch eine Zuweisung zu einem Dynpro-Feld ersetzt.

Die Eingabehilfe soll zunächst ein Popup anbieten, in dem je ein Muster für den Namen eines Benutzers und eines Entwicklungsobjekts eingegeben werden können. Entwicklungsobjekte sind z.B. Programme, Tabellen oder Modul-Pools. Nach Erfassen dieser beiden Werte liest die Anwendung alle Objekte aus der Tabelle `TADIR`, die den eingetragenen Selektionskriterien entsprechen und listet sie in einem zweiten Popup in Tabellenform auf. In diesem Popup kann ein Eintrag per Doppelklick selektiert werden. Dieser wird im eigentlichen Report zur Kontrolle ausgegeben.

Wegen seines Umfangs wird das Listing zur Verdeutlichung in mehrere Abschnitte geteilt, zwischen denen Sie Erläuterungen zum jeweiligen Quelltextausschnitt finden. Zunächst das eigentliche Hauptprogramm: Es besteht lediglich aus einigen Deklarationen, dem Aufruf des Unterprogramms und der Ausgabe des selektierten Eintrages.

```
REPORT  yz337030 NO STANDARD PAGE HEADING.
TABLES tadir.

DATA: name  LIKE tadir-obj_name,
      subrc LIKE sy-subrc.

PERFORM select_entry
  USING name subrc.

IF subrc = 0.
  WRITE: / 'Auswahl:',  name.
ELSE.
  WRITE: / 'ERROR'.
ENDIF.
```

Der Selektionsvorgang findet in einem Unterprogramm statt.

```
FORM select_entry USING p_name LIKE tadir-obj_name
                        p_subrc LIKE sy-subrc.

*...Felder zur Eingabe der Suchbegriffe
  DATA: l_name LIKE tadir-obj_name,  " Name Entwicklungsobjekt
        l_author LIKE tadir-author,  " Eigentümer des Objekts
        l_returncode LIKE sy-tcode,  " Rückgabewert FB
        l_field_len TYPE i,          " Hilfsfeld Feldlänge
        l_lines TYPE i.              " Hilfsfeld Zeilenzähler
```

Zu Beginn des Unterprogramms sind einige Datenfelder zu deklarieren. Die auszuwertende Tabelle TADIR wurde bereits im Hauptprogramm bekannt gemacht. Die Namen der Hilfsfelder beginnen alle mit dem Präfix L_. Dieser Präfix soll lokale Felder kennzeichnen. ABAP-Programme, vor allem dialogorientierte Anwendungen, machen aber intensiv Gebrauch von globalen Feldern. Diese globalen Felder werden sehr oft auch unter Umgehung von Schnittstellen in Unterprogrammen gelesen oder gar beschrieben. Eine Unterscheidung von lokal und global gültigen Feldern kann die Übersichtlichkeit in einer Anwendung erheblich steigern.

```
*..Makro zur Deklaration interner Tabellen
  DEFINE defitab.
    data: begin of &1 occurs &3.
            include structure &2.
    data: end of &1.
  END-OF-DEFINITION.

*...Tabelle für Eingabefelder
  defitab i_inf helpval 10.

*...Tabelle für Feldbezeichner des Listen-Popups
  defitab i_tabf help_value 10.

*...Tabelle für Datenzeilen des Listen-Popups
  DATA: BEGIN OF i_values OCCURS 100,
    lines(256) TYPE c,
  END OF i_values.
```

Neben den einfachen Datenfeldern sind drei verschiedene interne Tabellen erforderlich, mit denen die erwähnten Funktionsbausteine mit Daten versorgt werden. Zwei dieser Tabellen werden von Strukturen des Data Dictionary (HELPVAL und HELP_VALUE) abgeleitet. Diese Strukturen müssen nicht per TABLES deklariert werden, sofern sie nur in der INCLUDE STRUCTURE-Anweisung benutzt werden. Die Namen der internen Tabellen erhalten zur eindeutigen Kennzeichnung den Präfix I_. In der Tabelle i_inf wird die Beschreibung der Eingabefelder des ersten Popups aufgebaut. Der entsprechende Funktionsbaustein liefert in dieser Tabelle die eingelesenen Werte zurück. Die beiden anderen Tabellen definieren die Struktur des Listen-Popups (i_tabf) und füllen die Felder der Tabelle mit Werten (i_values). Zur Einsparung von Schreibarbeit wird bei der Deklaration ein Makro eingesetzt.

```
*...Aufbau der Definition für Eingabemaske
  CLEAR i_inf.
  i_inf-tabname   = 'TADIR'.
  i_inf-fieldname = 'OBJ_NAME'.
  i_inf-keyword   = 'Entwicklungsobjekt'.
  DESCRIBE FIELD tadir-obj_name LENGTH l_field_len
```

```
    IN CHARACTER MODE.
  i_inf-length    = l_field_len.
  i_inf-value     = '*'.                " Vorbelegung
  i_inf-lowercase = ' '.
  APPEND i_inf.

  CLEAR i_inf.
  i_inf-tabname   = 'TADIR'.
  i_inf-fieldname = 'AUTHOR'.
  i_inf-keyword   = 'Bearbeiter'.
  DESCRIBE FIELD tadir-author LENGTH l_field_len
    IN CHARACTER MODE.

  i_inf-length    = l_field_len.
  i_inf-value     = sy-uname.           " Vorbelegung
  i_inf-lowercase = ' '.
  APPEND i_inf.
```

Die Felder des Eingabe-Popups müssen durch mehrere Parameter eindeutig beschrieben werden. Diese Beschreibung ersetzt die Definition der Felder in einem Dynpro. Entsprechend umfangreich sind die erforderlichen Angaben. Für jedes Maskenfeld ist ein Datensatz in der Tabelle `i_inf` anzulegen. Die Felder `tabname` und `fieldname` benennen ein Feld aus dem Data Dictionary, das vom Funktionsbaustein benötigt wird, um die korrekten Strukturangaben zu ermitteln. Diese Angaben sind für die Typumwandlung des eingegebenen Wertes notwendig.

Jedes Eingabefeld wird innerhalb der Maske durch eine vorangestellte Bezeichnung erläutert. Diese ist im Feld `keyword` einzutragen. Im Feld `length` muss die gewünschte Länge des Eingabefeldes stehen. Dieser Wert sollte aus dem Data Dictionary ermittelt werden. Bei individuellen Anpassungen darf er keinesfalls größer sein als die im Data Dictionary hinterlegte Länge. Ein Eintrag im Feld `lowercase` bewirkt die Berücksichtigung von Groß- und Kleinschreibung. Ist es nicht gesetzt, werden eingegebene Werte automatisch in Großbuchstaben umgewandelt. Ein gesetztes Feld `intens` hebt die Feldbezeichnung in der Maske farbig hervor.

Das Feld `value` ist das eigentliche Wertefeld. Die dort bei Aufruf des Funktionsbausteins enthaltenen Werte werden angezeigt und editiert. Der Funktionsbaustein liefert im selben Feld die im Popup erfassten Werte zurück.

```
CALL FUNCTION 'HELP_GET_VALUES'
       EXPORTING
"           cucol = 5
"           curow = 5
            popup_title = 'Suchbegriffe Entwicklungsobjekte'
       IMPORTING
            returncode  = l_returncode
```

```
      TABLES
           fields      = i_inf
      EXCEPTIONS
           no_entries  = 01.

*...nach korrekter Eingabe Werte in Progfelder
  IF ( sy-subrc = 0 ) AND ( l_returncode = space ).
    LOOP AT i_inf.

      CASE i_inf-fieldname.
        WHEN 'OBJ_NAME'.
          l_name = i_inf-value.
          TRANSLATE l_name USING '*%+_'.

        WHEN 'AUTHOR'.
          l_author = i_inf-value.
          TRANSLATE l_author USING '*%+_'.
      ENDCASE.

    ENDLOOP.
  ELSE.
    WRITE: / 'Keine korrekte Eingabe'.
    EXIT.
  ENDIF.
```

Der Aufruf des Funktionsbausteins ist relativ einfach. Es sollte die Menüfunktion BEARBEITEN | ANWEISUNGSMUSTER benutzt werden, um den Aufruf in den Quelltext einzufügen. Die Anweisung ist dann nur noch mit einigen Parametern zu ergänzen, z.B. dem Titel und der vorher erstellten Tabelle mit den Feldbeschreibungen.

Nach Beendigung des Funktionsbausteins liefern sowohl `SY-SUBRC` als auch `returncode` Informationen über die Ausführung des Funktionsbausteins. Bei echten Fehlern wird eine Ausnahme ausgelöst, die das Feld `SY-SUBRC` auf einen Wert ungleich 0 setzt. Wird die Eingabe auf Wunsch des Anwenders abgebrochen, steht im Feld `returncode` ein Wert ungleich `SPACE`.

Nach Aufruf des Funktionsbausteins und korrekter Beendigung stehen die im Popup erfassten Werte im Feld `VALUE` der internen Tabelle. Um sie im weiteren Verlauf des Programms verwenden zu können, müssen sie aus den Tabellenfeldern in lokale Datenfelder des Programms übertragen werden. Dies erfolgt in einer `LOOP`-Schleife über die Tabelle `i_inf`. In dieser Schleife wird das Feld `fieldname` ausgewertet, um den Wert aus `value` dem korrekten Datenfeld des Programms zuzuweisen. Im Anschluss an die Übertragung werden eventuelle Musterzeichen umgesetzt. Der Anwender ist gewohnt, die Zeichen „*“ und „+“ nutzen zu können. Der `LIKE`-Operator der `SELECT`-Anweisung erwartet stattdessen die Zeichen „%“ und „_“.

```
*...Füllen der Tabelle mit den einzelnen Feldern,
*...jedes Feld ein Satz
*...Reihenfolge: 1. Zeile, 1. Feld; 1. Zeile, 2. Feld;
*...2. Zeile, 1. Feld ...
  SELECT * FROM tadir
  WHERE obj_name LIKE l_name
    AND author LIKE l_author.

    i_values = tadir-obj_name. APPEND i_values.
    i_values = tadir-pgmid.    APPEND i_values.
    i_values = tadir-object.   APPEND i_values.

  ENDSELECT.

  IF sy-dbcnt = 0.
    WRITE: / 'Keine Entwicklungsobjekte gefunden'.
    EXIT.
  ENDIF.
```

Nach dem Einlesen der Suchbegriffe kann die Suche in der Datenbank gestartet werden. Die Feldwerte der gefundenen Datensätze müssen einzeln und unbedingt entsprechend der Struktur der nachfolgend definierten Tabelle des Listen-Popups in die `i_values`-Tabelle übertragen werden. Falls die `SELECT`-Anweisung keine Datensätze findet, wird das Programm beendet.

```
*...Aufbau Auswahlliste

*...Füllen der Tabelle für Spaltenbezeichnungen
*...zuerst Spalte 1 für Name des Objekts
  i_tabf-tabname = 'TADIR'.
  i_tabf-fieldname = 'OBJ_NAME'.
  i_tabf-selectflag = 'X'. " Selektionsfeld, wird
zurückgegeben
  APPEND i_tabf.

*...anschließend Spalte 2 für Objektgruppe
  i_tabf-tabname = 'TADIR'.
  i_tabf-fieldname = 'PGMID'.
  i_tabf-selectflag = ' '.
  APPEND i_tabf.

*...zuletzt Spalte 3 für Objektart
  i_tabf-tabname = 'TADIR'.
  i_tabf-fieldname = 'OBJECT'.
  i_tabf-selectflag = ' '.
  APPEND i_tabf.
```

Der zweite Funktionsbaustein baut ein Popup mit einer Liste auf. Dazu muss ihm die Struktur der Liste übermittelt werden. Diese Struktur muss mit dem Inhalt der Tabelle `i_values` korrespondieren. Insbesondere muss die Zahl der Spalten der Tabelle mit der Zahl der aus einem Datensatz in `i_values` geschriebenen Werte übereinstimmen, da die Felder der Tabelle später ohne weitere Prüfungen zeilenweise aus `i_values` gefüllt werden. Die Deklaration der Tabelle ist einfach. Es sind lediglich wieder Tabellen- und Feldname zu übergeben, damit der Funktionsbaustein Typangaben aus dem Data Dictionary entnehmen kann. Eine Spalte der Tabelle kann eine besondere Bedeutung besitzen. Sie wird durch das gesetzte Feld `selectflag` gekennzeichnet. Bei Selektion einer Zeile der Tabelle wird der Inhalt dieser gekennzeichneten Spalte an das rufende Programm zurückgegeben.

```
*...Funktionsbaustein aufrufen, der Auswahlliste erzeugt
  CALL FUNCTION 'HELP_VALUES_GET_WITH_TABLE'
"     EXPORTING
"          cucol                          = 0
"          curow                          = 0
"          display                        = ' '
"          fieldname                      = ' '
"          tabname                        = ' '
"          no_marking_of_checkvalue       = ' '
"          title_in_values_list           = ' '
"          titel                          = ' '
"          show_all_values_at_first_time = ' '
     IMPORTING
          select_value                   = l_name
     TABLES
          fields                         = i_tabf
          valuetab                       = i_values
     EXCEPTIONS
          field_not_in_ddic              = 01
          more_then_one_selectfield      = 02
          no_selectfield                 = 03.

  p_subrc = sy-subrc.
  p_name = l_name.

ENDFORM.
```

Den Abschluss der Anwendung bildet der Aufruf des zweiten Funktionsbausteins. Er kann über eine größere Anzahl von Parametern modifiziert werden. Für den Standardfall reicht aber die Übergabe der beiden internen Tabellen und das Zuweisen eines Datenfeldes für den Rückgabewert aus. Der von diesem Funktionsbaustein gelieferte Eintrag sowie eine Kopie des Feldes `SY-SUBRC` werden den Parametern des Unterprogramms zugewiesen und so an das aufrufende Programm zurückgeliefert.

3.10.2 Lesen von Dynpro-Feldern

Das eben beschriebene Beispiel leitet über zu einer anderen Problematik. Eingabehilfen, die nach dem obigen Prinzip arbeiten, erfordern die Eingabe von Werten durch den Anwender. Möglicherweise hat er bereits im Dynpro einen Wert eingetragen, der nun als Suchbegriff verwendet werden könnte. Zum Zeitpunkt `PROCESS ON VALUE-REQUEST` werden die im Dynpro enthaltenen Werte aber noch nicht in die programminternen Felder übertragen. Mittels eines speziellen Funktionsbausteins ist aber auch hier Abhilfe möglich. Er gestattet das direkte Lesen von Werten aus einem Dynpro unter Umgehung des standardmäßigen Transfermechanismus. Die eigentliche Arbeit wird durch den Funktionsbaustein `DYNP_VALUES_READ` übernommen. Er erwartet nur wenige Parameter.

```
CALL FUNCTION 'DYNP_VALUES_READ'
     EXPORTING
          DYNAME               =
          DYNUMB               =
"         TRANSLATE_TO_UPPER   = ' '
     TABLES
          DYNPFIELDS           =
     EXCEPTIONS
          INVALID_ABAPWORKAREA = 01
          INVALID_DYNPROFIELD  = 02
          INVALID_DYNPRONAME   = 03
          INVALID_DYNPRONUMMER = 04
          INVALID_REQUEST      = 05
          NO_FIELDDESCRIPTION  = 06
          UNDEFIND_ERROR       = 07.
```

In den Parametern `dynumb` und `dyname` wird die Nummer des Dynpros und der Name des Programms, in dem dieses Dynpro enthalten ist, übergeben. Das Flag `translate_to_upper` sorgt für die Umwandlung des oder der gelesenen Werte in Großbuchstaben. Die bei `dynpfields` zu übergebende Tabelle enthält zunächst in den Feldern `fieldname` den Namen des oder der auszulesenden Felder. Nach erfolgreicher Ausführung des Funktionsbausteins stehen die gelesenen Werte in den Feldern `fieldvalue` dieser Tabelle. Die exakte Struktur der an `dynpfields` zu übergebenden Tabelle ist im Data Dictionary in der Struktur `dynpread` definiert.

3.11 Formularverarbeitung

Die Ausgabe- und Formatierungsmöglichkeiten herkömmlicher Reports ermöglichen zwar die Erzeugung von Drucklisten, sind aber nicht für den Formulardruck geeignet. Gerade diese Aufgabe stellt sich in der Praxis aber sehr häufig. Die Ausgabe von Formularen ist wesentlich komplizierter als die Erstellung

einer Druckliste. Der Formulardruck erfolgt daher über ein spezielles Verfahren, das in diesem Kapitel beschrieben werden soll. Grundlage für den Formulardruck sind Formulardefinitionen, die mit einem Werkzeug der Entwicklungsumgebung erstellt werden. Zum eigentlichen Formulardruck müssen Sie in Ihrem Programm spezielle Funktionsbausteine aufrufen, in denen die Formulardefinitionen benutzt werden.

Unter dem Begriff Formulardruck soll hier nicht nur das Ausfüllen von echten Vordrucken verstanden werden. Gemeint sind alle Druckausgaben, die formatiert oder gezielt auf dem Ausgabemedium platziert werden müssen. Neben dem Ausfüllen von echten Formularen wie z. B. Scheckvordrucken ist der Formulardruck beispielsweise auch für die Ausgabe von Serienbriefen oder die Aufbereitung von Langtexten innerhalb des SAP-Hilfe-Systems zuständig.

In die Formulardefinitionen können Sie Textelemente einbinden, die Sie mit dem so genannten *SAPScript-Editor* pflegen. Der Begriff Textelement bezeichnet in diesem Fall nicht die Textsymbole eines Programms, sondern andere Objekte, die nur innerhalb des Formulardrucks existieren.

Formulardefinitionen sind eine Teilmenge der allgemeinen Textverarbeitung des R/3-Systems. Im allgemeinen Sprachgebrauch hat sich daher die Bezeichnung *SAPScript* als Oberbegriff für alle Werkzeuge eingebürgert, die mit der Textpflege und dem Formulardruck zusammenhängen.

Formulare werden mandantenbezogen und natürlich sprachabhängig bearbeitet. Falls Eigenentwicklungen erfolgen, liegt die Verteilung der Formulare und eine eventuelle Übersetzung im Verantwortungsbereich des Entwicklers und des Systembetreuers.

Der vorliegende Abschnitt beschreibt zunächst das Prinzip des Formulardrucks. Ein daran anschließendes Beispiel gibt Ihnen die Gelegenheit, die Werkzeuge und die Programmierweise in der Praxis kennen zu lernen. Einige wichtige Details finden Sie in den Abschnitten nach dem Programmierbeispiel.

3.11.1 Prinzip

Die Formulardefinitionen pflegen Sie mit dem Formulareditor (Transaktion `SE71`). Ein Formular besteht aus der Kombination von unterschiedlichen Elementen. Die wichtigsten Elemente sind die *Seiten*, *Fenster*, *Textelemente und Absätze*. Die Erläuterung des Funktionsprinzips soll zunächst auf diese Elemente beschränkt bleiben.

Das grundlegende Element ist ein *Fenster*. Ein Fenster, auch als *Fensterdefinition* bezeichnet, enthält ein oder mehrere *Textelemente*. Jedes dieser Textelemente stellt einen beliebig umfangreichen Text dar, der einen innerhalb des Fensters eindeutigen Bezeichner erhält. Die Textelemente können neben normalem Text auch spezielle SAPScript-Anweisungen enthalten, zu denen beispielsweise Platzhalter für dynamische einzufügende Werte gehören. Ausgegeben werden

später nur Textelemente, die Sie explizit ansprechen. Es ist somit nicht notwendig, alle Textelemente eines Fensters zu drucken. Eine Fensterdefinition kann daher wesentlich mehr Text enthalten als später gedruckt wird.

In der Fensterdefinitionen legen Sie nur den Inhalt des jeweiligen Fensters und einige Textformatierungen fest. Die Fensterdefinition enthält allerdings keinerlei Informationen zur Anordnung des Textes bzw. des Fensters auf dem Formular.

Das endgültige Formular selbst besteht aus mindestens einer *Seite*. Neben anderen Angaben definieren Sie auf einer Seite ein oder mehrere so genannte *Seitenfenster*. Ein Seitenfenster legt fest, an welcher Stelle einer Seite ein Fenster gedruckt werden soll. Ein Seitenfenster ist im Gegensatz zu Seiten und Fenstern kein eigenständiges Element, sondern existiert nur als logisches Objekt innerhalb der Beschreibung einer Seite. Durch die Definition von Seiten und Seitenfenstern legen Sie fest, wo etwas gedruckt werden soll. Seitenfenster werden für jede Seite neu definiert. Es ist möglich, eine Fensterdefinition auf mehreren Seiten an unterschiedlichen Stellen, also in unterschiedlichen Seitenfestern, einzublenden.

Als weiteres Element stehen *Absatzdefinitionen*, abgekürzt und nicht ganz korrekt auch als *Absätze* bezeichnet, zur Verfügung. Durch Absatzdefinitionen legen Sie fest, wie der auszudruckende Text formatiert werden soll. Der Einfachheit halber können Sie für das gesamte Formular eine allgemein gültige Vorgabe treffen. Außerdem können Sie während der Pflege der Textelemente Attribute für einzelne Textabschnitte vergeben.

Für den Formulardruck stehen keine speziellen ABAP-Kommandos bereit. Der gesamte Formulardruck beruht auf dem Aufruf verschiedener Funktionsbausteine, die von Ihnen mit passenden Parametern versorgt werden müssen. Das folgende Listing enthält Pseudo-Code, der die einfachste Variante demonstriert.

Zunächst rufen Sie einen Funktionsbaustein auf, der Ihnen Druckparameter beschafft. Dieser Funktionsbaustein kann beispielsweise ein Popup anzeigen, im dem der Anwender den zu wählenden Drucker einträgt. Dieses Popup wird beispielsweise auch vom SAP-System automatisch vorgeblendet, wenn Sie eine Druckliste ausdrucken.

```
CALL FUNCTION 'GET_PRINT_PARAMETRERS'
  EXPORTING
    OUT_PARAMETERS = print_parameters
  ...
```

Die Druckparameter werden in der Anwendung zwischengespeichert. Sie müssen diversen anderen Funktionsbausteinen als Parameter übergeben werden. Unter Umständen können Sie sogar auf den Aufruf dieses Bausteins verzichten, da ein ähnlicher Druckdialog auch vom nachfolgenden Funktionsbaustein aufgerufen werden kann.

Nach dem Initialisieren der Druckparameter erzeugen Sie einen Spool-Auftrag. Alle im weiteren Verlauf der Anwendung erzeugten Formulare stehen in diesem Spool-Auftrag. Das bedeutet, dass alle Formulare später in einem Zug gedruckt werden. Beim Erzeugen des Spool-Auftrags können Sie gleichzeitig ein Formular auswählen. Das ist aber nicht zwingend notwendig, sie können das Formular auch noch später festlegen.

```
CALL FUNCTION 'OPEN_FORM' ...
```

Dieser Aufruf leitet nur die Erzeugung des Spool-Auftrags ein, nicht aber die Erstellung eines konkreten Formulars. Dazu müssen Sie den nachfolgenden Funktionsbaustein aufrufen. Bei diesem Aufruf können Sie die Sprache und das konkrete Formular festlegen. Diesen Baustein können Sie mehrfach aufrufen. Dadurch ist es Ihnen möglich, innerhalb eines Spool-Auftrags das Formular zu wechseln.

```
CALL FUNCTION 'START_FORM' ...
```

Nach der Auswahl des Formulars müssen Sie die auszugebenden Elemente (Fenster und Textelemente) ansprechen. Dadurch werden diese in das Formular gedruckt. Sofern in den auszugebenden Texten Platzhalter für dynamische Werte existieren, werden diese automatisch ersetzt.

```
CALL FUNCTION 'WRITE_FORM' ...
```

Nachdem alle Ausgaben in das Formular erfolgten, schließen Sie das Formular ab.

```
CALL FUNCTION 'END_FORM' ...
```

Sie können nun durch erneuten Aufruf des Funktionsbausteins `START_FORM` ein weiteres Formular drucken. In der Praxis werden Sie die drei Funktionsbausteine `START_FORM`, `WRITE_FORM` und `END_FORM` daher in einer Schleife anordnen.

Nach Ausgabe aller Formulare ist der Spool-Auftrag abzuschließen.

```
CALL FUNCTION 'CLOSE_FORM' ...
```

Alle zwischen `OPEN_FORM` und `CLOSE_FORM` erstellten Formulare erscheinen somit ein einem Spool-Auftrag. Die Funktionsbausteine `OPEN_FORM` und `CLOSE_FORM` sowie `START_FORM` und `END_FORM` müssen immer paarweise benutzt werden.

3.11.2 Beispiel

Das folgende Beispiel verwendet die Tabelle `TADIR` und Angaben aus dem Benutzerstammsatz, um einen Serienbrief zu erzeugen. Einen betriebswirtschaftlichen Sinn besitzt auch diese Anwendung nicht, sie dient lediglich zur Demonstration der diversen Werkzeuge und des Programmiermodells beim Formulardruck.

Das Programm ermittelt im aktuellen Mandanten alle existierenden Benutzer. Für jeden der Benutzer werden in der Tabelle TADIR die vom ihm erzeugten oder modifizierten Objekte gesucht. Alle ermittelten Objekte werden in Form eines Serienbriefs ausgegeben. Dieser besteht aus einem konstanten Absender, einer Adresse, die den Namen des Anwenders enthält sowie der tabellenartigen Auflistung aller vorher ermittelten Objekte. Falls ein Anwender keine Objekte angelegt hat, erhält er ebenfalls einen Brief, der lediglich einen Standardtext enthält.

Formular erzeugen

Zunächst wird das Formular erzeugt. Da der Formulareditor kein grafisches Werkzeug, sondern eine auf Dynpros beruhende Dialoganwendung ist, sollte der konkrete Aufbau des Formulars schon vor Aufruf des Werkzeugs feststehen. Die Anfertigung eines maßstabsgerechten Musters ist sehr zu empfehlen.

Der Serienbrief dieses Beispiels soll aus zwei unterschiedlich formatierten Seiten bestehen. Die erste Seite nimmt die Absenderangabe und die Adresse sowie einen Textkörper auf. Die folgenden Seiten bestehen nur aus dem Textkörper.

Am einfachsten starten Sie den Formulareditor über den Transaktionscode SE71. Vom Startmenü aus (nicht vom Object Navigator) erreichen Sie die Anwendung über den Menüpfad HILFSMITTEL | SAPSCRIPT | FORMULAR. Abbildung 3.168 zeigt Ihnen das Startbild des Formulareditors.

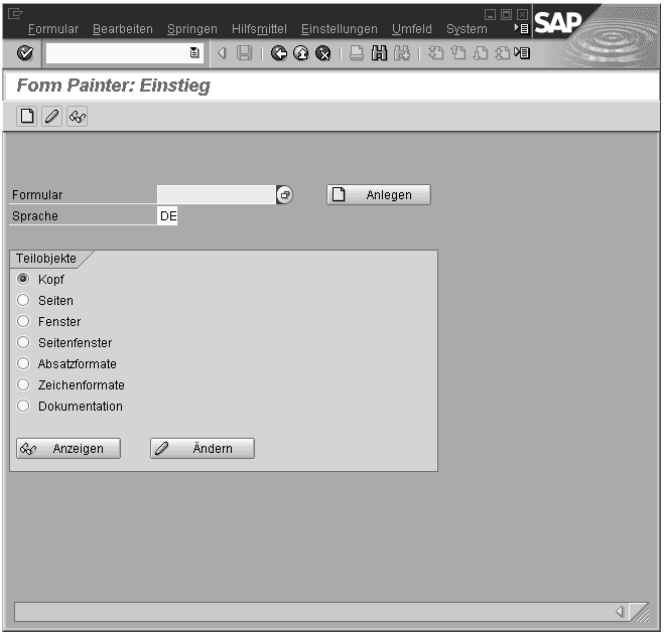

Abbildung 3.168 **© SAP AG**
Startbild des Formulareditors

Im Eingabefeld FORMULAR tragen Sie den Namen des anzulegenden Formulars ein. Auch dieser Name unterliegt den Namenskonventionen und muss in Kundensystemen daher mit dem Buchstaben „Y“ oder „Z“ beginnen. In diesem Beispiel wird als Name `Y_Z4_FORM1` vergeben. Im Feld SPRACHE tragen Sie die von Ihnen gewünschte Sprache ein. Eventuell ist dieses Feld mit der Anmeldesprache vorbelegt. Da noch kein Formular existiert, hat die Markierung im Bereich TEILOBJEKTE keinen Einfluss auf den weiteren Ablauf.

Durch Betätigen der Schaltfläche ANLEGEN gelangen Sie zu einem Dynpro zur Pflege der allgemeinen Verwaltungsdaten (Abbildung 3.169). In diesem Dynpro ist lediglich das Feld BEDEUTUNG mit einer Kurzbeschreibung der Aufgabe des Formulars zu füllen. Im Eingabebereich für die Sprachattribute können Sie festlegen, ob und wenn ja, in welche Sprachen das Formular übersetzt werden soll. Diese Einstellung ist nur beim Einsatz der R/3-eigenen Übersetzungswerkzeuge von Bedeutung. Dies wird vorrangig bei SAP der Fall sein. Falls Sie an einem System arbeiten, in dem das Übersetzungstool genutzt wird, sollten Sie das Flag NICHT ÜBERSETZEN markieren, es sei denn, Sie möchten im Anschluss an die Formularentwicklung auch die Übersetzungswerkzeuge kennen lernen. Alle anderen Einstellungen bleiben unverändert.

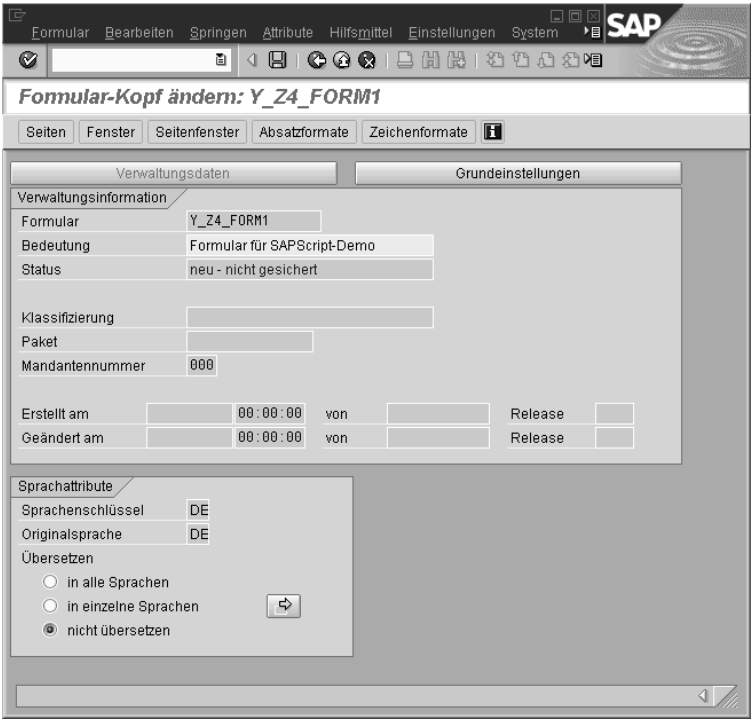

Abbildung 3.169 **© SAP AG**
Verwaltungsdaten des Formulars pflegen

Nach der Pflege der Verwaltungsdaten speichern Sie die Einstellungen. Im nächsten Schritt müssen Sie ein Absatzformat definieren. Dieses Format wird später als Standardformat für das Formular benutzt. Zum Pflege-Dynpro für die Absatzformate gelangen Sie mit der Schaltfläche ABSATZFORMATE.

Die Pflege-Dynpros für die diversen Elemente eines Formulars sind einander recht ähnlich. In einem Arbeitsbereich werden alle existierenden Elemente aufgelistet. Eines dieser Elemente können Sie durch einen Doppelklick mit der Maus zum aktiven Element machen. Es wird farblich hervorgehoben. In Eingabefeldern im unteren Bereich des Dynpros pflegen Sie die Eigenschaften des aktuellen Elements.

Um ein neues Format anzulegen, rufen Sie die Menüfunktion BEARBEITEN | ANLEGEN ELEMENT auf. In einem Popup (siehe Abbildung 3.170) müssen Sie nun den zweistelligen Bezeichner sowie einen Kurztext erfassen.

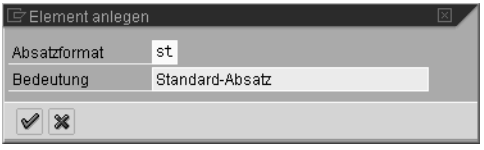

Abbildung 3.170 **© SAP AG**
Anlegen eines neuen Formats

Das so definierte Format wird in die Arbeitsfläche des Dynpros aufgenommen (siehe Abbildung 3.171). Da nur ein Element existiert, ist es automatisch als aktives Element markiert. Es ist für dieses erste Beispiel nicht notwendig, dem Format weitere Eigenschaften zuzuweisen. Sie können das Formular nun speichern und die Bearbeitung der Formularseiten fortsetzen.

Zur Pflege der Formularseiten gelangen Sie mit der Schaltfläche SEITEN. Das Dynpro ähnelt dem zur Pflege der Attribute. Eine neue Seitendefinition legen Sie mit der Menüfunktion BEARBEITEN | ANLEGEN ELEMENT an. Es erscheint wiederum ein Popup mit zwei Eingabefeldern (siehe Abbildung 3.172), in dem der Name der Seite sowie eine Kurzbezeichnung erfasst wird. Legen Sie zunächst die Seite `FIRST` und danach die Seite `NEXT` an, ohne Attribute für diese Seiten zu pflegen.

Wählen Sie nun durch einen Doppelklick mit der Maus die Seite `FIRST` aus. Pflegen Sie das Attribut FOLGESEITE. Tragen Sie als Folgeseite `NEXT` ein (siehe Abbildung 3.173). Das Attribut FOLGESEITE legt fest, dass nach der Seite FIRST die Seite NEXT aufgerufen wird. Auf diese Weise können Sie die erste Seite abweichend von den folgenden Seiten formatieren. Dieses Attribut muss auch in der Seite NEXT belegt werden, dort ebenfalls mit NEXT. Nur dadurch erreichen Sie, dass nach der Ausgabe der ersten mit NEXT formatierten Seite (also der zweiten des Gesamtdokuments) wieder eine mit NEXT formatierte Seite aufgerufen wird. Ohne diese Einstellung würde nach der Erstellung der zweiten Seite eine nicht definierte Folgeseite aufgerufen und das Programm dadurch abgebrochen werden.

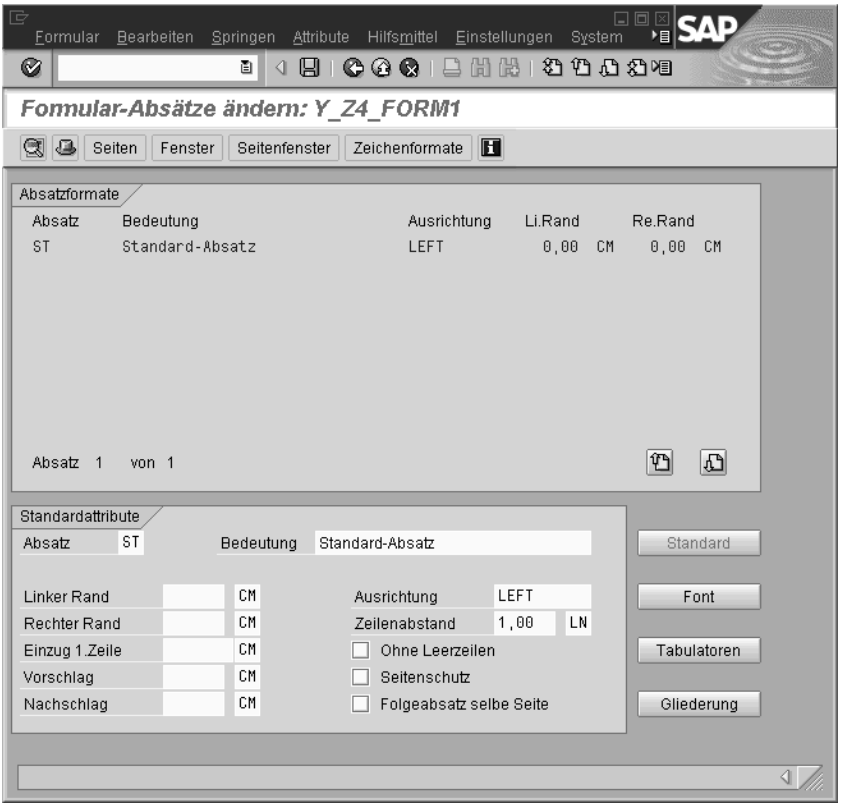

Abbildung 3.171 **© SAP AG**
Pflege-Dynpro für Formular-Absätze

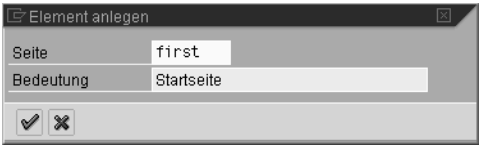

Abbildung 3.172 **© SAP AG**
Anlegen einer Seitenbeschreibung

Nach den Seiten definieren Sie als drittes wichtiges Element die Fenster. Rufen Sie dazu mit der Schaltfläche FENSTER das entsprechende Pflege-Dynpro auf. Abbildung 3.178 zeigt Ihnen den Initialzustand dieses Dynpros.

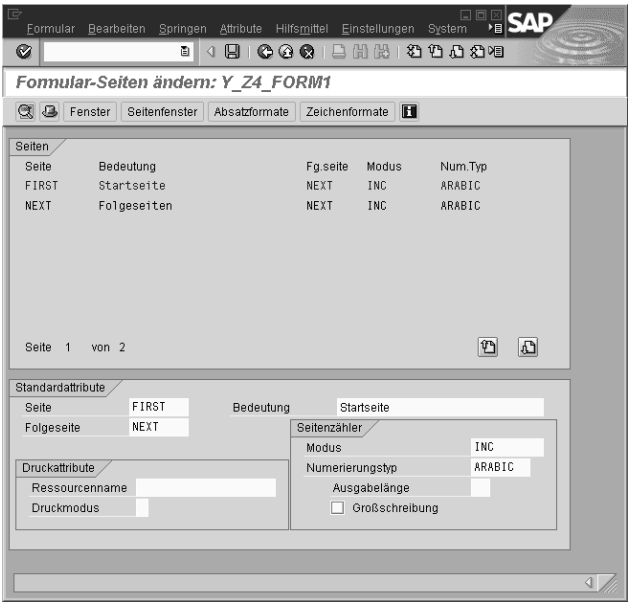

Abbildung 3.173 **© SAP AG**
Pflege der Attribute einer Seitenbeschreibung

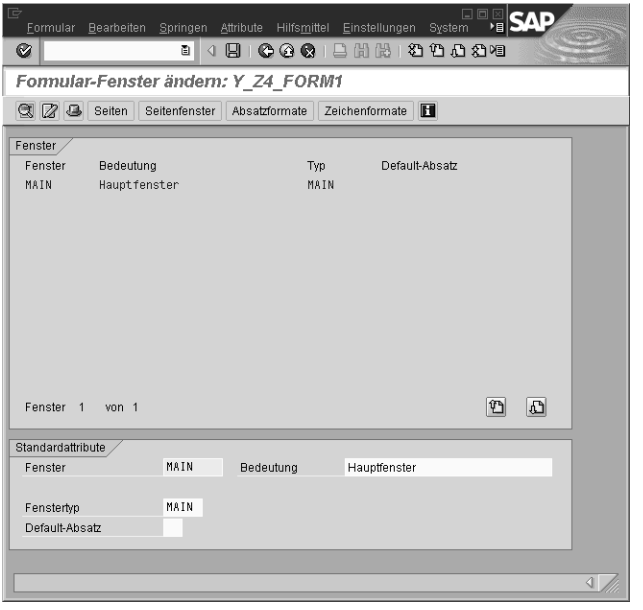

Abbildung 3.174 **© SAP AG**
Pflege-Dynpro für Fensterdefinitionen

Abweichend von den beiden anderen Dynpros ist hier ein Element standardmäßig vorgegeben. Da das Fenster MAIN in allen Formularen existieren muss, wird es bereits durch das System für Sie angelegt. Erzeugen Sie nun mit der bereits bekannten Menüfunktion BEARBEITEN | ANLEGEN ELEMENT die beiden Fenster ABS für den Absender und ADR für den Empfänger. Auch für die Fenster pflegen Sie in einem Popup (siehe Abbildung 3.175) zwei Felder mit dem Namen des Fensters und dessen Kurzbezeichnung.

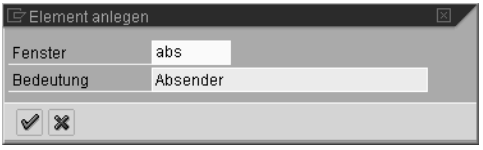

Abbildung 3.175 **© SAP AG**
Anlegen eines neuen Fensters

Die durch das System eingetragenen Standardwerte bleiben erhalten. Nach dem Speichern der Einstellungen können Sie die eben erzeugten Fenster benutzen, um die Seitenfenster anzulegen. Dies bedeutet, dass Sie auf jeder der beiden bereits definierten Seiten eines oder mehrere Fenster anordnen und dabei die Abmessungen der Fenster festlegen. Zum Pflege-Dynpro für die Seitenfenster (siehe Abbildung 3.176) gelangen Sie mit der Drucktaste SEITENFENSTER.

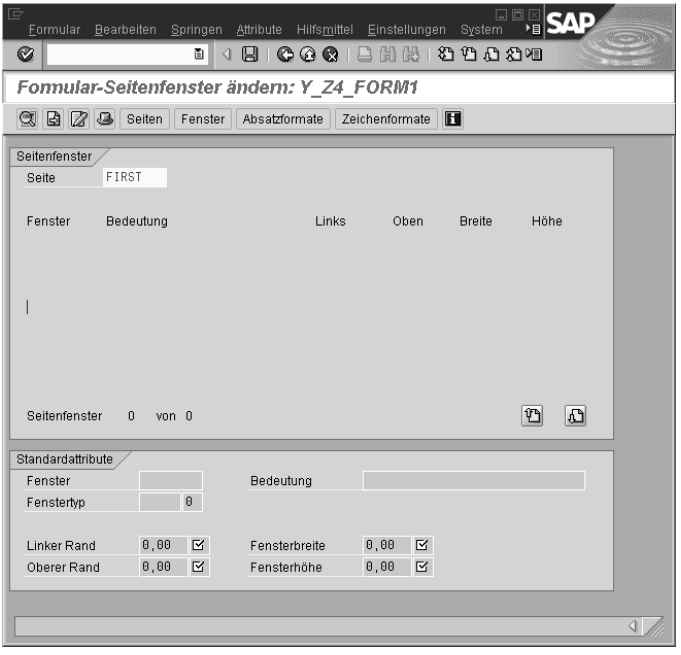

Abbildung 3.176 **© SAP AG**
Initialzustand des Pflege-Dynpros für Seitenfenster

Dieses Dynpro unterscheidet sich in einem Detail von den anderen. In einem Eingabefenster im oberen Teil der Arbeitsfläche wählen Sie zunächst per Eingabehilfe die zu bearbeitende Seite (also FIRST oder NEXT) aus. Der Rest des Dynpros zeigt dann nur Informationen für diese Formularseite an.

Wie in den anderen Dynpros auch, fügen Sie mit der Menüfunktion BEARBEITEN | ANLEGEN ELEMENT ein neues Seitenfenster ein. Diese Menüfunktion führt Sie zu einem Popup, in dem alle zur Verfügung stehenden Fensterdefinitionen aufgelistet werden (Abbildung 3.177). Per Doppelklick wählen Sie ein Fenster aus.

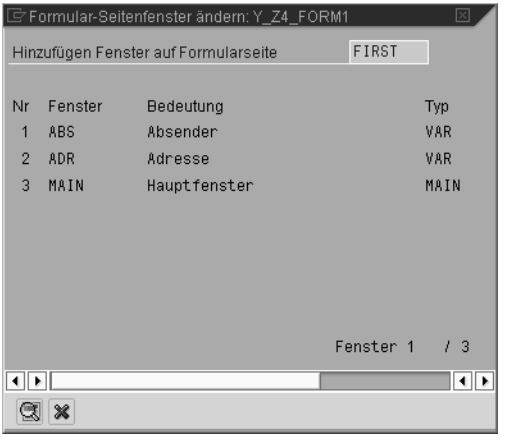

Abbildung 3.177 **© SAP AG**
Auswahl einer Fensterdefinition für ein neues Seitenfenster

Das selektierte Fenster wird in die Arbeistfläche des Seitenfenster-Dynpros übernommen. Im unteren Teil des Dynpros stehen vier Eingabefelder bereit, in denen die Abmessungen des jeweiligen Fensters gepflegt werden müssen (siehe Abbildung 3.178). Einige dieser Felder sind Muss-Eingabefelder. Sie müssen also einen Wert eintragen, bevor Sie weitere Seitenfenster übernehmen können. Die Bearbeitung beschränkt sich in diesem Beispiel auf das Festlegen der Größe und der Position des Seitenfensters.

Die einzufügenden Seitenfester und deren Position (in cm) finden Sie in Tabelle 3.45.

Seite	Fenster	Linker Rand	Oberer Rand	Breite	Höhe
FIRST	MAIN	1	7	15	20
	ABS	1	1	5	3
	ADR	7	1	8	4
NEXT	MAIN	1	1	15	26

Tabelle 3.45
Abmessungen der Seitenfenster des Formulars in cm

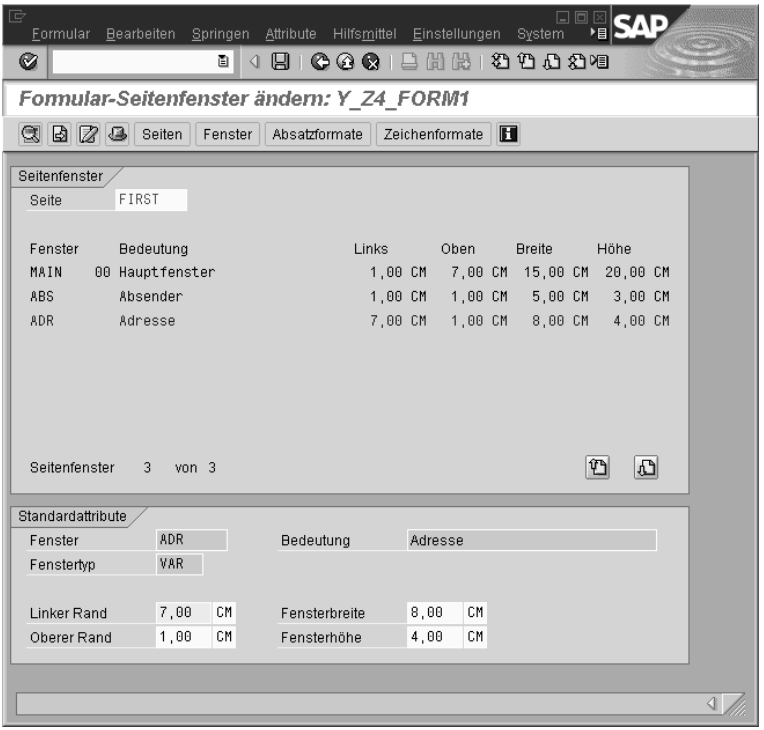

Abbildung 3.178 **© SAP AG**
Pflege eines Seitenfensters

Zum Abschluss der Definition der Formularstruktur müssen in den Verwaltungsdaten noch ein Default-Absatz sowie die Startseite eingetragen werden. Wechseln Sie dazu mit der Drucktaste KOPF (Menüfunktion SPRINGEN | KOPF bzw. Funktionstaste F5) zu den Kopfdaten des Formulars und von dort mit der Drucktaste GRUNDEINSTELLUNGEN zum im Abbildung 3.179 dargestellten Dynpro. Tragen Sie im Feld DEFAULT-ABSATZ die zu Beginn angelegte Absatzdefinition ST und im Feld STARTSEITE den Namen der ersten Seite FIRST ein.

Sie können das gesamte Formular nun speichern. Es ist empfehlenswert, das Formular durch Aufruf der Menüfunktion FORMULAR | PRÜFEN | DEFINITION auf Korrektheit zu überprüfen. Das Ergebnis der Prüfung erscheint in der Statuszeile.

Mit den beschriebenen Arbeitsgängen haben Sie ein leeres Formular angelegt. Es ist in diesem Zustand bereits verwendbar, allerdings müssten Sie den gesamten auszugebenden Text innerhalb Ihrer Anwendung durch Aufruf spezieller Funktionsbausteine in das Formular übertragen. Wesentlich komfortabler ist die Definition von Textelementen innerhalb der Fenster. Diese Tätigkeit soll im Folgenden beschrieben werden.

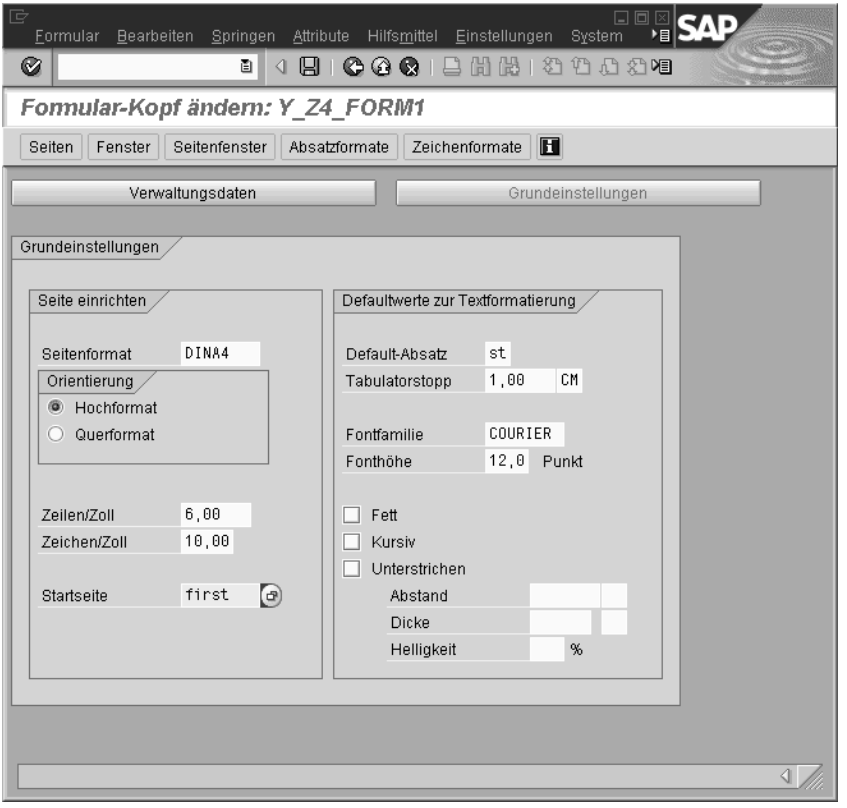

Abbildung 3.179 **© SAP AG**
Standardformat und Startseite für das Formular pflegen

Textelemente anlegen

Textelemente legen Sie als Unterobjekte innerhalb einer Fensterdefinitionen an. Rufen Sie dazu zunächst mit der Drucktaste FENSTER das Werkzeug zur Bearbeitung der Fensterdefinitionen auf. Markieren Sie den Eintrag für das Hauptfenster durch einen Doppelklick. Betätigen Sie dann das Symbol für die Textelemente oder rufen Sie den Editor mit der entsprechenden Menüfunktion BEARBEITEN | TEXTELEMENTE oder der Funktionstaste F9 auf. Das System stellt Ihnen nun einen Editor mit einer fast vollständig leeren Arbeitsfläche zur Verfügung (siehe Abbildung 3.180).

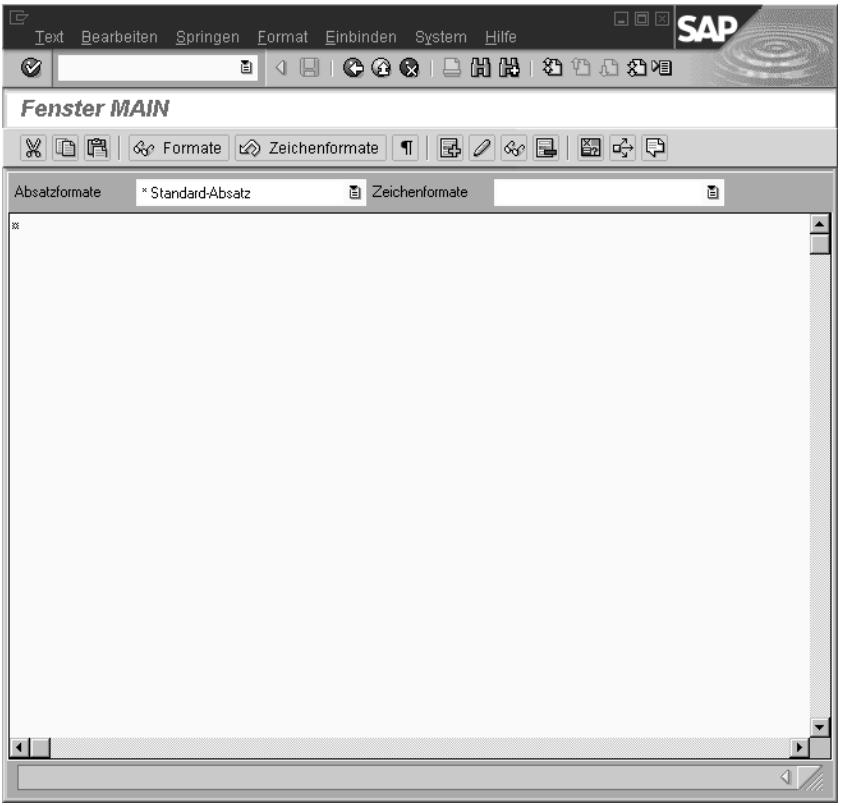

Abbildung 3.180 **© SAP AG**
Editor für Textelemente

Der Editor für die Formular-Textelemente wurde gegenüber älteren Versionen stärker überarbeitet, so dass sich größere Änderungen in der Bedienung ergeben. Innerhalb dieses Editors können nicht nur einfache Zeichenketten eingefügt werden. Es existieren einige zusätzliche Elemente, die zur Laufzeit eine besondere Bedeutung erhalten. So werden im dargestellten Editorfenster alle Textelemente eines Fensters gepflegt. Damit diese Textelemente separat ausgewählt und im Formular benutzt werden können, wird eine Namensgebung erforderlich. Ein spezielles Werkzeug ermöglicht es, neben dem eigentlichen Text eine Reihe von Symbolen einzufügen.

Rufen Sie mit dem Symbol ZUSATZINFORMATIONEN oder der Funktionstaste Ctrl-F9 die Eingabehilfe für die Zusatzinformationen auf (Abbildung 3.181). Markieren Sie dort die Auswahl und tragen Sie im dazu gehörigen Eingabefeld den Namen des anzulegenden Textsymbols, hier INTRO, ein.

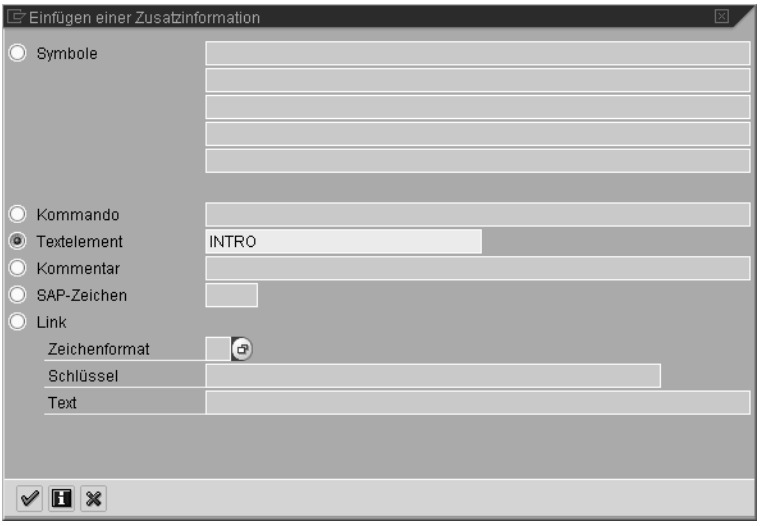

Abbildung 3.181 **© SAP AG**
Auswahl einer Formatangabe für Textelemente

Das System fügt nun im Editor eine grau hinterlegte Zeile mit dem Begriff `INTRO` ein. Diese Zeile kann nicht manuell editiert werden. Sie stellt den Identifikator eines Textelementes dar. Der eigentliche Text wird in der darauf folgenden Zeile erfasst. Er erstreckt sich bis zum nächsten Textsymbol-Namen (siehe Abbildung 3.182).

Innerhalb des Textes können Platzhalter für dynamisch einzufügende Werte berücksichtigt werden. Mit der Menüfunktion EINBINDEN | SYMBOLE | SYSTEM | ... können Sie beispielsweise diverse Systemfelder in den Text einfügen. Im Beispiel wird `SYST-UNAME` als Platzhalter für den Namen des angemeldeten Anwenders benutzt.

In der Abbildung 3.182 finden Sie neben dem eben erwähnten Systemsymbol weitere Symbole, mit denen auf Felder des Druckprogramms zugegriffen wird. Ein Formular existiert nie für sich allein, sondern wird immer von einem Programm, dem Druckprogramm, benutzt. Dieses Programm stellt in seinen internen Feldern Daten bereit, die im Formular ausgegeben werden sollen. Dazu muss im Formular durch so genannte Programmsymbole die Verbindung zu den Feldern des Programms hergestellt werden. Auch dafür steht ein Einfügewerkzeug bereit. Dieses funktioniert aber erst dann korrekt, wenn dem Formular mitgeteilt wird, von welchen Druckprogrammen aus es verwendet werden soll. Zwangsläufig muss dieses Programm bereits existieren. Die folgenden Aktionen können Sie deshalb erst durchführen, wenn Sie das im nächsten Unterabschnitt näher beschriebene Druckprogramm erstellt und getestet haben. Legen Sie zunächst auch die Textelemente der anderen Fenster an, verzichten Sie aber auf das Einfügen von Programmsymbolen.

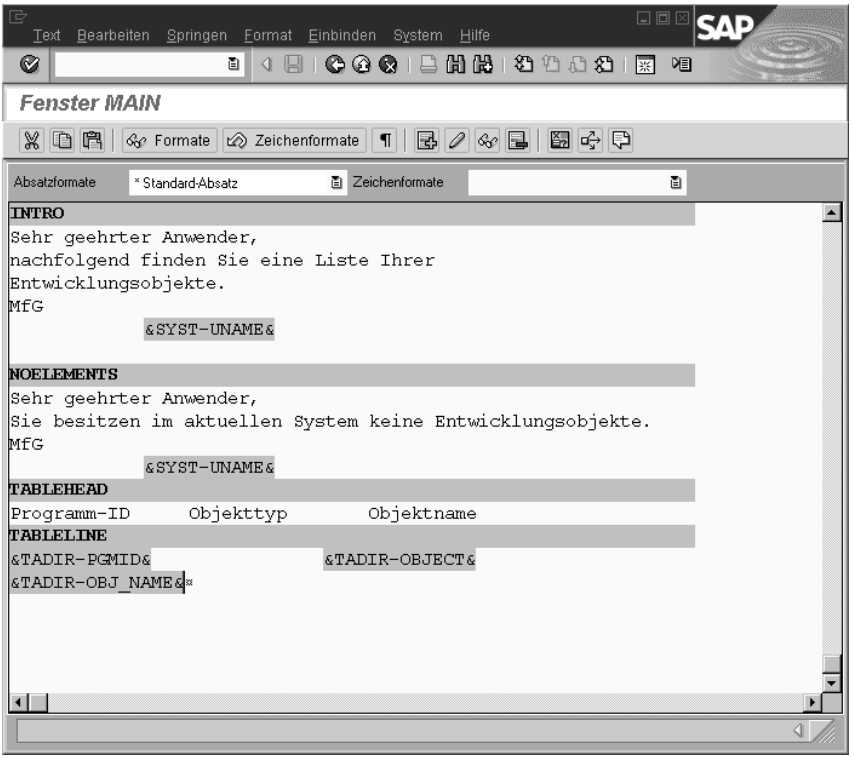

Abbildung 3.182 **© SAP AG**
Textelemente des MAIN-Fensters

Nach dem Erfassen des Textes kehren Sie zum Pflege-Dynpro für die Formularfenster zurück. Dazu können Sie nur die Funktionstaste F3 oder das entsprechende Symbol benutzen. Separates Abspeichern der Textelemente ist nicht notwendig, sie werden zusammen mit dem Fenster gespeichert. Weisen Sie nun den beiden anderen Fenstern Textelemente zu. Da in diesen Fenstern immer nur ein Standardtext erscheinen soll, müssen Sie innerhalb des Textes für diese Fenster keine explizit benannten Textelemente anlegen. Es reicht aus, den eigentlichen Text zu notieren.

Der Inhalt des Absender-Fensters besteht aus einer einzigen Zeile:

```
Absender: &SY-UNAME&
```

Der Inhalt des Adress-Fensters ist etwas umfangreicher:

```
Empfänger: &USR03-BNAME&
           &USR03-NAME1&
           &USR03-NAME2&
           &USR03-ABTLG&
```

Anschließend sichern Sie das Formular und kehren zur Pflege der Kopfdaten zurück. Es ist empfehlenswert, das Formular erneut zu prüfen. Benutzen Sie dazu die Menüfunktion FORMULAR | PRÜFEN | DEFINITION.

Bevor ein Formular verwendet werden kann, muss es aktiviert werden. Dazu dient die Menüfunktion FORMULAR | AKTIVIEREN. Erst durch die Aktivierung werden die gepflegten Daten für andere Anwendungen wirksam. Die Aktivierung ist deshalb nach jeder Änderung erneut durchzuführen. Während der Bearbeitung eines Formulars ändert sich der Inhalt der momentan aktiven Version nicht. Legen Sie nun zunächst das Druckprogramm an und testen Sie es. Das Programm wird zwar nicht die gewünschte Ausgabe liefern, aber durch die Verwendung des soeben angelegten Formulars im Programm kennt dieses nun das Druckprogramm. Kehren Sie zur Bearbeitung der Textelemente des Hauptfensters zurück. Rufen Sie die Menüfunktion SPRINGEN | DRUCKPROGRAMM auf. Bild 3.183 zeigt Ihnen das Popup, in dem Sie eines der Druckprogramme, in denen das aktuelle Formular benutzt wird, auswählen müssen. Falls Sie dem Formular ein neues Programm bekannt machen möchten, können Sie dies durch die Drucktaste DRUCKPROGRAMM HINZUFÜGEN tun. Wählen Sie dann über die Drucktasten DDIC-Felder oder GLOBALE DATEN aus, welche Art von Feldern Sie im Formular einfügen möchten.

Das nächste Popup (Abbildung 3.184) listet alle verfügbaren Datenobjekte auf.

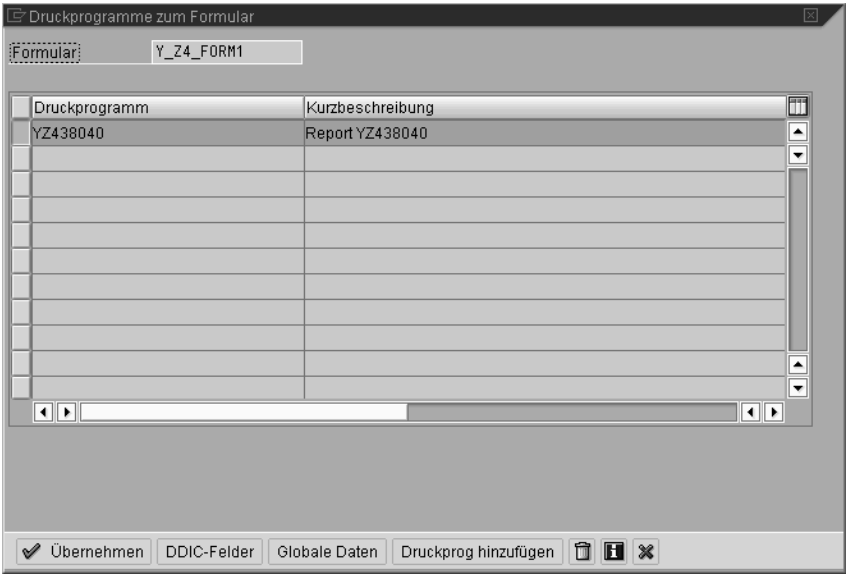

Abbildung 3.183 **© SAP AG**
Auswahl des Druckprogramms zwecks Übernahme von Programmsymbolen

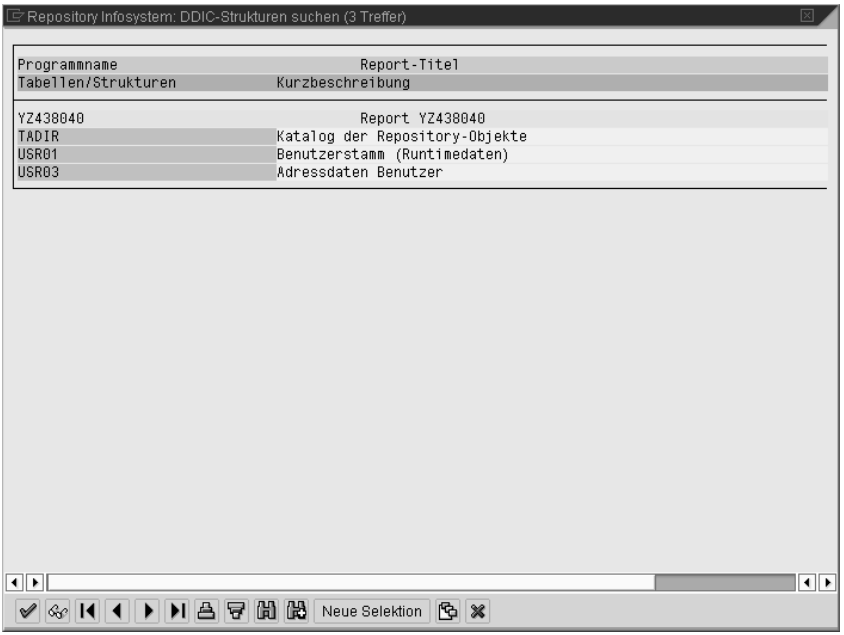

Abbildung 3.184 **© SAP AG**
Verfügbare Dictionary-Objekte im Druckprogramm

Nach Auswahl eines Dictionary-Objekts stehen im darauf folgenden Popup (Abbildung 3.185) alle Felder der Struktur bereit. Per Doppelklick kann eines übernommen werden. Im Textelemente-Editor erscheint dann das entsprechende Programmsymbol. Nach Übernahme aller Programmsymbole müssen Sie diese noch durch Tabulatoren ausrichten. Anschließend können Sie das Druckprogramm erneut testen. Nun sollten Sie die erwartete Bildschirmausgabe erhalten.

Neben dem eben beschriebenen Editor existiert weiterhin die per Standard-Dynpro-Technik realisierte Urform (siehe Abbildung 3.186). Sie kann per Menüfunktion EINSTELLUNGEN | FORMPAINTER aktiviert werden. Diese Funktion ruft ein Popup auf, in dem Sie per Checkbox sowohl den grafischen Form Painter als auch den Grafischen PC-Editor aktivieren oder deaktivieren können. Außerdem wird dieser Editor vom System automatisch benutzt, wenn in der Formulardefinition ein Fehler auftritt, der die korrekte Darstellung in der moderneren Variante verhindert.

Da die alte Variante des Editors komplett über Texteingaben ohne Einfügewerkzeuge o. Ä. bedient werden kann, bietet sie sich für den kundigen Anwender für die schnelle Arbeit nach wie vor an.

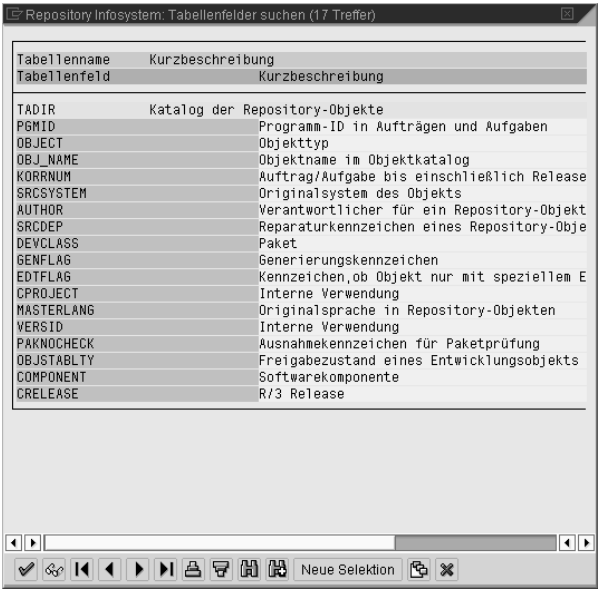

Abbildung 3.185 **© SAP AG**
Felder einer Dictionary-Tabelle

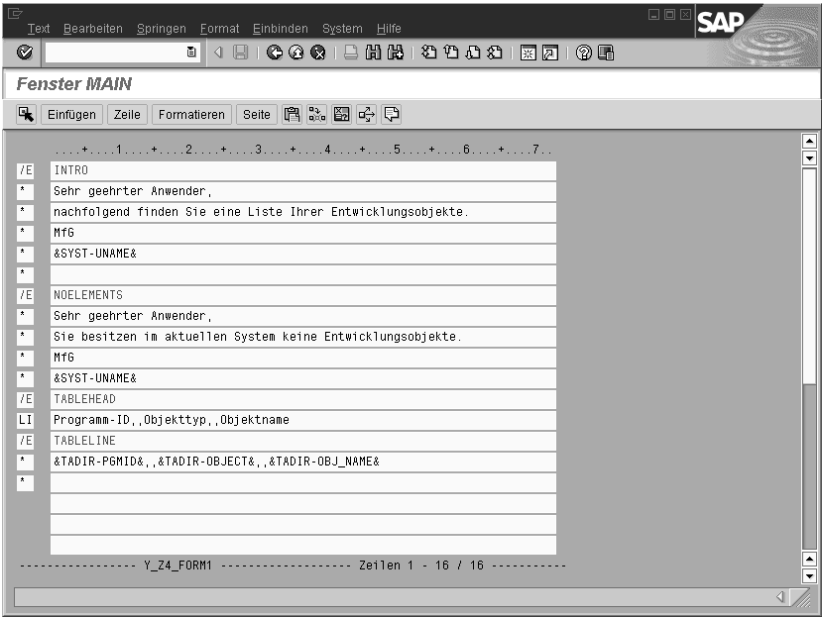

Abbildung 3.186 **© SAP AG**
Alter Textelemente-Editor

Programm zur Selektion

Das nachfolgende Programm entspricht in seiner Struktur dem bereits vorgestellten Pseudo-Code. Es soll lediglich die praktische Anwendung der diversen Funktionsbausteine demonstrieren und ist dementsprechend einfach gehalten. Der besseren Übersicht wegen wurden aus den Listings alle nicht benutzten Funktionsparameter bis auf eine Ausnahme entfernt. Die im Fehlerfall von den Funktionsbausteinen erzeugten Ausnahmen (Exceptions) werden einzeln behandelt, um eine Fehlersuche zu erleichtern. Im Fehlerfall bricht das Programm mit der Ausgabe des Namens des Funktionsbausteins und der Fehlernummer ab. Da mehrere `WRITE_FORM`-Aufrufe existieren, erfolgt bei Meldungen zu diesem Baustein auch die Angabe des Ausgabefensters. Dadurch ist es möglich, die Fehler auslösende Stelle im Programmcode zu ermitteln.

Das Programm beginnt mit der Deklaration der benötigten Tabellen und einiger Datenfelder. Die Tabelle `USR03` liefert die hier interessierenden Teile des Anwenderstammsatzes. In der Tabelle `TADIR` werden die Objekte der Entwicklungsumgebung gelesen. Erläuterungen zu den beiden Datenfeldern `f_count` und `f_headline` finden Sie im weiteren Verlauf des Textes. Damit die Zahl der auszugebenden Formulare eingeschränkt werden kann, steht eine Selektion `s_user` zur Auswahl der zu bearbeitenden Anwender zur Verfügung.

```
REPORT  yz438040.
TABLES:
  tadir,
  usr03.
DATA:
  f_count LIKE sy-tabix,
  f_headline VALUE space.
SELECT-OPTIONS:
  s_user FOR usr03-bname.
```

Die erste Aktion innerhalb der Anwendung besteht im Aufruf des Funktionsbausteins `OPEN_FORM`, um einen neuen Ausgabeauftrag zu erzeugen. Dazu müssen vier Parameter übergeben werden. Durch den Wert `SCREEN` im Parameter `DEVICE` legen Sie als Ausgabegerät den Bildschirm fest. In diesem Fall müssen keine Druckparameter beschafft werden. Daher ist der Parameter `DIALOG` auskommentiert. Er wird erst benötigt, wenn eine Ausgabe auf einen Drucker (`DEVICE = 'PRINTER'`) erfolgen soll. Innerhalb der Anwendung wird das Formular nicht gewechselt. Sie können es der Formularverwaltung daher bereits an dieser Stelle durch Zuweisen des Namens zum Parameter `FORM` bekannt machen. Letztendlich müssen Sie auch noch die Sprache festlegen, in der das Formular gesucht werden soll. Die Verwendung der Anmeldesprache ist natürlich nur dann korrekt, wenn Sie das Formular in ebendieser Sprache erstellt haben.

```
CALL FUNCTION 'OPEN_FORM'
  EXPORTING
    device                        = 'SCREEN'
*   dialog                        = 'X'
    form                          = 'Y_Z4_FORM1'
    language                      = sy-langu
  EXCEPTIONS
    canceled                      = 1
    device                        = 2
    form                          = 3
    options                       = 4
    unclosed                      = 5
    mail_options                  = 6
    archive_error                 = 7
    invalid_fax_number            = 8
    more_params_needed_in_batch = 9
    OTHERS                        = 10.
IF sy-subrc <> 0.
  WRITE: / 'OPEN_FORM:', sy-subrc.
  EXIT.
ENDIF.
```

In der Anwendung müssen Sie nur einen einzigen Ausgabeauftrag erzeugen. Innerhalb dieses Auftrages sollen aber mehrere Formulare erstellt werden. Die weiteren Anweisungen befinden sich daher in einer `SELECT`-Schleife, in der nacheinander alle durch das Selektionskriterium angesprochenen Anwenderstammsätze behandelt werden.

```
SELECT * FROM usr03
  WHERE bname IN s_user.
```

Möglicherweise existieren für den aktuellen Anwender keine Objekte in der Entwicklungsumgebung. Das erzeugte Formular soll in diesem Fall einen entsprechenden Text enthalten. Außerdem ist es dann nicht notwendig, die Tabellenüberschrift auszugeben. An das Lesen des Anwenderstammsatzes schließt sich daher eine Anweisung an, welche die Zahl der Einträge in der Tabelle `TADIR` für den aktuellen Anwender ermittelt und im Feld `f_count` ablegt.

```
SELECT COUNT(*) FROM tadir
    INTO f_count
    WHERE author = usr03-bname.
```

Nun kann mit dem Funktionsbaustein `START_FORM` die Ausgabe eines neuen Formulars eingeleitet werden. Alle notwendigen Angaben bezüglich des Formulars erfolgten schon bei `OPEN_FORM`, so dass hier keinerlei Parameter übergeben werden müssen.

```
  CALL FUNCTION 'START_FORM'
    EXCEPTIONS
      form     = 1
      format   = 2
      unended  = 3
      unopened = 4
      unused   = 5
      OTHERS   = 6.
  IF sy-subrc <> 0.
    WRITE: / 'OPEN_FORM:', sy-subrc.
    EXIT.
  ENDIF.
```

Für beide Ausgabe-Varianten sind die Inhalte der Fenster für Adresse und Absender identisch. Diese beiden Fenster können Sie unmittelbar nach dem Start des Formulars ausgeben. Beim Aufruf von `WRITE_FORM` müssen Sie lediglich den Namen der beiden Fenster (`ADR` und `ABS`) als Parameter übergeben. Diese beiden Fenster enthalten nur einen einzigen Text ohne Namen, so dass diese Form des Aufrufs eindeutig ist.

```
CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      window                   = 'ADR'
    EXCEPTIONS
      element                  = 1
      function                 = 2
      type                     = 3
      unopened                 = 4
      unstarted                = 5
      window                   = 6
      bad_pageformat_for_print = 7
      OTHERS                   = 8.
  IF sy-subrc <> 0.
    WRITE: / 'WRITE_FORM (ADDR):', sy-subrc.
    EXIT.
  ENDIF.

  CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      window                   = 'ABS'
    EXCEPTIONS
      element                  = 1
      function                 = 2
      type                     = 3
      unopened                 = 4
      unstarted                = 5
      window                   = 6
```

```
      bad_pageformat_for_print = 7
      OTHERS                   = 8.
  IF sy-subrc <> 0.
    WRITE: / 'WRITE_FORM (ABS):', sy-subrc.
    EXIT.
  ENDIF.
```

Das Feld `f_count` enthält die Anzahl der `TADIR`-Objekte des aktuellen Anwenders. Sollte diese Zahl größer als Null sein, so ist das ausführliche Dokument mit einem einleitenden Text, der Tabelle der Objekte und einer entsprechenden Tabellenüberschrift auszugeben. Da diese Ausgaben in das `MAIN`-Fenster erfolgen, müssen Sie nicht nur das Fenster, sondern auch den Fensterbereich (Parameter `TYPE`) und die Bezeichnung des gewünschten Textelementes (Parameter `ELEMENT`) angeben. Zunächst erfolgt die Ausgabe eines erläuternden Textes.

```
  IF f_count > 0.
    CALL FUNCTION 'WRITE_FORM'
      EXPORTING
        element                  = 'INTRO'
        function                 = 'SET'
        type                     = 'BODY'
        window                   = 'MAIN'
      EXCEPTIONS
        element                  = 1
        function                 = 2
        type                     = 3
        unopened                 = 4
        unstarted                = 5
        window                   = 6
        bad_pageformat_for_print = 7
        OTHERS                   = 8.
    IF sy-subrc <> 0.
      WRITE: / 'WRITE_FORM (INTRO):', sy-subrc.
      EXIT.
    ENDIF.
```

Nach dem Text folgt die Tabelle. Diese ist mit einem Tabellenkopf zu versehen. Dieser Kopf steht auf den Folgeseiten im `TOP`-Bereich des Fensters. Auf der ersten Seite allerdings ist er erst nach dem Textelement `INTRO` auszugeben. Dieses Verhalten wird durch den Inhalt des Flags `F_HEADLINE` gesteuert. Ist dieses Flag nicht gesetzt, erfolgt die Ausgabe der Tabellenüberschrift in den `BODY`-Bereich des `MAIN`-Fensters.

```
    IF f_headline = space.
      CALL FUNCTION 'WRITE_FORM'
        EXPORTING
          element                = 'TABLEHEAD'
          function               = 'SET'
```

```
      type                       = 'BODY'
      window                     = 'MAIN'
    EXCEPTIONS
      element                    = 1
      function                   = 2
      type                       = 3
      unopened                   = 4
      unstarted                  = 5
      window                     = 6
      bad_pageformat_for_print   = 7
      OTHERS                     = 8.
  IF sy-subrc <> 0.
    WRITE: / 'WRITE_FORM (TABLEHEAD/BODY):', sy-subrc.
    EXIT.
  ENDIF.

  f_headline = 'X'.
ENDIF.
```

Auf allen anderen Seiten wird die Überschrift in den TOP-Bereich geschrieben. Dabei ist es nicht unbedingt erforderlich, nochmals das F_HEADLINE-Flag zu testen. Auf der ersten Seite führt die folgende Anweisung nicht zu einer Ausgabe im Dokument, da Ausgaben in den TOP-Bereich wirkungslos sind, wenn bereits eine Ausgabe in den BODY-Bereich erfolgte. Die Ausgabe wird allerdings für die folgenden Seiten vorgemerkt und erscheint nach einem Seitenwechsel automatisch.

```
CALL FUNCTION 'WRITE_FORM'
  EXPORTING
    element                  = 'TABLEHEAD'
    function                 = 'SET'
    type                     = 'TOP'
    window                   = 'MAIN'
  EXCEPTIONS
    element                  = 1
    function                 = 2
    type                     = 3
    unopened                 = 4
    unstarted                = 5
    window                   = 6
    bad_pageformat_for_print = 7
    OTHERS                   = 8.
IF sy-subrc <> 0.
  WRITE: / 'WRITE_FORM (TABLEHEAD/TOP):', sy-subrc.
  EXIT.
ENDIF.
```

An die Ausgabe der Tabellenüberschrift schließt sich eine SELECT-Schleife über die Tabelle TADIR an. In dieser Schleife listet ein weiterer Aufruf der WRITE_FORM-Funktion die einzelnen Objekte auf. Die Art des Aufrufs und die Parameter unterscheiden sich prinzipiell nicht von den vorangegangenen Kommandos.

```
SELECT * FROM tadir
  WHERE author = usr03-bname
  ORDER BY PRIMARY KEY.

  CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      element                  = 'TABLELINE'
      type                     = 'BODY'
      window                   = 'MAIN'
    EXCEPTIONS
      element                  = 1
      function                 = 2
      type                     = 3
      unopened                 = 4
      unstarted                = 5
      window                   = 6
      bad_pageformat_for_print = 7
      OTHERS                   = 8.
  IF sy-subrc <> 0.
    WRITE: / 'WRITE_FORM (TABLELINE):', sy-subrc.
    EXIT.
  ENDIF.

ENDSELECT.
```

Die folgende ELSE-Anweisung gehört zur Auswertung der Anzahl der TADIR-Objekte für den aktuellen Anwender. Ist diese Zahl gleich Null, informiert ein entsprechender Text über diese Tatsache. Andere Informationen werden nicht ausgegeben.

```
ELSE.                             "IF f_count > 0.
  CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      element                  = 'NOELEMENTS'
      function                 = 'SET'
      type                     = 'BODY'
      window                   = 'MAIN'
    EXCEPTIONS
      element                  = 1
      function                 = 2
      type                     = 3
      unopened                 = 4
```

```
        unstarted                = 5
        window                   = 6
        bad_pageformat_for_print = 7
        OTHERS                   = 8.
    IF sy-subrc <> 0.
      WRITE: / 'WRITE_FORM (NOELEMENTS):', sy-subrc.
      EXIT.
    ENDIF.

  ENDIF.
```

Nach Auswertung eines Anwenders wird das aktuelle Formular abgeschlossen. Dies erfolgt durch Aufruf des Funktionsbausteins `END_FORM`. Parameter sind nicht erforderlich.

```
  CALL FUNCTION 'END_FORM'
    EXCEPTIONS
      unopened                 = 1
      bad_pageformat_for_print = 2
      OTHERS                   = 3.
  IF sy-subrc <> 0.
    WRITE: / 'END_FORM:', sy-subrc.
    EXIT.
  ENDIF.

ENDSELECT.
```

Die vorangegangene `ENDSELECT`-Anweisung gehört zur Schleife über die Tabelle `USR03` mit den Anwenderstammsätzen. Nach Auswertung aller Anwender schließen Sie den Ausgabeauftrag durch Aufruf von `CLOSE_FORM` ab. Auch dieser Funktionsbaustein kann ohne Parameter aufgerufen werden.

```
CALL FUNCTION 'CLOSE_FORM'
  EXCEPTIONS
    unopened                 = 1
    bad_pageformat_for_print = 2
    OTHERS                   = 3.
IF sy-subrc <> 0.
  WRITE: / 'CLOSE_FORM:', sy-subrc.
  EXIT.
ENDIF.

WRITE: / 'END'.
```

Das Programm ist damit abgeschlossen. Sie können es nun ausführen. Beachten Sie in diesem Zusammenhang bitte die anschließenden Übungen.

An dieser Stelle soll noch auf eine Besonderheit hingewiesen werden. Die automatische Ersetzung von Platzhaltern erfolgt bis einschließlich Release 3.1x nur für Felder, die auf dem Data Dictionary beruhen, also für Felder von Tabellen

und Strukturen. Erst ab Release 4.0 ist es möglich, auch die Inhalte programminterner Felder automatisch zu übernehmen. Trotzdem ist es nicht notwendig, in älteren Versionen Formulare innerhalb einer SELECT-Schleife zu erzeugen. Alternativ können Sie die bereits erwähnten EXTRACT-Datenbestände nutzen, sofern diese mit Dictionary-Feldern zusammenarbeiten. Eines der Haupteinsatzgebiete der EXTRACT-Anweisung sind daher Druckreports zur Massendatenausgabe.

Übungen

1. Führen Sie das Programm aus. Schränken Sie dazu gegebenenfalls die Zahl der auszuwertenden Anwender durch ein geeignetes Selektionskriterium ein.
2. Drucken Sie ein Formular aus der Bildschirmansicht heraus.
3. Ändern Sie die Vorgabe für das Ausgabegerät in PRINTER und aktivieren Sie den Parameter dialog. Drucken Sie das oder die Formulare. Schalten Sie die Option SOFORT AUSGEBEN im Druckdialog ab, falls markiert. Wechseln Sie danach zur Ausgabesteuerung. Zeigen Sie dort den Inhalt des Spool-Auftrags an und drucken ihn aus.

3.11.3 Wichtige Attribute und Kommandos

Während der Definition des Formulars wurden überwiegend Standardeinstellungen übernommen. Einige wichtige Attribute erlauben es, das Verhalten des Formulars zu modifizieren. Nachfolgend sollen einige dieser Attribute näher erläutert werden.

Fenstertypen

Fenster besitzen einen Typ, der die Art und Weise ihrer Aufbereitung und Darstellung bestimmt. Nicht alle Typen können für jedes Fenster gesetzt werden.

CONST

Fenster dieses Typs besitzen auf allen Seiten, auf denen sie erscheinen, eine identische Größe. Der Inhalt wird nur einmal zu Beginn der Verarbeitung des Formulars aufbereitet. Durch diesen Fenstertyp erreichen Sie eine auf allen Seiten identische Darstellung des Fensters.

Wenn Sie das Fenster mit Daten füllen, gehen alle Informationen, die sich nicht im Fenster darstellen lassen, verloren.

VAR

Fenster des Typs VAR werden vom Formularprozessor beim Erstellen jeder Seite neu verarbeitet. Sie können daher auf den verschiedenen Seiten unterschiedliche Größen aufweisen. Allerdings gehen auch hier alle Eingaben verloren, welche die aktuelle Größe des Fensters überschreiten.

MAIN

Der Fenstertyp MAIN ist der einzige, bei dessen Verarbeitung ein Seitenvorschub ausgelöst wird, falls das aktuelle Fenster auf einer Seite vollständig gefüllt ist. Beim Erstellen eines Seitenvorschubes wird unter Umständen auch die Seitendefinition gewechselt. Falls die neue Seite kein MAIN-Fenster enthält und als Folgeseite wiederum dieselbe Seitendefinition benutzt wird, entsteht eine Endlosschleife. Aus diesem Grund sollte jede Seite ein MAIN-Fenster besitzen.

Innerhalb des MAIN-Fensters existieren drei Teilebereiche, TOP, BODY und BOTTOM. Die Inhalte der Bereiche TOP und BOTTOM erscheinen, wenn Sie einmal definiert wurden, automatisch auf allen Seiten. Sie werden beispielsweise für Tabellenüberschriften und Fußzeilen genutzt. Die Auswahl der Bereiche erfolgt durch Parameter des Funktionsbausteins WRITE_FORM.

Symbole

Im SAPScript-Text können Symbole enthalten sein. Symbole sind Bezeichner, die in das „&"-Zeichen eingeschlossen werden. Zur Laufzeit findet eine Symbolersetzung statt. Das Symbol wird durch seinen Wert ersetzt. Falls das Symbol dem Namen eines Dictionary-Feldes oder eines programminternen Feldes des Druckprogramms entspricht, wird es durch den Inhalt dieses Feldes ersetzt. Außerdem existieren vordefinierte Symbole, beispielsweise für diverse Überschriften, Seitenzahlen und Datums- sowie Zeitangaben. Die Symbole können im Editor zur Textelementpflege mit der Menüfunktion EINBINDEN | SYMBOLE ermittelt und eingefügt werden.

Absatzformate

Innerhalb einer Formulardefinition stellt Ihnen das R/3-System einige vorgefertigte Absatzformate zur Verfügung. Diese Formate werden in der ersten Spalte links neben dem eigentlichen Text eingetragen. Tabelle 3.46 zeigt Ihnen die Symbole und die Namen dieser Formate. Es schließt sich eine kurze Beschreibung für jedes Format an.

Symbol	Name
*	Default-Absatz
/	Neue Zeile
/:	SAPScript-Steuerkommando
/*	Kommentar

Tabelle 3.46
Vordefinierte Absatzformate

Symbol	Name
=	Langzeile
/=	Langzeile mit Zeilenumbruch
(	Rohzeile
/(	Rohzeile mit Zeilenumbruch
>x	Fixzeile
/E	Textelement

Tabelle 3.46
Vordefinierte Absatzformate (Fortsetzung)

Default-Absatz

Im den Kopfdaten eines Formulars müssen Sie ein Default-Absatzformat festlegen. Dieses Format muss vorher von Ihnen definiert werden. Im Formular können Sie sich mit dem Stern „*“ als Symbol auf dieses Format beziehen. Jede so gekennzeichnete Zeile leitet einen neuen Absatz ein, vor Ausgabe der Zeile wird also ein Zeilenvorschub ausgeführt.

Neue Zeile

Die Textzeilen innerhalb eines Absatzes werden fortlaufend formatiert. Falls Sie einen Zeilenumbruch erzwingen möchten, können Sie dies mit dem Zeichen „/“ tun. Die aktuellen Formatierungseigenschaften bleiben dabei erhalten.

SAPScript-Steuerkommando

Innerhalb des Textes zu einem Formularfenster können Sie neben reinem Text auch einige Steueranweisungen ablegen. Diese Kommandos werden durch die Zeichen „/:“ in der Formatspalte gekennzeichnet. Den Steuerkommandos ist ein eigener Abschnitt gewidmet.

Kommentar

Mit den Zeichen „/*“ fügen Sie einen Kommentar ein. Das Zeichen gilt nur für die aktuelle Zeile. Soll der Kommentar mehrere Zeilen umfassen, so ist das Kommentarsymbol für jede Zeile anzugeben.

Langzeile

Für diese Zeile führt der SAPScript-Editor keine Formatierung aus. Das bedeutet, dass auch kein Zeilenumbruch stattfindet. Der Inhalt dieser Zeile wird somit direkt an das Ende der vorangegangenen Zeile angehängt. Dabei wird auch kein trennendes Leerzeichen eingefügt, es sei denn, die Langzeile beginnt mit einem Leerzeichen.

Langzeile mit Zeilenumbruch

Der Text wird ebenso behandelt wie bei einer Langzeile. Allerdings findet vor Ausgabe des Textes ein Zeilenumbruch statt.

Rohzeile

Diese Zeile wird zwar formatiert, allerdings findet keine Aufbereitung durch den so genannten SAPScript-Composer statt. Dieses Werkzeug verarbeitet z. B. Zeichenformate innerhalb einer Zeile, Symbole, Tabulator- und Maskierungszeichen und Hypertextverweise. Stattdessen werden die betreffenden Zeichen unverändert ausgegeben.

Ebenso wie bei einer Langzeile wird der Text unmittelbar, also ohne trennendes Leerzeichen, an das Ende der vorangegangenen Zeile angehängt. Falls ein solches Trennzeichen gewünscht ist, muss die Rohzeile mit einem Leerzeichen beginnen.

Rohzeile mit Zeilenumbruch

Die Formatierung erfolgt ebenso wie bei der Rohzeile, allerdings wird vor Ausgabe des Textes ein Zeilenumbruch ausgeführt.

Fixzeile

Diese Zeile ist weder eingabebereit noch editier- oder trennbar. Die ersten beiden Stellen des eigentlichen Textes werden wiederum als Absatzformat ausgewertet, die darauf folgenden als normaler Text ausgegeben.

Der Buchstabe „X“ im Formatsymbol wird durch eine Ziffer oder einen beliebigen Buchstaben ersetzt, damit unterschiedliche Fixzeilen definiert werden können.

Textelement

In der so gekennzeichneten Zeile steht der Name eines Textelements. Ein Textelement umfasst den gesamten Text bis zum nächsten Textelement-Namen oder aber bis zum Ende des Textes. Der Inhalt eines Textelements erscheint nur dann im Formular, wenn das Textelement durch den Funktionsbaustein `WRITE_FORM` explizit angesprochen wird.

Steuerkommandos

Das Attribut „/:“ ermöglicht es, in den Text eines Formulars Steuerkommandos einzufügen. Dieselben Kommandos können Sie auch durch den Funktionsbaustein `CONTROL_FORM` zur Laufzeit auslösen. Die Steuerkommandos stehen jeweils in einer eigenen Zeile. Dieser Abschnitt stellt Ihnen die zur Verfügung stehenden Kommandos näher vor.

NEW-PAGE

Dieses Kommando löst einen Seitenumbruch aus. Die gerade bearbeitete Seite wird dadurch abgeschlossen. Als Folgeseite dient im Normalfall die innerhalb der Formulardefinition statisch angegebene Folgeseite. Sie können diese Vorgabe aber auch überschreiben, in dem Sie die von Ihnen gewünschte Seite als Parameter an `NEW-PAGE` übergeben.

Beispiele:

```
/: NEW-PAGE
/: NEW-PAGE NEXT2
```

PROTECT

Falls Sie Absätze vor einem automatischen Zeilenumbruch schützen möchten, können Sie diese in die beiden Kommandos `PROTECT` und `ENDPROTECT` einschließen. Reichte der zur Verfügung stehende Platz auf der aktuellen Seite nicht aus, um den eingeschlossenen Text auszugeben, wird ein Seitenvorschub ausgeführt. Nach dem Seitenvorschub wird der Text auf jeden Fall ausgegeben, auch wenn er so lang ist, dass er nicht auf eine leere Seite passt.

Syntax:

```
/: PROTECT
 text/: ENDPROTECT
```

NEW-WINDOW

Auf einer Formularseite können bis zu 99 Fenster des Typs `MAIN` enthalten sein. Erforderlich ist das beispielsweise dann, wenn mehrspaltiger Text gedruckt werden soll oder Etiketten zu bedrucken sind. Die Fenster werden durch eine eindeutige Nummer im Bereich von 0 bis 98 unterschieden. Mit dem Kommando `NEW-WINDOW` schalten Sie in das jeweils nächste Fenster mit dem Typ `MAIN` um.

Syntax:

```
/: NEW-WINDOW
```

DEFINE

Mit dem Kommando `DEFINE` können Sie innerhalb des Quelltextes Textsymbolen einen Wert zuweisen. Es existieren zwei Zuweisungsoperatoren. Das einfache Gleichheitszeichen „=“ löst eine einstufige Symbolersetzung aus. Sind im zuzuweisenden Wert weitere Textsymbole erhalten, werden diese erst zur Laufzeit ersetzt. Der Zuweisungsoperator „:=“ hingegen ersetzt erst alle Symbole innerhalb der Werte (rechte Seite der Zuweisung) und führt erst dann die Zuweisung im Text aus.

Syntax:

```
/: DEFINE &symbolname& = 'value'
```

SET DATE MASK

Die Aufbereitung von Datumsfeldern kann mit dem SAPScript-Kommando `SET DATE MASK` definiert werden. Nach Ausführung dieses Kommandos werden alle Datumsfelder mit dieser Darstellung ausgegeben.

Syntax:

```
/: SET DATE MASK = 'datemask'
```

In der Datumsmaske können die in Tabelle 3.47 aufgeführten Schablonen verwendet werden:

Maskenzeichen	Beschreibung
DD	Tag (zweistellig)
DDD	Tagesname abgekürzt
DDDD	Tagesname ausgeschrieben
MM	Monat (zweistellig)
MMM	Monatsname abgekürzt
MMMM	Monatsname ausgeschrieben
YY	Jahr (zweistellig)
YYYY	Jahr (vierstellig)
LD	Tag (aufbereitet wie bei Zusatzoption L)
LM	Monat (aufbereitet wie bei Zusatzoption L)
LY	Jahr (aufbereitet wie bei Zusatzoption L)

Tabelle 3.47
Maskenzeichen für Datumsmaske

Alle anderen Zeichen in der Maske werden als Text interpretiert und entsprechend übernommen.

Die Standardeinstellung aktivieren Sie durch Zuweisen eines Leerstrings. Die sprachabhängigen Texte für die Tages- und Monatsnamen sind in der Tabelle `TTDTG` hinterlegt. Im Normalfall müssen Sie diese Einträge nie ändern.

SET TIME MASK

Analog zur Datumsformatierung können Sie auch Zeitangaben aufbereiten. Dazu verwenden Sie das Kommando `SET TIME MASK`. Die Syntax des Kommandos lautet

```
/: SET TIME MASK = 'time mask'
```

In der Zeitmaske besitzen die in Tabelle 3.48 aufgeführten Platzhalter eine besondere Bedeutung. Alle anderen Zeichen werden als Text interpretiert und unverändert ausgegeben.

Zeichen	Bedeutung
HH	Stunden (zweistellig)
MM	Minuten (zweistellig)
SS	Sekunden (zweistellig)

Tabelle 3.48
Maskenzeichen für Zeitangaben

Alle anderen Zeichen in der Maske werden als Text interpretiert und entsprechend ausgegeben.

SET COUNTRY

Mit diesem Kommando veranlassen Sie die länderspezifische Aufbereitung spezieller Felder und Werte. Das trifft z.B. für die Darstellung des Datums, des Dezimalzeichens oder des Tausender-Punktes zu. Bei der Erstellung von Reports etc. werden dazu die Einstellungen im Benutzerstammsatz ausgewertet. Da der Formulardruck in einem Arbeitsgang durchaus Dokumente in verschiedenen Sprachen erzeugen kann (beispielsweise in der Sprache des Empfängers), muss die sprachspezifische Aufbereitung gegebenenfalls auch zur Laufzeit gesetzt werden können. Der Länderschlüssel kann daher sowohl über einen in Hochkommas eingefassten Wert als auch in Form eines SAPScript-Symbols angegeben werden, wobei letztere Variante natürlich die häufiger verwendete ist. Die Syntax des Kommandos lautet:

```
/: SET COUNTRY country key
```

SET SIGN

Mit Hilfe dieses Kommandos legen Sie fest, ob ein Vorzeichen rechts oder links von dieser Zahl erscheint. Die Position wird im Kommando durch die Worte `LEFT` oder `RIGHT` bestimmt, wodurch sich die beiden folgenden Syntax-Varianten ergeben:

```
/: SET SIGN LEFT
```

bzw.

```
/: SET SIGN RIGHT
```

RESET

Dieses Kommando setz die aktuelle Absatznummer auf den Startwert zurück. Dabei muss der Name des jeweiligen Absatzformates als Parameter übergeben werden.

```
/: RESET paragraph
```

INCLUDE

Die Textelemente einer Fensterdefinition gelten nur innerhalb eines Formulars. Die Wiederverwendung von Textbausteinen ist so nicht möglich. Falls Sie wiederverwendbare Bausteine benutzen möchten, die in mehrere Formulare eingefügt werden können, so müssen Sie diese als normale SAPScript-Texte anlegen. In den Formularen fügen Sie diese Texte dann mit der Anweisung `INCLUDE` ein. Die Syntax dieses Kommandos ist etwas umfangreicher, da einige Optionen möglich sind:

```
/: INCLUDE name
  [OBJECT object]
  [ID ident]
  [LANGUAGE language]
  [PARAGRAPH paragraph]  [NEW-PARAGRAPH npar].
```

Der Name des einzufügenden Textes muss auf jeden Fall angegeben werden. Er muss nur dann in Hochkommas stehen, wenn er Leerzeichen enthält. Es ist möglich, ihn durch einen Platzhalter für einen Feldnamen zu übergeben. Die übrigen Parameter des `INCLUDE`-Kommandos sind optional.

Der Formularprozessor sucht die sprachabhängigen Bestandteile eines Formulars entweder in der Anmeldesprache oder aber in der beim Aufruf der SAPScript-Funktionsbausteine angegebenen Sprache. Dies gilt auch für Includes. Sie können diese Vorgabe aber durch die Verwendung der Option `LANGUAGE` überschreiben. Dies gilt auch für die Formatangabe, die durch den Parameter `PARAGRAPH` festgelegt wird. Dieses Kommando ersetzt in allen Textteilen, die mit dem Standardformat („*“) formatiert sind, dieses Format durch das als Parameter übergebene Format. Ähnlich arbeitet `NEW-PARAGRAPH`. Allerdings wird mit diesem Kommando nur die erste Zeile eines Textes umformatiert. Nur dann, wenn keine `FORMAT`-Option benutzt wird, werden alle Absätze mit Standardformat ebenfalls neu formatiert.

Die Option `OBJECT` ist im Zusammenhang mit der reinen Formularerstellung nicht von Bedeutung. Mit dem Parameter `ID` legen sie einen weiteren Teil des Textnamens fest.

STYLE

Mit diesem Kommando wechseln Sie den Ausgabestil. Die Syntax ist sehr einfach:

```
/: STYLE style
```

Falls Sie für *style* den Stern eintragen, schaltet der Formularprozessor auf den ursprünglichen Stil zurück.

ADDRESS

Innerhalb der Kommandoklammer `ADDRESS - ENDADDRESS` stehende Adressdaten werden nach den Normen des Empfängerlandes aufbereitet. Dieses Land ist im Parameter `COUNTRY` festzulegen.

```
/: ADDRESS [COUNTRY country] ...
  ...
/: ENDADDRESS
```

TOP

Mit dem Kommando `TOP` definieren Sie einen Text, der ab Zeitpunkt der Definition automatisch am Anfang jedes Hauptfensters erscheint. Dieser Text kann durchaus mehrzeilig sein.

```
/: TOP  text
/: ENDTOP
```

BOTTOM

Mit diesem Kommando erzeugen Sie einen Text, der automatisch als Fußzeile erscheint.

```
/: BOTTOM      text
/: ENDBOTTOM
```

IF

Innerhalb der Textdefinitionen können logische Ausdrücke ausgewertet werden. Dazu dienen wie im ABAP die Anweisungen `IF`, `ELSE`, `ELSEIF` und `ENDIF`. Die verwendbaren Operatoren finden Sie in Tabelle 3.49. Die Syntax ähnelt der bekannten ABAP-Anweisung:

```
/: IF condition      ... /: ELSEIF condition      ... /: ELSE
...      /: ENDIF
```

Operations-zeichen	Textsymbol für Operationszeichen	Beschreibung
=	EQ	Gleich
<	LT	Kleiner
>	GT	Größer
<=	LE	Kleiner oder gleich
>=	GE	Größer oder gleich
<>	NE	Ungleich
	NOT	Verknüpfungsoperator Negation
	AND	Verknüpfungsoperator UND
	OR	Verknüpfungsoperator ODER

Tabelle 3.49
Operatoren des IF-Kommandos

CASE

Die CASE-Anweisung zur Fallunterscheidung kann ebenfalls eingesetzt werden. Auch hier entspricht die Syntax im Wesentlichen der ABAP-Anweisung:

```
/: CASE symbol   /: WHEN value_1      ...  /: WHEN value_2
...              /: WHEN value_n     ...   /: WHEN OTHERS
...            /: ENDCASE
```

PERFORM

Der Formularprozessor kann ABAP-Unterprogramme aufrufen, um zusätzliche Auswertungen oder Berechnungen mit Textinhalten durchzuführen. Der Datenaustausch erfolgt über Symbole, also Platzhalter für Feldnamen oder vordefinierte Symbole. Auch die Syntax dieses Kommandos ähnelt der des entsprechenden ABAP-Kommandos:

```
/: PERFORM subroutine IN PROGRAM program/: USING &invar1&/:
USING &invar2&   .../: CHANGING &outvar1&/: CHANGING
&outvar2&   .../: ENDPERFORM
```

PRINT-CONTROL

Innerhalb der Transaktion SPAD können Sie spezielle Print-Controls für jeweils einen Drucker definieren. Print-Controls sind Steuerzeichen, mit denen Sie am Drucker spezielle Aktionen auslösen können. Innerhalb des Formulars veranlassen Sie mit dem Kommando PRINT-CONTROL das Aussenden einer solchen

Steuerzeichenfolge. Da der Formularprozessor diese Steuerzeichen nicht auswertet, kann durch unsachgemäßen Einsatz die gesamte Formatierung des Textes gestört werden.

Die Syntax des Kommandos ist sehr einfach:

```
/: PRINT-CONTROL name
```

BOX

Durch Verwendung des `BOX`-Kommandos können Sie einen Rahmen zeichnen und die umschlossene Fläche schattieren. Dieses Kommando wird allerdings nicht von allen Druckertreibern verarbeitet. Die Syntax des Kommandos lautet:

```
/: BOX
  [XPOS xpos] [YPOS ypos]
  [WIDTH width] [HEIGTH height]
  [FRAME frame]
  [INTENSITY intensity]
```

Die beiden Parameter `xpos` und `ypos` legen die Position der linken oberen Ecke des Rahmens fest. Dabei ist die Wirkung des Kommandos `POSITION` zu berücksichtigen, das den Nullpunkt innerhalb des Fensters festlegt. Dieser muss nicht der linken oberen Ecke des Fensters entsprechen. Zur Ermittlung der exakten Position müssen daher die Werte von `xpos` und `xoffset` bzw. `ypos` und `yoffset` addiert werden.

Mit `width` und `heigth` bestimmen Sie Breite und Höhe des Rahmens, mit frame die Rahmendicke. Ein Wert von 0 für `frame` bedeutet, dass kein Rahmen gezeichnet wird.

Falls die Angaben für `xpos`, `ypos`, `width` oder `heigth` fehlen, werden Initialwerte bzw. die mit `POSITION` und `SIZE` gesetzten Werte übernommen.

Im Unterschied zu anderen Kommandos bestehen die Parameter außer `intensity` aus einem Wert und einer Maßangabe. Die möglichen Einheiten entnehmen Sie bitte Tabelle 3.50. Dezimalzahlen sind als Zeichenketten zu notieren, also in einfache Anführungsstriche einzuschließen.

Der Parameter `intensity` legt den Grad der Schwärzung der Fläche innerhalb der Box fest. Der Wertebereich liegt zwischen 0 und 100.

Einheit	Bezeichnung	Umrechnung
CH	Zeichen	
CM	Zentimeter	
IN	Zoll	2,54 CM

Tabelle 3.50
Maßeinheiten und Umrechnungen

Einheit	Bezeichnung	Umrechnung
LN	Zeile	
MM	Millimeter	
PT	Punkt	1/72 IN
TW	Twip	1/20 PT

Tabelle 3.50
Maßeinheiten und Umrechnungen (Fortsetzung)

Beispiele:

```
/: BOX 5 CM 5 CM 10 CM 10 CM FRAME 10 TW INTENSITY 10
/: BOX WIDTH '5.5' CM HEIGHT 1 CM FRAME 10 TW
/: BOX WIDTH 5 CM HEIGHT 5 CM INTENSITY 10/: BOX FRAME 20 TW
```

POSITION

Im Initialzustand ist der Koordinatenursprung für das BOX-Kommando die linke obere Ecke eines Fensters. Mit dem Kommando POSITION können Sie diesen Ursprung verschieben. Dabei können Sie durch Angabe von vorzeichenlosen Werten absolut bzw. mit vorzeichenbehafteten Werten relativ zum letzten Wert positionieren.

Die Parameter WINDOW bzw. PAGE werden als Schalter und damit ohne weitere Zusätze benutzt. Sie setzen den Ursprung auf die linke obere Ecke des Fensters bzw. der Seite.

Syntax:

```
/: POSITION [XORIGIN xoffset] [YORIGIN yoffset] [WINDOW]
[PAGE]
```

SIZE

Mit diesem Kommando können Sie Vorgabewerte für die Fensterhöhe und -breite setzen. Wie beim POSITION-Kommando können Sie dabei absolut bzw. relativ positionieren. Die Schalter WINDOW und PAGE setzen wiederum Initialwerte.

Syntax:

```
/: SIZE [WIDTH width] [HEIGHT heigth] [WINDOW] [PAGE]
```

HEX, ENDHEX

Die zwischen HEX und ENDHEX stehenden Zeichen werden als Hexadezimalwerte interpretiert und unverändert an den Drucker gesendet.

SUMMING

Dieses Kommando funktioniert ähnlich wie das gleichnamige ABAP-Kommando. Es wird nur einmal notiert. Anschließend wird es bei jeder Verarbeitung des angegebenen Quellfeldes ausgeführt. Die Syntax des Kommandos lautet:

```
/: SUMMING program symbol INTO summing symbol
```

3.11.4 Funktionsbausteine im Detail

Für die Formularausgabe sind einige Funktionsbausteine von Bedeutung, die nachfolgend detailliert beschrieben werden sollen.

OPEN_FORM

Der Funktionsbaustein `OPEN_FORM` leitet den Formulardruck ein. Er muss vor allen anderen Bausteinen des Formulardrucks aufgerufen werden. Die Hauptaufgabe des Bausteins besteht darin, einen geeigneten Spool-Auftrag zu erzeugen bzw. einen vorhandenen Auftrag zu finden. Die dazu notwendigen Informationen werden entweder in einem Dialog vom Anwender erfragt oder müssen beim Aufruf des Bausteins übergeben werden. Es ist weiterhin möglich, aber nicht unbedingt erforderlich, bereits beim Aufruf von `OPEN_FORM` das zu verwendende Formular anzugeben.

Die beiden am häufigsten verwendeten Parameter des Bausteins sind `DIALOG` und `OPTIONS`. Mit dem Wert für `DIALOG` steuern Sie den Aufruf des Popups zur Erfassung der Druckparameter. Im Parameter `OPTIONS`, der die Struktur `ITCPO` besitzt, übergeben Sie Vorgabewerte für die Druckparameter. Die endgültigen Druckparameter ergeben sich aus den Werten im Parameter `OPTIONS`, den Voreinstellungen im Benutzerstammsatz und, falls der Dialog freigegeben wurde, aus den Eingaben des Anwenders. Der Baustein liefert im Export-Parameter `RESULT` die gewählten Druckparameter zurück.

Mit dem Inhalt des Parameters `DEVICE` legen Sie das Ausgabemedium fest. Der Default-Wert ist `PRINTER` für Druckausgaben. Mit dem Wert `SCREEN` im Parameter `DEVICE` wird die Erzeugung eines Spool-Auftrages unterdrückt und dafür das Dokument am Bildschirm dargestellt. Im letzteren Fall können Sie mit dem Parameter `APPLICATION` die Oberfläche des Dynpros beeinflussen, in dem das Dokument erscheint. Weitere mögliche Werte für `DEVICE` sind `TELEX`, `TELEFAX` und `ABAP`. Falls das zu benutzende Formular bereits bei `OPEN_FORM` vorgegeben werden soll, erfolgt dies mit den beiden Parametern `FORM` und `LANGUAGE`.

Falls das gewünschte Formular nicht in der angegebenen Sprache existiert, verwendet das System ggf. das Formular in einer anderen Sprache. Im Export-Parameter `LANGUAGE` erfolgt daher die Rückgabe der aktuellen Formularsprache.

Druckaufträge können alternativ zur Ausgabe auf einem Drucker auch in einem optischen Archiv abgelegt werden. In diesem Zusammenhang sind die Parame-

ter ARCHIVE_INDEX und ARCHIVE_PARAMS von Bedeutung. Sie sollen hier nicht näher beschrieben werden, da die Archivierung ein eigenständiges Arbeitsgebiet darstellt.

Im Dialog zur Pflege der Druckparameter hat der Anwender die Möglichkeit zum Abbruch. Außerdem besteht die Möglichkeit, dass ungültige Eingaben einen Fehler auslösen. Es ist deshalb unbedingt erforderlich, die durch den Funktionsbaustein ausgelösten Ausnahmen auszuwerten. Es ist nicht sinnvoll, den Formulardruck fortzusetzen, falls der Baustein mit Erzeugung einer Ausnahme abbricht. Während der Programmentwicklung liefert der Wert der Ausnahme oft hilfreiche Informationen zur Fehlereingrenzung.

START_FORM

Aufgabe des Funktionsbausteins START_FORM ist es, die Bearbeitung eines neuen Formulars einzuleiten. Bevor Sie diesen Funktionsbaustein aufrufen können, müssen Sie mit OPEN_FORM den Formulardruck geöffnet haben. Dazu müssen Sie dem Baustein in den Parametern FORM und LANGUAGE den Namen des Formulars übergeben, falls das nicht schon beim Aufruf von OPEN_FORM geschehen ist. Sie können innerhalb einer OPEN_FORM-CLOSE_FORM-Klammer den Baustein START_FORM mehrfach aufrufen und dabei jedes Mal ein anderes Formular setzen. Falls Sie den Formularnamen schon bei OPEN_FORM angegeben haben, müssen Sie dies bei START_FORM nicht nochmals tun. In diesem Fall leitet der Aufruf von START_FORM die Bearbeitung eines neuen Exemplars des jeweiligen Formulars ein. Auch dieser Baustein liefert im Export-Parameter LANGUAGE die wirklich gewählte Formularsprache zurück.

In einem Formular ist üblicherweise eine Seite als Startseite definiert. Falls Sie von dieser Vorgabe abweichen möchten, können Sie dies durch Übergabe des entsprechenden Seitennamens im Parameter STARTPAGE tun.

In den Textelementen eines Formulars können Platzhalter für Tabellenfelder stehen. Zur Laufzeit werden diese Platzhalter durch den Inhalt der jeweiligen Tabellenfelder ersetzt. Dabei wird im aktuellen Programm gesucht, falls Sie durch den Parameter PROGRAM keine andere Wahl treffen.

WRITE_FORM

Der Funktionsbaustein WRITE_FORM ist für die Ausgabe der im Formular befindlichen Textelemente zuständig. Er kann daher erst dann erfolgreich aufgerufen werden, wenn mit OPEN_FORM der Formulardruck geöffnet und mit START_FORM ein Formular angelegt wurde.

Beim Aufruf von WRITE_FORM ist im Parameter WINDOW stets das Fenster anzugeben, in das eine Ausgabe erfolgen soll. Die Aufteilung des Formulars in verschiedenen Seiten ist für WRITE_FORM nicht von Bedeutung. Ein Seitenvorschub erfolgt entweder beim Zugriff auf ein Fenster, das auf der aktuellen Seite nicht existiert oder aber, wenn das MAIN-Fenster der aktuellen Seite gefüllt ist.

Ein Fenster kann mehrere Textelemente enthalten. Der Bezeichner des Elements ist im Parameter ELEMENT zu übergeben.

Durch Wahl einer Ausgabefunktionen im Parameter FUNCTION legen Sie fest, wie bereits existierender Text behandelt wird. Die Wirkung des Parameters ist vom Fenster abhängig. Für das MAIN-Fenster bewirken die Funktionen SET und APPEND das Anhängen an die vorangegangenen Ausgaben. Die Funktion DELETE ist unwirksam. In allen anderen Fenstern können Sie mit der Funktion SET den gesamten Fensterinhalt neu setzen. Alle eventuell schon vorhandenen Werte gehen verloren. Zum Anfügen muss die Funktion SET benutzt werden. Die Funktion DELETE ist wirksam und löscht den Text im angegebenen Fenster.

Der Parameter TYPE ist nur für Ausgaben im MAIN-Fenster wirksam. Dieses Fenster ist in die drei Bereiche TOP, BODY und BOTTOM unterteilt, die mit dem genannten Parameter gewählt werden müssen.

Der Export-Parameter PENDING_LINES ist für Ausgaben im BOTTOM-Bereich des MAIN-Fensters von Bedeutung. Falls im BOTTOM-Bereich für einen auszugebenen Text nicht ausreichend Platz vorhanden ist, erscheinen die fehlenden Zeichen im BOTTOM-Bereich der nächsten Seite. Unter Umständen müssen Sie die Ausgabe dieser Folgeseite explizit mit NEW_PAGE anstoßen. Das ist z.B. dann notwendig, wenn die normale Textausgabe beendet ist. Ein 'X' im Export-Parameter PENDING_LINES signalisiert Ihnen, dass die Ausgabe von Textteilen im BOTTOM-Bereich noch aussteht.

END_FORM

Nachdem Sie ein Formular durch Aufrufe des Funktionsbausteins WRITE_FORM mit Text gefüllt haben, müssen Sie dieses Formular durch Aufruf von END_FORM abschließen. Erst nach dem Aufruf von END_FORM kann mit START_FORM ein neues Formular geöffnet werden. Nach dem Aufruf von END_FORM sind natürlich keine Ausgaben in das Formular mehr möglich. Der Funktionsbaustein liefert im Export-Parameter RESULT die aktuellen Druckparameter zurück.

CLOSE_FORM

Durch Aufruf von CLOSE_FORM beenden Sie den gesamten Formulardruck. Erst nach Aufruf von CLOSE_FORM erfolgt die Ausgabe in den Druckspool oder den Bildschirm. Falls Sie jeweils ein einzelnes Dokument am Bildschirm ausgeben möchten, müssen Sie für jedes dieser Dokumente eine eigene OPEN_FORM-CLOSE_FORM-Schleife durchlaufen.

In den Parametern RESULT bzw. RDI_RESULT liefert der Baustein die aktuellen Ausgabeparameter zurück.

Falls beim Aufruf von OPEN_FORM in den Druckparametern der Parameter TDGETOTF mit „X“ belegt wurde, erfolgt keine Spool- oder Bildschirmausgabe. In diesem Fall wird in der Tabelle OTFDATA die aufbereitete Ausgabe im OTF-Format zurückgeliefert.

PRINT_TEXT

Dieser Funktionsbaustein bereitet einen Textbaustein für die Ausgabe auf und sendet ihn an das Ausgabegerät.

CONTROL_FORM

Mit diesem Funktionsbaustein können Sie Steuerkommandos zur Laufzeit dynamisch an das Formular übergeben. Dazu übergeben Sie das Kommando im Parameter `COMMAND`. Dabei darf das Absatzkennzeichen für Steuerkommandos („/:“) natürlich nicht übergeben werden. Weitere Parameter existieren für diesen Baustein nicht, als Formular wird das zuletzt mit `START_FORM` geöffnete verwendet.

3.11.5 Druckparameter

Die Erstellung von Formularen ist immer mit der Erzeugung eines Spool-Auftrages verbunden. Dazu muss der Funktionsbaustein `OPEN_FORM` mit entsprechenden Parametern versorgt werden, da dieser Baustein einen Spool-Auftrag erzeugt. Auch innerhalb von Reports ohne Formularverarbeitung ist möglicherweise der Zugriff auf die Druckparameter erforderlich. Die Beschaffung der Druckparameter kann auf unterschiedliche Weise erfolgen. Der Funktionsbaustein `OPEN_FORM` kann, gesteuert durch einen speziellen Parameter, den Standard-Druckdialog des SAP-Systems aufrufen. Der Anwender hat so die Möglichkeit, zur Laufzeit der Anwendung die Druckparameter festzulegen. Dies ist nicht in allen Fällen sinnvoll. Falls der belegerzeugende Report im Batch-Modus läuft, kann der entsprechende Dialog nicht ausgeführt werden. Außerdem ist es durchaus möglich, dass innerhalb einer Anwendung der `OPEN_FORM`-Baustein mehrfach aufgerufen wird. In diesem Fall würde der Dialog für die Druckparameter mehrfach erscheinen. Sie können dem Funktionsbaustein daher eine Datenstruktur mit Druckparametern übergeben und den Dialog-Modus deaktivieren. Die erforderlichen Parameter können Sie im Programm fest vorgeben, über eigene Dialoge (bzw. Report-Parameter) ermitteln oder aber den Standard-Druckdialog selbst aufrufen.

Datenstrukturen

Im Zusammenhang mit den Druckparametern sind zwei Datenstrukturen von Bedeutung. Der Funktionsbaustein `GET_PRINT_PARAMETERS`, mit dem Sie den Standard-Druckdialog aufrufen können, benutzt die Struktur `PRI_PARAMS`, um die Druckparameter zurückzuliefern. Der Funktionsbaustein `OPEN_FORM` erwartet hingegen einen auf der Struktur `ITCPO` beruhenden Eingabeparameter. Beide Strukturen besitzen Felder mit identischem Inhalt, leider aber mit unterschiedlichem Namen. Es ist daher erforderlich, die durch den Aufruf von `GET_PRINT_PARAMETERS` ermittelten Parameter feldweise in die Feldleiste zu übertragen, die als Eingabeparameter für `OPEN_FORM` dient.

Die Tabelle 3.51 zeigt Ihnen zunächst den Aufbau der Struktur PRI_PARAMS.

Feldname	Beschreibung
PDEST	Ausgabegerät
PRCOP	Anzahl der Ausdrucke
PLIST	Name des Spool-Auftrags (Listname)
PRTXT	Text für Deckblatt
PRIMM	‚X': Sofort ausgeben
PRREL	‚X': Löschen nach Ausgabe
PRNEW	‚X': Neuer Spool-Auftrag
PEXPI	Spool-Verweildauer in Tagen
LINCT	Länge der Liste in Zeilen
LINSZ	Breite der Liste in Zeichen
PAART	Aufbereitung der Liste
PRBIG	Selektionen ausgeben
PRSAP	Deckblatt ausgeben
PRREC	Empfänger
PRABT	Abteilung
PRBER	Berechtigung für Zugriff auf Spool-Auftrag
PRDSN	Name des Spool-Datasets
PTYPE	Typ des Spool-Auftrags
ARMOD	Archivierungsmodus
FOOTL	Fußzeile ausgeben
PRCHK	Prüfsumme

Tabelle 3.51
Aufbau der Struktur PRI_PARAMS

Jeder Ausgabeauftrag muss an ein Ausgabegerät (meist einen Drucker) gesendet werden. Das Ausgabegerät bestimmt möglicherweise den Wertevorrat für einige andere Attribute, beispielsweise die maximale Seitenbreite und -länge. Im Benutzerstammsatz kann für diesen Wert eine Vorgabe gepflegt werden, die

das System automatisch übernimmt. Den Namen des Ausgabegerätes finden Sie im Feld PDEST wieder. Im Feld PRCOP können Sie angeben, wie viele Exemplare gedruckt werden sollen.

Spool-Aufträge müssen nicht zwangsweise sofort gedruckt werden. Sehr oft, gerade bei sehr großen Druckaufträgen, wird zunächst nur ein Spool-Auftrag erzeugt, dessen Druck später manuell oder per Batch ausgeführt wird. Mit dem Flag PRIMM legen Sie fest, ob die Ausgabe automatisch sofort nach der Erstellung des Ausgabeauftrags ausgeführt werden soll. Nach der Ausgabe eines Spool-Auftrags ist dieser natürlich abgeschlossen, es können keine weiteren Ausgaben mehr angehängt werden. Diese Eigenschaft steht in Zusammenhang mit dem Flag PRNEW. Ist dieses Flag gesetzt, so wird für jeden Ausgabeauftrag ein neuer Spool-Auftrag angelegt. Ist dieses Flag nicht gesetzt, so sucht das System zunächst nach einem passenden Spool-Auftrag des aktuellen Anwenders, an den die Ausgaben angehängt werden können. Ein neuer Spool-Auftrag wird in diesem Fall nur dann angelegt, wenn kein passender existiert.

Nach der Ausgabe verbleiben Spool-Aufträge für eine gewisse Zeit in der Spool-Liste. Der Zeitraum (in Tagen) wird im Feld PEXPI festgelegt. Nach Ablauf der dort eingetragenen Zeit wird der Spool-Auftrag automatisch gelöscht. Falls es nicht wünschenswert ist, den Spool-Auftrag nach erfolgreicher Ausgabe in der Spool-Liste zu belassen, so kann durch Markieren des Flags PRREL das sofortige Löschen nach der Ausgabe erzwungen werden. Interessant ist dies vor allem für Dokumente oder Listen, deren Inhalt nicht allen Anwendern zugänglich sein soll. Im Normalfall kann jeder Anwender alle Spool-Aufträge einsehen, und zwar nicht nur die Verwaltungsinformationen und Attribute, sondern auch den eigentlichen Inhalt. Bei noch größerem Sicherheitsbedürfnis kann im Feld PRBER der Name einer Berechtigung abgelegt werden, die beim Zugriff auf den Spool-Auftrag überprüft wird.

Innerhalb der Spool-Verwaltung müssen textuelle Informationen über die einzelnen Spool-Aufträge verfügbar sein, die es einem Administrator ermöglichen, einen bestimmten Spool-Auftrag zu finden. Für derartige Informationen steht das Feld PRTXT zur Verfügung.

Bei der Ausgabe von Druckaufträgen auf Druckern, die mehreren Anwendern zu Verfügung stehen, erweist es sich oft als sinnvoll, ein Deckblatt voranzustellen. Für den Spool-Auftrag wird immer dann ein Deckblatt erstellt, wenn das Flag PRSAP markiert ist. In das Deckblatt gehen neben einigen automatisch vom System erstellten Angaben die Inhalte der Felder PRTXT, PRREC und PRABT ein. Die beiden letztgenannten Felder geben den Namen des Empfängers und dessen Abteilung an.

Relativ selten benutzt, aber nicht unwichtig ist das Feld ARMOD. Ausgabeaufträge können nicht nur an einen Drucker geschickt, sondern auch in einem optischen Archiv abgelegt werden. Derartige Archive, die zusätzlich zum R/3-System beschafft werden müssen, speichern die ausgegebenen Dokumente und gestatten später die Darstellung des Dokuments in der Form, in der es zu Papier gebracht

wurde. Viele R/3-Transaktionen verknüpfen einen Datensatz im R/3 mit den zugehörigen Dokumenten und gestatten den transparenten Zugriff auf diese. Die Ablage des Dokuments im optischen Archiv wird durch das Flag TDARMOD geregelt. Der Wert 1 initiiert das Drucken. Bei einem Wert 2 wird das Dokument nur im optischen Archiv abgelegt. Die Ausgabe auf beide Geräte erfolgt, wenn im Feld ARMOD der Wert 3 eingetragen wird.

Der Baustein GET_PRINT_PARAMERTERS führt einige Plausibilitätsprüfungen durch. Änderungen am Inhalt von PRI_PARAMS sollten daher nur unter Verwendung des genannten Funktionsbausteins erfolgen. Um nachträgliche Veränderungen außerhalb des Bausteins erkennen zu können, ermittelt er eine Prüfsumme, die im Feld PRCHK aufbewahrt wird.

Grundlage für die Übermittlung der Druckparameter an den Funktionsbaustein OPEN_FORM ist die Dictionary-Struktur ITCPO. In dieser Struktur (siehe Tabelle 3.52) finden Sie einige Felder, die in der Struktur PRI_PARAMS keine Entsprechung haben. Das Feld TDPAGESLCT ermöglicht es Ihnen, die zu druckende Seiten auszuwählen. Dabei werden die Seiten durch die Seitennummer identifiziert. Mehrere aufeinander folgende Werte trennen Sie durch das Komma-Zeichen (Beispiel: 1,5,7). Mit dem Bindestrich kennzeichnen Sie Bereiche (Beispiel: 5-7). Fehlt bei einer Bereichsangabe der erste bzw. der letzte Wert, so wird vom Anfang bzw. bis zum Ende gedruckt (Beispiel: -5,7-).

Zusätzlich zum Titel des Spool-Auftrages, der in TDCOVTITLE steht, existieren drei weitere Felder – TDDATASET, TDSUFFIX1 und TDSUFFIX2.

Feldname	Kurzbeschreibung
TDPAGESLCT	Auswahl der zu druckenden Seiten
TDCOPIES	Anzahl der Ausdrucke
TDDEST	Name des Ausgabegeräts
TDPRINTER	Name des Gerätetyps
TDPREVIEW	‚X': Anzeige der Druckansicht am Bildschirm
TDNOPREV	‚X': Keine Druckvorschau
TDNOPRINT	‚X': Keine Druckausgabe aus der Druckvorschau möglich
TDNEWID	‚X': Neuen Spool-Auftrag erzeugen
TDDATASET	Name des Spool-Auftrags
TDSUFFIX1	Suffix 1 zum Namen des Spool-Auftrags

Tabelle 3.52
Felder der Struktur ITCPO

Feldname	Kurzbeschreibung
TDSUFFIX2	Suffix 2 zum Namen des Spool-Auftrags
TDIMMED	‚X': Spool-Auftrag sofort ausgeben
TDDELETE	‚X': Spool-Auftrag sofort nach Ausgabe löschen
TDLIFETIME	Spool-Verweildauer
TDSCHEDULE	Sendezeit Anforderung
TDSENDDATE	Gewünschtes Sendedatum
TDSENDTIME	Gewünschte Sendezeit
TDTELELAND	Länderkennzeichen
TDTELENUM	Telekommunikationspartner
TDTITLE	Text-Beschreibung
TDTEST	Test-Formular
TDPROGRAM	Programmname
TDSCRNPOS	Bildschirm-Anzeigeposition für OTF
TDCOVER	‚X': Deckblatt erstellen
TDCOVTITLE	Deckblatt: Titel
TDRECEIVER	Deckblatt: Empfänger
TDDIVISION	Deckblatt: Abteilung
TDAUTORITY	Berechtigung zum Zugriff auf Spool-Auftrag
TDARMOD	Archivierungsmodus
TDIEXIT	Sofortiger Exit nach Drucken/Faxen aus Druckansicht
TDGETOTF	Nur OTF-Tabelle erstellen
TDFAXUSER	SAP-Office-Benutzer
TDRDIDEV	Ausgabegerät

Tabelle 3.52
Felder der Struktur ITCPO (Fortsetzung)

Der Funktionsbaustein GET_PRINT_PARAMETERS

Dieser Baustein dient zum Bearbeiten und Lesen der aktuellen Druckparameter. Er liefert das Ergebnis in einem Parameter OUT_PARAMETERS mit der Struktur PRI_PARAMS zurück. Es stehen zwei Möglichkeiten zur Verfügung, diesen Parameter mit Daten zu füllen. Zum einen kann der Baustein den Standard-Druckdialog des SAP-Systems aufrufen, um so die interaktive Eingabe der Druckparameter zu ermöglichen. Des Weiteren können Sie in einer Reihe von Import-Parametern einzelne Vorgabewerte und die aktuelle Version des Druckparameter-Datensatzes an den Baustein übergeben, aus denen dieser einen neuen Druckparameter erzeugt. Bei Bedarf können Sie beide Varianten kombinieren. Der Anwender findet dann im Dialog für die Druckparameter die von Ihnen übergebenen Vorgabewerte vor. Bei der Ermittlung der Druckparameter werden eventuelle Voreinstellungen im Benutzerstammsatz berücksichtigt.

Der Funktionsbaustein speichert in einem Feld der Druckparameter-Datensatzes eine Prüfsumme, um die Gültigkeit der Werte überprüfen zu können. Änderungen der Druckparameter sollten daher nie durch direkten Zugriff auf den Datensatz, sondern immer nur mit Hilfe des Funktionsbausteins erfolgen.

Die konkrete Funktion des Bausteins steuern Sie durch zwei Parameter. Mit dem Wert für den Parameter NO_DIALOG legen Sie fest, ob der Baustein im Dialogmodus arbeiten soll oder nicht. Mit dem Parameter MODE können Sie in gewissen Grenzen Einfluss auf den Aufbau des Dialogs nehmen, der vom Funktionsbaustein aufgerufen wird. Die Beschreibung folgt.

Bei der Verwendung der Export-Parameter ist eine Besonderheit zu beachten. Falls Sie den Baustein im Dialogmodus betreiben, liefert er im Parameter VALID eine Information darüber zurück, ob der Dialog ordnungsgemäß beendet oder durch den Anwender abgebrochen wurde. Für die korrekte Funktion Ihrer Anwendung ist es unbedingt erforderlich, diesen Parameter entgegenzunehmen und auszuwerten. Falls Sie den Parameter nicht verwenden, den Funktionsbaustein aber in einem der Modi PARAMS, PARAMSEL, BATCH oder CURRENT aufrufen, reagiert er mit dem Auslösen eines Laufzeitfehlers. Gleiches gilt für die Auswertung des Parameters OUT_PARAMETERS für alle Modi ungleich DISPLAY. In älteren Versionen der SAP-Software wird die Auswertung der Parameter ebenfalls geprüft. Allerdings erzeugt der Baustein im Fehlerfall nur eine Ausnahme und keinen Laufzeitfehler.

Neben den bei der Beschreibung der Struktur PRI_PARAMS erwähnten Feldern verarbeitet der Baustein auch eine Reihe von Parametern, die für die Bedienung eines optischen Archivs notwendig sind und hier nicht beschrieben werden sollen.

Modus PARAMS

Dieser Modus dient zum Aufruf des Standard-Druckdialogs. Es handelt sich um den Standardmodus, der auch dann aufgerufen wird, wenn der Parameter MODE nicht belegt ist. Bei Bedarf werden fehlende Werte aus dem Benutzerstammsatz entnommen oder mit Standardwerten belegt.

Modus PARAMSEL

Dieser Modus entspricht im Wesentlichen dem Mode PARAMS. Allerdings wird im Dialog ein weiteres Feld angezeigt, das den Ausdruck der Selektionen des Reports ermöglich. Das entsprechende Eingabefeld korrespondiert mit dem Feld PRBIG der Struktur PRI_PARAMS.

Modus BATCH

Sehr oft wird der Funktionsbaustein GET_PRINT_PARAMETERS eingesetzt, um die Druckparameter für ein im Hintergrund zu startendes Programm zu ermitteln. In diesem Fall sollte der Import-Parameter REPORT mit dem Namen des zu startenden Programms gefüllt werden. Aus diesem Programm werden eventuell vorhandene Angaben für Seitenlänge und –breite in den Dialog für die Druckparameter übernommen. Außerdem wird die Drucktaste DRUCKEN durch die Drucktaste SICHERN ersetzt. Das Eingabefeld zur Aktivierung des Selektionswerte-Drucks wird ebenfalls aktiviert.

Modus CURRENT

Dieser Modus sollte nicht im Dialogmodus ausgeführt werden. Er ermittelt die gerade aktuellen Druckparameter. Dadurch können Sie innerhalb einer Anwendung die beim Start angegebenen Druckparameter ermitteln.

Modus DISPLAY

Dieser Modus ist nur im Dialog sinnvoll. Er stellt die aktuellen Druckparameter in einem nicht eingabebereiten Popup dar.

Aufgaben

❶ Analysieren Sie die Struktur ITCPO. Fügen Sie eine passende Feldleiste in das Programm YZ438040 ein und füllen Sie die Struktur mit allen Vorgabedaten, die für den Druck erforderlich sind und übergeben diese an OPEN_FORM. Schalten Sie den Dialogmodus für OPEN_FORM aus.

❷ Setzen Sie in ITCPO den Probedruck-Modus und führen Sie das Programm erneut aus.

❸ Bauen Sie den Funktionsbaustein GET_PRINT_PARAMETERS in das Programm ein und versorgen Sie den Funktionsbaustein OPEN_FORM mit den Daten, die GET_PRINT_PARAMETERS liefert.

ABAP Objects

4

Mit der Version 4.0 wurden in ABAP erste objektorientierte Erweiterungen wirksam. Seit der Version 4.6 ist ABAP eine vollwertige objektorientierte Sprache. In diesem Zusammenhang führte die SAP AG die neue Bezeichnung *ABAP Objects* für die R/3-eigene Programmiersprache ein.

Im Gegensatz zu anderen Abschnitten dieses Buches, in denen zunächst Beispiele den praktischen Einstieg ermöglichten und später auf Details eingegangen wurde, soll hier auf entgegengesetzte Weise verfahren werden. An die Erläuterung der Begriffe schließen sich einige einfache Beispiele an. Danach werden die Werkzeuge zur Erstellung und Verwaltung von systemweit verfügbaren Klassen vorgestellt. Den Abschluss des Kapitels bilden einige Beispiele zum Einsatz existierender Klassenbibliotheken, vor allem im Zusammenhang mit dynamischen Dokumenten.

4.1 Begriffe aus der ABAP-Objects-Welt

Der Grundgedanke der objektorientierten Programmierung besteht darin, die Daten und den Programmcode zu deren Bearbeitung in kompakten Elementen, den Objekten, zusammenzufassen. Diese Vorgehensweise erleichtert die Modellierung realer Sachverhalte in einem Programm.

Die objektorientierte Programmierung führt eine ganze Reihe neuer Begriffe ein, die unabhängig von der jeweiligen Programmiersprache existieren. Dieser Abschnitt beschreibt die wichtigsten neuen Begriffe, ohne dass er eine komplette Einführung in das Gebiet der objektorientierten Programmierung sein soll. Dabei sollen Analogien zum herkömmlichen ABAP den Einstieg erleichtern.

4.1.1 Klasse

Eine Klasse ist die Beschreibung eines Objekts, vereinfacht ausgedrückt der Datentyp für ein Objekt. In der Klasse definieren Sie die Daten sowie den Programmcode, der auf diese Daten zugreifen darf. Ein wesentlicher Unterschied zu herkömmlichen Datentypen besteht darin, dass in der Klassendefinition nicht nur einfache Datenfelder definiert werden, sondern auch Quelltext steht. Von der Klasse leiten Sie später die Objekte ab.

4.1.2 Objekt

Ein Objekt ist die Instanz einer Klasse, ebenso wie ein Datenfeld die Instanz eines Datentyps ist. In Ihrem Programm arbeiten Sie mit Objekten. Sie erzeugen diese, weisen Ihnen Daten zu und rufen Funktionen auf, mit denen Sie die Daten der Objekte bearbeiten. Innerhalb eines Programms kann es beliebig viele Objekte einer Klasse geben.

4.1.3 Referenz

Eine Referenz ist ein Verweis auf ein Objekt, entfernt vergleichbar mit einem Feldsymbol. Der Zugriff auf ein Objekt ist nur über eine derartige Referenz möglich. Im Programm erscheint die Referenz als Datenfeld eines speziellen Typs. Es ist möglich, dass mehrere Referenzen auf ein und dasselbe Objekt zeigen.

Neben der Referenz auf echte Objekte existieren weitere Unterarten, die auf Klassen oder Interfaces verweisen können.

4.1.4 Attribut

Attribute sind die Datenfelder eines Objekts. Sie werden innerhalb der Klassendefinition mit denselben Anweisungen deklariert wie in einem herkömmlichen Programm. Da Objekte unabhängig voneinander existieren, sind auch die Attribute unabhängig von denen anderer Objekte, selbst wenn beide Objekte zur selben Klasse gehören.

Die Attribute sind, vom Spezialfall Klassenattribut abgesehen, erst dann verfügbar, wenn das Objekt erzeugt wurde, also eine Instanz einer Klasse existiert.

4.1.5 Methode

Als Methode bezeichnet man die Quelltextteile, mit denen Sie auf die Attribute eines Objekts zugreifen. Verglichen mit der herkömmlichen Programmierweise ähneln die Methoden den Unterprogrammen bzw. Funktionsbausteinen. Auf die Methoden greifen Sie, wieder von der Ausnahme Klassenmethode abgese-

hen, immer über ein konkretes Objekt zu. Daher sind die Methoden erst verfügbar, wenn Objekte, also Instanzen einer Klasse, existieren.

4.1.6 Event

Objekte können ein Ereignis (*Event*) auslösen. Andere Objekte können dieses Ereignis erkennen und darauf reagieren. Durch den Event-Mechanismus können Objekte miteinander kommunizieren. Die Events sind an konkrete Objekte gebunden, beziehen sich also immer auf explizit benannte Objekte und nicht pauschal auf alle existierenden Instanzen einer Klasse.

Um auf einen Event reagieren zu können, muss sich ein Objekt für die Behandlung dieses Ereignisses registrieren. Dabei muss es nicht nur den Namen des Ereignisses, sondern auch das auslösende Objekt angeben.

4.1.7 Klassenattribut

Die Klassenattribute gelten nicht für die einzelnen Objekte, sondern gemeinsam für alle Objekte einer Klasse. Zur Laufzeit einer Anwendung existiert je Klasse nur eine Instanz der Klassenattribute. Der Speicherplatz für die Klassenattribute steht zur Verfügung, bevor Instanzen von dieser Klasse abgeleitet wurden. Klassenattribute können Sie unter anderem für Informationen benutzen, die für alle Objekte dieser Klasse gelten sollen.

4.1.8 Klassenmethode

Eine Klassenmethode unterscheidet sich von einer herkömmlichen Methode durch den Gültigkeitsbereich. Herkömmliche Methoden gehören zu einem Objekt. Die Klassenmethoden hingegen gehören zur Klasse und existieren demzufolge, ähnlich wie die Klassenattribute, bereits vor Erzeugung von Objekten. Sie können Klassenmethoden benutzen, um auf Klassenattribute zuzugreifen. In der Praxis werden Klassenattribute benutzt, um Programmfunktionalität objektorientiert aufzubereiten, ohne bei der späteren Verwendung immer erst ein konkretes Objekt instanziieren zu müssen.

4.1.9 Klassenevent

Die Klassenevents existieren ebenso wie Klassenattribute und Klassenmethoden für die gesamte Klasse. Falls sich Objekte zur Behandlung von Ereignissen registrieren, müssen Sie für Klassenereignisse das auslösende Objekt nicht angeben. Die Behandlung des Ereignisses erfolgt dann unabhängig davon, welches konkrete Objekt das Ereignis auslöst.

4.1.10 Ausnahmen

Ausnahmen (*Exceptions*) sind kein Begriff aus der objektorientierten Programmierung. Allerdings können Methoden ebenso wie Funktionsbausteine Ausnahmen auslösen, um dem rufenden Programm Fehlerzustände zu signalisieren. Während die Events zur normalen Kommunikation zwischen Objekten vorgesehen sind, stellen Ausnahmen die Rückmeldung von Fehlerzuständen an den herkömmlich programmierten Programmteil sicher.

Innerhalb von ABAP Objects existieren inzwischen zwei unterschiedliche Arten von Ausnahmen, die im umgebenden Programm auch unterschiedlich behandelt werden müssen. Zunächst steht die schon in frühen ABAP-Versionen verfügbare einfache Exception bereit, die innerhalb der Funktionsbaustein-Programmierung benutzt wird. Das Auslösen einer solchen Exception hat vor allem Einfluss auf das Systemfeld `SY-SUBRC`. Neu sind die klassenbasierten Ausnahmen, deren Konzept teilweise an die Programmiersprache Java erinnert. Klassenbasierte Ausnahmen beruhen auf vordefinierten Ausnahmeklassen, die direkt oder durch Vererbung und Erweiterung benutzt werden. Eine derartige Ausnahme kann Informationen aufnehmen und an das umgebende Programm übermitteln. Außerdem können diese Ausnahmen individuell behandelt werden. Das Konzept dazu unterscheidet sich aber erheblich von der Behandlung einfacher Exceptions. Perspektivisch sollen die klassenbasierten Ausnahmen die einfachen Exceptions ablösen.

4.1.11 Interface

Ein Interface stellt die Definition einer Schnittstelle dar. Im Interface definieren Sie sowohl Attribute als auch Methoden. Allerdings implementieren Sie die Methoden nicht, sondern legen lediglich deren Aufrufparameter fest. Die Implementierung erfolgt in den Klassen, die das Interface in ihre Deklaration einbinden. Durch die Verwendung eines Interface stellen Sie sicher, dass Sie unterschiedliche Klassen über eine identische Schnittstelle ansprechen können. Die Interfaces bieten somit neben dem Vererbungsmechanismus eine zweite Möglichkeit, Schnittstellen für mehrere Klassen zu definieren.

4.1.12 Konstruktor

In einer Klasse kann, muss aber nicht, ein so genannter Konstruktor enthalten sein. Dabei handelt es sich um eine Methode, die beim Erzeugen einer Instanz einer Klasse automatisch aufgerufen wird. Konstruktoren dienen meist dazu, das angelegte Objekt zu initialisieren. Sie können zu diesem Zweck mit Parametern versorgt werden. Genau genommen existieren zwei Konstruktoren. Der Klassenkonstruktor (vordefinierter Name `CLASS_CONSTRUCTOR`) wird einmalig beim ersten Zugriff auf eine Klasse aufgerufen. Der Instanz-Konstruktor (vorde-

finierter Name CONSTRUCTOR) hingegen wird für jedes neu erstellte Objekt ausgeführt.

4.1.13 Sichtbarkeit

Sowohl Attribute als auch Methoden können mit Parametern versehen werden, die ihre Sichtbarkeit einschränken. Es existieren drei Stufen der Sichtbarkeit. Öffentliche Attribute und Methoden sind uneingeschränkt vom umgebenden Programm aus sichtbar. Auf die öffentlichen Attribute können Sie lesend und schreibend zugreifen. Geschütze Attribute und Methoden sind nur innerhalb der Klasse und möglicher Erben bekannt, es kann also kein externer Zugriff erfolgen. Noch stärker sind die Einschränkungen bei privaten Elementen. Diese sind nur innerhalb der Klasse sichtbar.

4.1.14 Vererbung

Als Vererbung bezeichnet man das Ableiten neuer Klassen aus bereits vorhandenen. Dabei erbt eine neue Klasse alle Eigenschaften (Attribute und Methoden) der Klasse, die sie beerbt. Alle diese Eigenschaften sind in der neuen Klasse verfügbar, als wären sie dort definiert. Die als Ausgangspunkt für die Vererbung dienende Klasse wird auch als Oberklasse bezeichnet.

Eine beerbende Klasse kann weitere Attribute oder Methoden hinzufügen oder aber Methoden überschreiben (mit neuer Funktionalität versehen).

Die untergeordneten Klassen sind zuweisungskompatibel zu Referenzen mit dem Datentyp der Oberklasse.

4.1.15 Kapselung

Durch Einschränkung der Sichtbarkeit von Attributen und Methoden können Sie die innere Funktionalität eines Objekts vor der Außenwelt verbergen. Die Verwender eines Objekts greifen nur über die öffentliche Schnittstelle auf die Ressourcen desselben zu, ohne dessen innere Funktionsweise zu kennen.

4.1.16 Polymorphie

Unterschiedliche Klassen können gleichnamige Methoden implementieren. Dabei verhalten sich diese Methoden möglicherweise völlig unterschiedlich.

4.1.17 Klassen-Pool

Ein Klassen-Pool ist eine spezielle Programmart innerhalb des R/3-Systems. In einem Klassen-Pool definieren Sie Klassen, die systemweit genutzt werden können. Die Klassen in einem Klassen-Pool werden mit dem *Class Builder* bearbeitet.

4.2 Kommandos

Dieser Abschnitt beschreibt die wichtigsten neuen Sprachelemente, die im Zusammenhang mit der objektorientierten Weiterentwicklung von ABAP geschaffen wurden. Sie können diese Kommandos nutzen, um in eigenen Programmen objektorientierte Elemente zu erzeugen und zu verwenden.

4.2.1 Definieren einer Klasse

Eine Klassendefinition besteht immer aus zwei Blöcken, von denen jeder für sich mit dem Kommando `CLASS` eingeleitet und durch `ENDCLASS` abgeschlossen wird. Der erste Block stellt die eigentliche Definition dar, der zweite dient zur Implementierung der Methoden. Zur Unterscheidung der beiden Blöcke dienen weitere Schlüsselwörter innerhalb der `CLASS`-Anweisung.

Eine neue Klasse müssen Sie zunächst definieren. Dabei legen Sie die Attribute und die Schnittstellen der Methoden fest. Klassen dürfen nicht in Unterprogrammen, Funktionsbausteinen oder Modulen definiert werden. Eine Schachtelung von Klassendefinitonen ist ebenfalls nicht möglich. Die Syntax des Kommandos zur Definition einer Klasse lautet:

```
CLASS class DEFINITION
  [ INHERITING FROM superclass |
    ABSTRACT |
    FINAL    |
    PUBLIC   |
    CREATE ( PUBLIC | PROTECTED | PRIVATE )
  ]
  [ PUBLIC SECTION. ]
    Component definitions...
  [ PROTECTED SECTION. ]
    Component definitions...
  [ PRIVATE SECTION. ]
    Component definitions...
ENDCLASS.
```

Eingeleitet wird die Definition durch das Schlüsselwort `CLASS`, gefolgt vom Namen der anzulegenden Klasse und dem Schlüsselwort `DEFINITION`. Mit dieser Form erzeugen Sie eine Klasse, die nur innerhalb des aktuellen Programms be-

kannt ist. Obwohl die Klasse bezüglich des Programms zu den globalen Definitionen gehört, spricht man von einer lokalen Klasse, da sie außerhalb des Programms nicht sichtbar ist.

Die Eigenschaften einer Klasse können durch zusätzliche Schlüsselwörter modifiziert werden. Eines dieser Schlüsselwörter ermöglicht die Nutzung des Vererbungsmechanismus. Durch `INHERITING FROM` legen Sie fest, dass die neue Klasse alle Elemente der angegebenen Oberklasse erbt, die in dieser nicht privat sind. Sofern dies nicht ausdrücklich verboten wurde, können Sie in der neuen Klasse alle geerbten Methoden neu implementieren. Jede Klasse kann nur von einer einzigen Klasse erben. Auf diese Weise steht ein Klassenbaum, an dessen Spitze die implizit vorhandene Klasse `OBJECT` steht.

Der Zusatz `ABSTRACT` kennzeichnet eine Klasse als abstrakt. Dies bedeutet, dass von dieser Klasse keine Instanzen (Objekte) erzeugt werden können. Derartige Klassen werden üblicherweise als Ausgangspunkt für einen Zweig im Klassenbaum benutzt. Falls mehrere ähnliche Objekte einige Attribute und Methoden gemeinsam haben, werden diese oft in einer abstrakten Klasse implementiert. Dies erspart die mehrfache Implementierung identischer Funktionen.

Die gegensätzliche Wirkung hat der Zusatz `FINAL`. Dieser Zusatz legt fest, dass von dieser Klasse keine weiteren Unterklassen abgeleitet werden dürfen.

Der `CREATE`-Zusatz bestimmt, wer Instanzen der Klasse anlegen darf. Die Einstellung `CREATE PUBLIC` ist Standard und muss nicht notiert werden. Sie ermöglicht, dass an jeder Stelle des Programmcodes eine Instanz erzeugt werden darf. Falls `CREATE PROTECTED` benutzt wird, darf eine Instanz der Klasse nur innerhalb der Klasse selbst oder innerhalb einer Unterklasse erzeugt werden. Der Zusatz `CREATE PRIVATE` schließlich erlaubt die Erzeugung von Instanzen nur innerhalb der Klasse selbst. Dies erfordert die Bereitstellung einer Klassenmethode, in der Instanzen der Klasse erzeugt werden. Diese Methode muss eine Referenz auf die erzeugte Instanz zurückliefern. Sie entspricht in ihrer Wirkung in etwa einem aus anderen objektorientierten Programmiersprachen bekannten Konstruktor.

Durch den Zusatz von `PUBLIC` entsteht eine im SAP-System global bekannte Klasse. Derartige Klassen können allerdings nur im Class Builder angelegt werden, der den `PUBLIC`-Zusatz automatisch anfügt. Bei der manuellen Definition einer Klasse kann dieser Zusatz nicht verwendet werden.

Nach den Angaben zur Klasse selbst werden die Komponenten der Klasse definiert. Dazu stehen drei verschiedene Sektionen (`PUBLIC SECTION`, `PROTECTED SECTION` und `PRIVATE SECTION`) bereit. In einer Klasse müssen nicht alle Sektionen vorhanden sein, die genannte Reihenfolge ist allerdings verbindlich. Die Sektionen legen den Gültigkeitsbereich der in ihnen definierten Attribute und Methoden fest. Auf öffentliche Elemente (`PUBLIC SECTION`) kann von überallher zugegriffen werden. Die geschützten Elemente (`PROTECTED SECTION`) sind nur innerhalb der Klasse selbst und von untergeordneten Klassen aus verfüg-

bar. Private Elemente (PRIVATE SECTION) können nur innerhalb der Klasse selbst benutzt werden.

In einer Klasse existieren drei Gruppen von Komponenten. Nicht jede Komponente kann in jeder der genannten Sektionen angelegt werden. Zur ersten Gruppe gehören die Schnittstellen-Komponenten. Sie werden mit den Anweisungen INTERFACES und ALIASES erzeugt.

Die zweite Gruppe bilden die Klassen-Komponenten, auch als statische Komponenten bezeichnet. Zu dieser Gruppe gehören Typ- und Konstantendefinitionen (TYPES und CONSTATNTS), Klassendaten (CLASS-DATA), Klassenmethoden (CLASS-METHODS) und Klassenereignisse (CLASS-EVENTS).

Instanzenbezogene Daten, Methoden und Ereignisse bilden die dritte Gruppe. Sie werden mit den Anweisungen DATA, METHOD und EVENT definiert. Die Syntax der Komponentendefinitionen lautet somit:

```
[ INTERFACES interfaces ]
[ ALIASES name FOR interface~component ]
[ TYPES type declarations ]
[ CONSTANTS constant declarations ]
[ CLASS-DATA field definitions ]
[ CLASS-METHODS method declarations ]
[ CLASS-EVENTS event declarations ]
[ DATA field definitions ]
[ METHODS method declarations ]
[ EVENTS event declarations ]
```

Die Syntax der einzelnen Anweisungen wird im Anschluss an diesen Abschnitt detaillierter beschrieben. Hier soll die Definition der Klasse als Ganzes im Mittelpunkt stehen.

Die Syntax des Implementierungsteils ist wesentlich einfacher, da hier nur Methoden enthalten sein können:

```
CLASS class IMPLEMENTATION.
{
  METHOD header.
    method implementaion
  ENDMETHOD.
} . . .
ENDCLASS.
```

Innerhalb des Implementierungsteils müssen alle Methoden, die im Definitionsteil deklariert oder über Interfaces importiert wurden, implementiert werden.

Die CLASS-Anweisung existiert noch in zwei anderen Formen. Sie besteht dann nur aus einer einzelnen Zeile:

```
CLASS class DEFINITION DEFERRED.
```

bzw.

```
CLASS class DEFINITION LOAD.
```

Der DEFERRED-Zusatz macht nur den Namen der Klasse bekannt, wobei die Definition später folgt. Sie können den Klassennamen jetzt aber schon vor der eigentlichen Definition der Klasse in anderen Definitionen benutzten.

Der LOAD-Zusatz ist nur im Zusammenhang mit einer globalen (PUBLIC-)Klasse erforderlich. Er sorgt dafür, dass die Definition der Klasse geladen wird.

4.2.2 Definieren eines Interface

Die Interface-Definition ähnelt der Definition einer Klasse. Die wesentlichen Unterschiede bestehen im Fehlen von Sichtbarkeitsgruppen. Alle Elemente sind automatisch öffentlich (public). Es können dieselben Komponenten wie in Klassen definiert werden. Somit ist es auch möglich, Interfaces ineinander zu schachteln.

```
INTERFACE interface [ DEFERRED ].
  [ INTERFACES interfaces ]
  [ ALIASES name FOR interface~component ]
  [ TYPES type declarations ]
  [ CONSTANTS constant declarations ]
  [ CLASS-DATA field definitions ]
  [ CLASS-METHODS method declarations ]
  [ CLASS-EVENTS event declarations ]
  [ DATA field definitions ]
  [ METHODS method declarations ]
  [ EVENTS event declarations ]
ENDINTERFACE.
```

Der Zugriff auf die Elemente eines Interface innerhalb der Klasse, die dieses Interface benutzt, erfolgt in der Form

```
interface~component
```

4.2.3 Definieren von Komponenten

Innerhalb von Klassen und Interfaces können diverse Komponenten definiert werden. Einige der dazu verwendeten Anweisungen (z.B. DATA oder TYPES) sind bereits bekannt, andere hingegen sind neu. Dieser Abschnitt beschreibt die diversen Komponenten und die dazu notwendigen Anweisungen detailliert, wobei der Schwerpunkt auf den neu eingeführten Anweisungen liegt.

Ereignisse deklarieren

Events können Sie innerhalb von Klassen- und Interfacedefinitionen deklarieren. Sie benutzen dazu das Schlüsselwort EVENT, dessen komplette Syntax Sie nachfolgend finden.

```
EVENTS event EXPORTING {
  VALUE(parameter) [
    TYPE type | LIKE field [OPTIONAL | DEFAULT default]]}.
```

Ein derartiges Ereignis kann neben der Tatsache seiner eigenen Existenz an den Empfänger weitere Informationen übermitteln. Diese sind ähnlich wie Export-Parameter einer Methode bei der Definition des Ereignisses anzugeben und zu typisieren.

Ein klassenbezogenes Ereignis kann keine zusätzlichen Parameter übermitteln. Die Syntax des Kommandos zur Definition derartiger Ereignisse vereinfacht sich daher zu

```
CLASS-EVENTS event.
```

Methodendefinition

Die Methoden ähneln bezüglich ihrer Funktionsweise und ihrer Eigenschaften den Funktionsbausteinen. Dementsprechend ähneln sich auch die Anweisungen zur Definition der Schnittstelle. Im Unterschied zu Funktionsbausteinen pflegen Sie die Schnittstellendefinition aber nicht mit einem speziellen Werkzeug in mehreren aufeinander folgenden Dynpros, sondern legen sie direkt im Quelltext fest, was an die herkömmlichen Unterprogramme erinnert.

Methoden und deren Schnittstellen werden im Definitionsteil einer Klasse definiert.

```
METHODS method
  [ IMPORTING ( [[VALUE | REFERENCE](parameter)] | parameter
      [TYPE type | LIKE field]
      [OPTIONAL | DEFAULT default] ) . . . ]
  [ EXPORTING ( [[VALUE | REFERENCE](parameter)] | parameter
      [TYPE type | LIKE field]
      [OPTIONAL | DEFAULT default] ) . . . ]
  [ CHANGING  ( [[VALUE | REFERENCE](parameter)] | parameter
      [TYPE type | LIKE field]
      [OPTIONAL | DEFAULT default] ) . . . ]
  [ RETURNING VALUE(parameter) [ TYPE type | LIKE field ] ]
  [ RAISING { class_exception } ]
  [ EXCEPTIONS { exception } ]
  [ ABSTRACT ]
  [ FINAL ]
  [ REDEFINITION ].
```

Die Definition von Import- und Export-Parametern durch die Schlüsselwörter IMPORT, EXPORT und CHANGING entspricht der Verfahrensweise bei Funktionsbausteinen. Die Typisierung der Parameter ist ähnlich wie bei Unterprogrammen durch die Zusätze TYPE oder LIKE möglich. Allerdings ist die Typisierung obligatorisch. Schließlich können Sie Parameter mit OPTIONAL als optional definieren, wobei der Parameter mit dem typgerechten Initialwert gefüllt wird, falls er nicht übergeben wird. Mit dem Zusatz DEFAULT definieren Sie ebenfalls einen optionalen Parameter, wobei aber ein Vorgabewert gesetzt wird.

Bei der Typisierung mit TYPE stehen nicht nur die Datentypen aus Programm und Dictionary zur Verfügung. Tabelle 4.1 zeigt alle Varianten zur Typisierung.

Anweisung	Beschreibung
`Type`	Parameter entspricht dem Datentyp.
`ANY`	Parameter ist zu allen Datentypen kompatibel.
`REF TO class`	Parameter ist Referenz auf Klasse oder Interface.
`LINE OF itab`	Parameter entspricht der Zeilenstruktur einer internen Tabelle.
`TYPE [ ANY | INDEX | STANDARD | SORTED | HASHED ] TABLE`	Parameter ist eine interne Tabelle mit der angegebenen Eigenschaft.

Tabelle 4.1
Typisierungsvarianten

Gegenüber Funktionsbausteinen existiert eine prinzipielle Erweiterung. Nach dem Zusatz RETURNING können Sie einen Parameter angeben, der als alleiniger Rückgabewert der Methode dient. Die Verwendung dieses Zusatzes schließt den Einsatz der Parametertypen EXPORTING und CHANGING aus. Der RETURNING-Zusatz soll später dazu dienen, den Rückgabewert der Methode direkt in Ausdrücken verwenden zu können. Im Moment ist diese Variante der Wertübergabe allerdings nur in speziellen Fällen möglich. Beachten Sie dazu bitte den Abschnitt zur Methodenimplementierung und die Beispiele.

Falls die Parameternamen einer Methode identisch sind mit dem Namen von Attributen, so verdecken die Parameter innerhalb der Methode die Attribute. Der Zugriff auf die Attribute ist aber weiterhin durch den Vorsatz ME-> möglich. Beachten Sie dazu das Programm YZ342040.

Mit dem EXCEPTIONS-Zusatz definieren Sie die Ausnahmen, welche die Methode auslösen kann. Dies entspricht wieder völlig dem Verfahren bei Funktionsbausteinen. Die klassenbasierten Ausnahmen hingegen werden mit der Anweisung RAISING deklariert.

Im Zusammenhang mit der Vererbung von Klassen stehen die drei Anweisungen ABSTRACT, FINAL und REDEFINITION. Falls Sie eine Methode einer Oberklasse in einer abgeleiteten Klasse neu implementieren (überschreiben) möchten, müssen Sie im Definitionsteil der abgeleiteten Klasse die Methode durch das Schlüsselwort REDEFINITION kennzeichnen. Für Methoden, die überschrieben werden, dürfen außer dem Zusatz FINAL keine weiteren Optionen vergeben werden. Das bedeutet, dass beim Überschreiben von Methoden die Schnittstelle nicht verändert, also auch nicht um zusätzliche Parameter erweitert werden darf. Die beiden einzig möglichen Anweisungen im Zusammenhang mit REDEFINITION sind somit:

```
METHOD method REDEFINITION.
METHOD method FINAL REDEFINITION.
```

Eine als ABSTRACT gekennzeichnete Methode wird in der aktuellen Klasse nicht implementiert. Die Implementierung erfolgt in abgeleiteten Klassen. Im Gegensatz dazu kann eine mit dem Zusatz FINAL versehene Methode nicht mehr überschrieben werden.

Im Zusammenhang mit der Ereignisverarbeitung sind spezielle Methoden vorgesehen, die zur Behandlung der Ereignisse dienen. Voraussetzung ist, dass diese Methoden entsprechend definiert sind und später als behandelnde Methode für das Ereignis registriert werden. Eine Methode zur Ereignisbehandlung definieren Sie wie folgt:

```
METHODS method FOR EVENT event OF class
  [ IMPORTING { parameter } ]
  [ ABSTRACT ]
  [ FINAL ]
  [ REDEFINITION ].
```

Die importierten Parameter müssen den Export-Parametern des Ereignisses entsprechen.

Klassenmethoden definieren Sie ähnlich wie einfache Methoden. Allerdings können Klassenmethoden keine Schnittstelle besitzen. Die Definition vereinfacht sich daher zu:

```
CLASS METHODS method [ FOR EVENT event OF class ].
```

Methodenimplementierung

Die im Definitionsteil einer Klasse definierten Methoden müssen später natürlich auch implementiert werden. Im Implementierungsteil einer Klasse leiten Sie die Implementierung der Methode mit der Anweisung

```
METHOD method.
```

ein und schließen Sie mit

```
ENDMETHOD.
```

ab. Die Schnittstellendefinition müssen Sie nicht nochmals wiederholen. Zwischen den beiden Anweisungen notieren Sie den Quelltext der Methode.

Methode aufrufen

Eine Methode wird ähnlich einem Funktionsbaustein mit dem Kommando `CALL METHOD` aufgerufen. Die Parameterübergabe erfolgt wie bei herkömmlichen Funktionsbausteinen. Neu ist das Schlüsselwort `RECEIVING`, mit dem Sie den `RETURNING`-Parameter der Methode entgegennehmen. Eine Erweiterung der Aufrufsyntax besteht in der Verwendung der Zusätze `PARAMETER-TABLE` und `EXCEPTION-TABLE`. Diese Optionen erlauben es, die Parameter bzw. Ausnahmen in einer internen Tabelle zu übergeben.

```
CALL METHOD objectreference->method
  [ EXPORTING   { parameter = value } ]
  [ IMPORTING   { parameter = field } ]
  [ CHANGING    { parameter = field } ]
  [ RECEIVING     parameter = field ]
  [ EXCEPTIONS [{ exception = value } ]
  [ PARAMETER-TABLE itab ]
  [ EXCEPTION-TABLE itab ].
```

Da einfache Methoden immer an existierende Objekte gekoppelt sind, müssen Sie die Methode natürlich unter Verwendung der Objektreferenz aufrufen.

Falls die gesamte Schnittstelle einer Methode nur aus einen einzigen Import-Parameter besteht, kann der Aufruf der Methode vereinfacht werden. Der Import-Parameter wird einfach in runde Klammern eingeschlossen:

```
CALL METHOD objectreference->method( single_parameter ).
```

Eine ähnliche Form des Aufrufs ist möglich, wenn die Schnittstelle der Methode nur aus Import-Parametern besteht. In diesem Fall werden die Parameter beim Aufruf als Name-Wert-Paare übergeben:

```
CALL METHOD objectreference->method(
              param_name_1 = param_value_1 ...
              param_name_n = param_value_n).
```

Für die beiden zuletzt genannten Formen der Parameterübergabe ist in speziellen Fällen eine weiter vereinfachte Form des Aufrufs möglich, wenn die Methode einen Wert zurückliefert, also einen `RETURNING`-Parameter in der Schnittstelle definiert hat. Dabei kann auf das Schlüsselwort `CALL METHOD` verzichtet werden. Der Methodenaufruf kann dann direkt als Operand in Anweisungen stehen. Möglich ist dies z. B. in Zuweisungen (Methodenaufruf als Quelle in einer `MOVE`-Anweisung oder als arithmetischer Operand in herkömmlichen Zuweisungen) und in logischen Ausdrücken. Dabei ist zu beachten, dass die run-

den Klammern zur Syntax gehören, also auch dann notiert werden müssen, wenn die Methode keine Import-Parameter besitzt. In diesem Fall muss ein Leerzeichen zwischen öffnender und schließender Klammer stehen!

Objektreferenz erzeugen

Um auf Objekte zuzugreifen, benötigen Sie eine so genannte Objektreferenz. Unter einer solchen Referenz können Sie sich ein Feldsymbol vorstellen, das auf einen bestimmten Speicherbereich zeigt. An eben diesem Platz im Speicher befindet sich das Objekt. Es ist durchaus möglich, dass mehrere Referenzen auf ein Objekt zeigen. Zuweisungen zu Objektreferenzen ändern damit nicht das Objekt, sondern modifizieren lediglich einen Zeiger, der dann ggf. auf ein anderes Objekt zeigt. Ebenso führt das Löschen einer Objektreferenz nicht automatisch zum Löschen des Objekts. Erst wenn auf ein Objekt keine Objektreferenzen mehr zeigen, wird der vom Objekt belegte Speicherplatz automatisch freigegeben.

Erzeugt wird eine Objektreferenz mit dem Kommando

```
DATA reference [
  TYPE REF TO class | TYPE referencetype | LIKE reference ].
```

Natürlich können Sie in der TYPES-Anweisung auch eigene Datentypen für Objektreferenzen deklarieren:

```
TYPES referencetype [ TYPE REF TO class | LIKE reference ].
```

Beim Erzeugen der Objektreferenz wird noch kein Objekt angelegt. Es steht lediglich ein Feld zur Verfügung, das auf ein Objekt zeigen kann. Beachten Sie bitte, dass in den eben genannten Anweisungen der Platzhalter class sowohl für vollwertige Klassen als auch für Interfaces steht.

Innerhalb jedes Objekts stehen zwei Objektreferenzen implizit zur Verfügung. Es handelt sich dabei um die so genannte Selbstreferenz ME und die Pseudoreferenz SUPER. Die Referenz ME zeigt auf das eigene Objekt. Dadurch kann auf jede Komponente des Objekts durch explizite Angabe von

```
ME->component
```

zugegriffen werden. Das ist mitunter notwendig, da die Komponenten des Objekts durch gleichnamige Parameter etc. verdeckt sein können.

Die Pseudoreferenz SUPER zeigt nicht auf ein reales Objekt, sondern auf die übergeordnete Klasse. Sie wird eingesetzt, um in überschriebenen Methoden auf die entsprechende Methode der Oberklasse zugreifen zu können.

Objekt erzeugen

Objekte müssen Sie explizit erzeugen. Dies erfolgt am einfachsten mit der Anweisung

```
CREATE OBJECT reference.
```

Dabei bestimmt der Typ des Referenzfelds das anzulegende Objekt. Mitunter soll von dieser Vorgabe abgewichen werden. Dazu wird der Typ der anzulegenden Klasse explizit übergeben:

```
CREATE OBJECT reference TYPE class | (class_field).
```

Dabei muss Zuweisungskompatibilität zwischen dem Typ der anzulegenden Klasse und dem des Referenzfeldes bestehen. Der Klassenname des anzulegenden Objekts kann sowohl statisch als Direktwert als auch dynamisch als Inhalt eines Feldes übergeben werden.

Nach diesem Kommando existiert ein Objekt, und die von Ihnen angegebene Objektreferenz zeigt auf dieses Objekt. Auf die Elemente innerhalb eines Objekts können Sie – vorausgesetzt, die Sichtbarkeitsklassen erlauben dies – durch die in Tabelle 4.2 dargestellten Anweisungen zugreifen.

Element	Zugriff
Attribut	`Reference->attribute`
Methode	`CALL reference->method`
Atttribut aus einem Interface	`Reference->interface~attribute`
Methode aus einem Interface	`CALL reference->interface~method`

Tabelle 4.2
Zugriff auf Elemente in Objekten

Beim Erzeugen eines Objekts kann automatisch ein so genannter Konstruktor aufgerufen werden. Dabei handelt es sich um eine Methode mit dem vordefinierten Namen `CONSTRUCTOR`. Dieser Methode können Sie Parameter übergeben. Dazu erhält die `CREATE`-Anweisung den Zusatz

```
EXPORTING param_name_1 = param_value_1 ...
          param_name_n = param_value_n.
```

Ein Konstruktor kann Ausnahmen auslösen, die ebenfalls bei `CREATE` zu definieren sind. Dazu dient der Zusatz

```
EXCEPTIONS exception_name_1 = exception_value_1 ...
           exception_name_n = exception_value_n.
```

Wenn ein Konstruktor nicht-optionale Parameter besitzt, müssen diese Parameter bei CREATE OBJECT versorgt werden.

Ausnahmen und Ereignisse auslösen

Um in der Methode eines Objekts eine Ausnahme auszulösen, benutzen Sie für herkömmliche, also nicht klassenbasierte Ausnahmen ebenso wie in Funktionsbausteinen das Kommando

```
RAISE exception.
```

Für die Erzeugung von klassenbasierte Ausnahmen wird eine von zwei neuen Varianten dieses Kommandos erforderlich. Für eine klassenbasierte Ausnahme ist immer ein reales Objekt, also die Instanz einer Klasse, erforderlich. Diese muss nun nicht mittels des CREATE-Kommandos erzeugt werden. Die entsprechende Klasse muss dabei CX_ROOT oder eine davon abgeleitete sein.

Das Kommando

```
RAISE EXCEPTION TYPE exception_class
  [ EXPORTING { parameter = value } ].
```

erzeugt eine Instanz der angegebenen Klasse, füllt ggf. deren Attribute mit den angegebenen Werten und löst die Ausnahme aus. Falls jedoch bereits ein Ausnahme-Objekt existiert, so kann dieses mit dem Kommando

```
RAISE EXCEPTION exception_object_reference.
```

weitergereicht werden. Zusätze zum Modifizieren von Attributen sind hier nicht mehr möglich. Weiterreichen von Ausnahmen findet oft in den Behandlungsroutinen innerhalb einer Anwendung statt.

Als Grundlage für alle im System verwendeten Ausnahmen stehen die drei Klassen CX_STATIC_CHECK, CX_DYNAMIC_CHECK und CX_NO_CHECK zur Verfügung. Dies sind allerdings abstrakte Klassen, von denen keine konkreten Objekte instanziiert werden können. Für alle auszulösenden Ausnahmen muss also eine eigene Klasse erstellt werden, die von einer der drei erwähnten Klassen erbt. Im System existieren bereits einige Ausnahmeklassen, die aber sehr systemnah sind und daher für eigene Anwendungen eher selten genutzt werden können. Diese Ausnahme-Klassen werden durch weitere abstrakte Klassen gruppiert. Dies ermöglicht es, mehrere Ausnahmen durch Angabe der abstrakten Oberklasse gemeinsam abzufangen.

Die drei Ausnahmeklassen unterscheiden sich hinsichtlich Ihrer Behandlung und Prüfung durch den Compiler. Ausnahmen vom Typ CX_STATIC_CHECK, CX_DYNAMIC_CHECK werden wie beschrieben ausgelöst und verarbeitet. Bei Ausnahmen vom Typ CX_STATIC_CHECK kann der Compiler bereits bei der Übersetzung prüfen, ob alle Ausnahmen korrekt behandelt werden. Bei dem Typ CX_DYNAMIC_CHECK erfolgt keine Prüfung zur Übersetzungszeit. Daher

kann aber auch ein Laufzeitfehler entstehen, wenn eine Ausnahme ausgelöst, aber nicht behandelt wird.

Ausnahmen vom Typ CX_NO_CHECK dienen üblicherweise zur Signalisierung von systemnahen Problemen. Sie werden durch das Laufzeitsystem oder systemnahe Routinen abgefangen. In echten Anwendungen sind sie nicht erforderlich.

Für einfache Events ist ebenfalls das Kommando RAISE mit einem anderen Zusatz zuständig.

```
RAISE EVENT event [ EXPORTING { parameter = value } ].
```

Vor allem können Sie an einfache Events (keine Klassen-Events) zusätzliche Parameter übergeben. Diese Parameter kann die später definierte Methode zur Ereignisbehandlung übernehmen und auswerten.

Es gibt, abgesehen von der optionalen Parameterliste, keinen Unterschied beim Aufruf von einfachen und von Klassenereignissen. Allerdings können Sie in Klassenmethoden nur Klassenereignisse auslösen, während Ihnen in instanzbezogenen Methoden beide Arten von Ereignissen zur Verfügung stehen.

Ereignisbehandlung

Zur Ereignisbehandlung müssen Sie zunächst innerhalb einer Klassendefinition eine Methode als *Event-Handler* definieren. Dies erfolgt dort mit dem Zusatz FOR EVENT zur Methodendefinition. Innerhalb der Anwendung müssen Sie diese Methode dann noch registrieren. Dabei erfolgt gegebenenfalls auch die Zuordnung von konkreten Ereignis auslösenden Objekten zu Ereignis behandelnden Objekten.

Den Handler für Klassenereignisse setzen Sie durch die Anweisung

```
SET HANDLER handlerlist [ ACTIVATION field ].
```

Dabei ist handlerlist eine Liste mit den Ereignisbehandlungs-Methoden. Diese Methoden sind vollständig zu benennen, also in der Form

```
object->method.
```

Die einzelnen Methoden notieren Sie nur durch Leerzeichen voneinander getrennt. Das Kommando SET HANDLER kennt keine Doppelpunkt-Variante.

Die Registrierung von instanzbezogenen Ereignissen ist geringfügig aufwändiger, da sie auch noch das Ereignis auslösende Objekt angeben müssen. Dies erfolgt durch den Zusatz FOR, der von der Objektreferenz gefolgt wird.

```
SET HANDLER handlerlist FOR
  [ objectreferenc | ALL INSTANCES ]
  [ ACTIVATION field ].
```

Falls Sie eine Ereignisbehandlung für alle Instanzen (Objekte) einer Klasse durchführen möchten, können Sie dies durch den Zusatz `ALL INSTANCES` nach dem Schlüsselwort `FOR` erreichen.

In beiden Formen des Kommandos besteht die Möglichkeit, über das Schlüsselwort `ACTIVATION` die Ereignisbehandlung zu aktivieren oder wieder zu deaktivieren. Sie übergeben dazu ein Feld mit der Länge 1 und dem Typ C. Ist dieses Feld leer, so wird die Ereignisbehandlung deaktiviert. Hat das Feld den Wert „X" oder fehlt der Zusatz `ACTIVATION`, erfolgt die Aktivierung der Ereignisbehandlung.

Behandeln von klassenbasierten Ausnahmen

Zur Behandlung klassenbasierter Ausnahmen müssen die Methodenaufrufe, die möglicherweise eine Ausnahme auslösen, in einen Anweisungsblock eingeschlossen werden. Dieser Block beginnt mit dem Kommando

```
TRY.
```

und endet mit

```
ENDTRY.
```

Innerhalb dieses Anweisungsblocks werden nach dem Methodenaufruf mittels des Kommandos

```
CATCH class_exception [INTO reference].
```

weitere Anweisungsblöcke innerhalb des `TRY`-Blocks eingeleitet, die jeweils dann ausgeführt werden, wenn eine Ausnahme auftritt, die eine Instanz der nach `CATCH` angegebenen Klasse ist. Sofern nur das Auftreten der Ausnahme an sich abgefangen werden soll, so reicht die einfache Form der `CATCH`-Anweisung aus. Ausnahmen können jedoch individuell programmiert werden und beliebige Attribute oder Methoden besitzen, die nähere Informationen über die Ausnahmesituation liefern. Um diese Attribute oder Methoden nutzen zu können, ist eine Referenz auf das Ausnahme-Objekt notwendig. Diese Referenz kann mit dem Zusatz `INTO` gefüllt werden. Sie muss vorher deklariert werden und natürlich den korrekten Typ besitzen. Das Schlüsselwort `CATCH` leitet einen Anweisungsblock ein, der durch ein weiteres `CATCH`, ein `ENDTRY` oder die Anweisung `CLEANUP` beendet wird. Innerhalb dieses Anweisungsblocks kann auf die Ausnahmesituation reagiert werden. Worin die Reaktion konkret besteht, hängt sehr von der jeweiligen Anwendung ab. Möglich sind beispielsweise Fehlerausschriften für den Anwender, das Setzen von Default-Werten oder der geordnete Abbruch der Anwendung bzw. der aktuellen Funktion.

Es ist nicht unbedingt notwendig, eine Ausnahme in der Methode zu behandeln, in der sie auftritt. Bei der Deklaration einer Methode muss durch die Anweisung `RAISING` festgelegt werden, welche klassenbasierten Ausnahmen diese Methode nach außen hin auslösen kann. Das bedeutet nicht, dass in der

Methode nur diese Ausnahmen auftreten können. Allerdings werden die nach RAISING aufgeführten Ausnahmen nach oben durchgereicht, falls sie in der eigenen Methode nicht behandelt werden. In diesem Fall ist die rufende Methode für die korrekte Behandlung der Ausnahme zuständig. Dieses Weiterleiten der Ausnahme kann sich über mehrere Ebenen erstrecken. Irgendwann muss eine einmal aufgetretene Ausnahme allerdings auch behandelt werden. Sollte eine Ausnahme daher eine Methode erreichen, die diese nicht weiterleiten kann und sie auch nicht mit CATCH behandelt, tritt eine andere Ausnahme auf (CX_SY_NO_HANDLER), die auf den Programmierfehler hinweist.

Falls eine Ausnahme nicht in der Methode behandelt wird, in der sie auftritt, muss die betroffene Methode möglicherweise trotzdem auf die Ausnahme reagieren, beispielsweise um bereits durchgeführte Änderungen an Daten wieder rückgängig zu machen. Für derartige Aufgaben ist die Anweisung

```
CLEANUP.
```

vorgesehen. Diese Anweisung leitet einen Block ein, der die Aufräumarbeiten ausführt. Sie ist die letzte Block-Anweisung vor dem ENDTRY-Kommando.

4.2.4 Beispiele

Die folgenden Beispiele demonstrieren die objektorientierte Programmierung mit ABAP Objects. Die durch diese Programme gelösten Probleme sind zwar keine typischen Einsatzfälle für objektorientierte Konzepte, bieten dafür aber einen einfachen Einstieg in die Problematik.

Definieren einer Klasse

Das erste Programm stellt eine Klasse vor, die arithmetische Berechnungen ausführt. Damit die hier definierte Klasse auch im nächsten Beispiel genutzt werden kann, wurde deren Definition und Implementierung in ein Include ausgelagert. Nachfolgend zunächst der Inhalt diese Includes. Zur besseren Erläuterung wurden Zeilennummern eingefügt.

```
 1 *---------------------------------------------------------
-*
 2 *   INCLUDE YZ44_CALC1
 3 *---------------------------------------------------------
-*
 4 CLASS calc1 DEFINITION.
 5   PUBLIC SECTION.
 6     METHODS:
 7       constructor
 8         IMPORTING value1 TYPE f
 9                   value2 TYPE f,
10       do_operation
```

```
11         IMPORTING operation TYPE c
12         EXPORTING result TYPE f.
13   PROTECTED SECTION.
14     DATA: op1 TYPE f,
15           op2 TYPE f.
16 ENDCLASS.
17
18 CLASS calc1 IMPLEMENTATION.
19   METHOD constructor.
20     op1 = value1.
21     op2 = value2.
22   ENDMETHOD.
23
24   METHOD do_operation.
25     CASE operation.
26       WHEN '+'.
27         result = op1 + op2.
28         CLEAR: op1, op2.
29       WHEN '-'.
30         result = op1 - op2.
31         CLEAR: op1, op2.
32       WHEN '*'.
33         result = op1 * op2.
34         CLEAR: op1, op2.
35       WHEN '/'.
36         result = op1 / op2.
37         CLEAR: op1, op2.
38     ENDCASE.
39   ENDMETHOD.
40 ENDCLASS.
```

Zwischen den Zeilen 4 und 16 finden Sie die Definition der Klasse. Die Anweisung in Zeile 4 leitet die Definition ein. Sie erhält den Namen `calc1`. In den Zeilen 5 und 13 legt jeweils eine Anweisung den Sichtbarkeitsbereich für die nachfolgenden Definitionen fest. Da sich die beiden Felder `OP1` und `OP2` im `PRIVATE`-Bereich befinden, sind sie später von außen nicht sichtbar. Das Füllen dieser Felder erfolgt beim Erzeugen des realen Objekts. Dazu wird die spezielle Methode `constructor` definiert, die beim Erzeugen eines neuen Objekts automatisch aufgerufen wird. Eine Klasse muss nicht unbedingt einen Konstruktor haben. Er ist in vielen Fällen aber sehr hilfreich, da so beim Anlegen einer realen Instanz sofort deren Attribute auf sinnvolle Werte gesetzt werden können. Ohne den Konstruktor würden Sie für diesen Zweck eine andere Methode benötigen, die Sie separat aufrufen müssten.

Der Konstruktor nimmt zwei Werte entgegen, die er in den beiden privaten Feldern des Objekts ablegt. Im `PUBLIC`-Abschnitt finden Sie die Methode `do_operation`, die im Eingabeparameter `operation` einen Operationscode ent-

gegennimmt und im Parameter `result` das Ergebnis zurückliefert. Der mögliche Problemfall einer Division durch Null soll in diesem Beispiel noch nicht behandelt werden. Die Rückgabe des berechneten Ergebnisses erfolgt wie bei Funktionsbausteinen mittels eines Export-Parameters.

Da die Anweisung `METHODS` die Doppelpunkt-Schreibweise unterstützt, können Sie die beiden genannten Methoden im einem Zug definieren, ohne vor jeder Methode die `METHODS`-Anweisung aufschreiben zu müssen. Allerdings darf deshalb die Definition des Konstruktors nicht mit einem Punkt, sondern nur mit einem Komma abgeschlossen werden.

Die Definitionen legen zunächst nur die Schnittstelle der Methoden fest. Nach der Definition müssen Sie im Implementierungsteil (Zeile 18 – 40) die eben erwähnten Methoden mit Programmcode hinterlegen. Dies erfolgt durch Notieren von Quelltext zwischen den Anweisungen `METHOD` und `ENDMETHOD`. Die Funktionalität der beiden Methoden ist sehr einfach. In `constructor` werden `OP1` und `OP2` mit den Werten der Import-Parameters gefüllt. Die Methode `do_operation` verknüpft die beiden Felder gemäß dem übergebenen Operationszeichen.

Die Klasse `calc1` kann nun in Programmen benutzt werden, um Objekte zu instanziieren. Nachfolgend der Quelltext eines einfachen Programms:

```
 1 REPORT  yz442010.
 2 INCLUDE yz44_calc1.
 3 PARAMETERS: a(20),
 4             b(20),
 5             op.
 6 DATA: r  TYPE REF TO calc1,
 7       f1 TYPE f,
 8       f2 TYPE f.
 9
10 START-OF-SELECTION.
11  TRANSLATE a USING ',.'.
12  TRANSLATE b USING ',.'.
13  f1 = a.
14  f2 = b.
15
16  CREATE OBJECT r EXPORTING value1 = f1
17                            value2 = f2.
18
19  CALL METHOD r->do_operation
20    EXPORTING
21      operation = op
22    IMPORTING
23      result = f1.
24
25   WRITE: / f1 EXPONENT 0.
```

Im Hauptprogramm wird zunächst das Include mit der Klassendefinition eingebunden. Anschließend werden drei Parameter definiert, um die beiden nummerischen Eingabewerte und das Operationszeichen entgegennehmen zu können.

Um die Klasse nutzen zu können, muss eine Objektreferenz erzeugt und später mit einem Objekt verknüpft werden. Den ersten Punkt, das Erzeugen der Referenz, übernimmt die Anweisung in Zeile 6. Diese Anweisung erzeugt eine Referenz `r`, die auf ein Objekt der Klasse `calc1` zeigen kann.

Die Definition und Implementierung einer Klasse stellen eigenständige Anweisungsblöcke dar, ähnlich wie Unterprogrammdeklarationen. Anweisungen nach einem solchen Block werden vom ABAP-Interpreter nur dann gefunden, wenn explizit der Beginn eines neuen Anweisungsblocks programmiert wird. Dies erfolgt in Zeile 10 durch die Anweisung `START-OF-SELECTION`. Unmittelbar nach dieser Anweisung werden die als Zeichenkette vorliegenden Eingabeparameter in gebrochene Zahlen gewandelt. Dies erfolgt durch eine implizite Typumwandlung in den Zeilen 13 und 14. Zuvor wird sicherheitshalber der korrekte Dezimaltrenner gesetzt.

Unmittelbar danach in Zeile 16 wird ein Objekt erzeugt. Dies erfolgt durch Verwendung der Objektreferenz. Da die Objektreferenz mit einem Bezug auf eine bestimmte Klasse programmiert werden muss, kann beim Erzeugen mit `CREATE OBJECT` ein erneuter Bezug auf die Klasse entfallen. Da die verwendete Klasse einen Konstruktor besitzt, dessen Import-Parameter Zwangsparameter sind, muss die `CREATE OBJECT`-Anweisung mit einem `EXPORTING`-Teil versehen werden, in dem die beiden nummerischen Werte übergeben werden.

Die eigentliche Berechnung wird durch den Aufruf von `do_operation` in der Zeile 19 ausgeführt. Der Aufruf selbst ähnelt dem Aufruf eines Funktionsbausteins. Allerdings müssen Sie den Namen des Objekts und den Namen der Methode, getrennt durch die beiden Zeichen `->`, angeben. Unmittelbar nach dem Berechnen wird das Ergebnis ausgegeben.

Vererbung

Das nächste Beispiel demonstriert das Prinzip der Vererbung. Innerhalb der `do_operation`-Methode der Klasse `calc1` wird die Division durch 0 nicht abgefangen und löst somit einen Laufzeitfehler aus. Dieses Problem soll nun durch Programmierung einer neuen Klasse `calc2` behoben werden, in der die genannte Methode neu implementiert wird. Alle anderen Eigenschaften sollen von der vorhandenen Klasse `calc1` geerbt werden. Definition und Implementierung der neuen Klasse erfolgen direkt im Hauptprogramm, dessen Listing Sie nachfolgend finden.

```
1 REPORT  yz442020.
2 INCLUDE yz44_calc1.
3
4 CLASS calc2 DEFINITION INHERITING FROM calc1.
```

```
 5   PUBLIC SECTION.
 6     METHODS do_operation REDEFINITION.
 7 ENDCLASS.
 8
 9 CLASS calc2 IMPLEMENTATION.
10   METHOD do_operation.
11     IF operation <> '/' OR op2 <> 0.
12       CALL METHOD super->do_operation
13         EXPORTING
14           operation = operation
15         IMPORTING
16           result = result.
17     ELSE.
18       CLEAR result.
19     ENDIF.
20   ENDMETHOD.
21 ENDCLASS.
22
23 PARAMETERS: a(20),
24             b(20),
25             op.
26 DATA: r  TYPE REF TO calc2,
27       f1 TYPE f,
28       f2 TYPE f.
29
30 START-OF-SELECTION.
31   TRANSLATE a USING ',.'.
32   TRANSLATE b USING ',.'.
33   f1 = a.
34   f2 = b.
35
36   CREATE OBJECT r EXPORTING value1 = f1
37                             value2 = f2.
38
39   CALL METHOD r->do_operation
40     EXPORTING
41       operation = op
42     IMPORTING
43       result = f1.
44
45   WRITE: / f1 EXPONENT 0.
```

Zunächst wird in Zeile 2 das Include mit der Klasse `calc1` eingebunden. Dadurch ist diese Klasse im aktuellen Programm vorhanden. In der Zeile 4 wird die Klasse `calc2` definiert. Durch den Zusatz `INHERITING FROM` wird festgelegt, dass die neue Klasse alle Eigenschaften der Klasse `calc1` erben soll. Sollte die Definition der Klasse `calc2` keine weiteren Anweisungen enthalten und auf

die CLASS-Anweisung sofort die ENDCLASS-Anweisung folgen, würde sich die Klasse calc2 ebenso verhalten wie die Klasse calc1. Obwohl von der neuen Klasse Referenzen angelegt und benutzt werden können, würden Aufrufe der beiden Methoden sofort an die Klasse calc1 durchgereicht. Nun wird in Zeile 6 aber festgelegt, dass die Methode do_operation neu definiert werden soll. Da durch die Neudefinition die Schnittstelle nicht geändert werden kann, die der entsprechenden Methode aus der Klasse calc1 also weiterhin gültig ist, muss die Schnittstelle nicht noch einmal notiert werden. Allerdings ist es erforderlich, den Quellcode der neuen Methode im Implementierungsteil anzugeben. Dabei gäbe es in diesem speziellen Fall zwei Möglichkeiten. Sie könnten den gesamten Quelltext der Methode aus der Klasse calc1 übernehmen und nur im CASE-Zweig für die Division eine zusätzliche Prüfung einbauen. Sie können aber auch zuerst die Prüfung der Operatoren und Operationszeichen vornehmen und, falls alle Werte korrekt sind, die do_operation-Methode der Klasse calc1 aufrufen. Letzgenannte Variante ist die elegantere, da sie es Ihnen erspart, vorhandene Funktionalität neu zu implementieren.

In Zeile 11 des Quelltextes finden Sie die entsprechende Prüfung. Zeile 12 enthält den Aufruf der Methode der übergeordneten Klasse. Da beide Methoden denselben Namen haben, müssen Sie durch Verwendung der Pseudo-Referenz super angeben, dass Sie auf die Methode der Basisklasse zugreifen möchten. Ohne Angabe dieser Referenz würde der ABAP-Interpreter die do_operation-Methode der Klasse calc2 aufrufen und so eine Endlosschleife erzeugen.

Der Rest des Programms entspricht, bis auf den Typ der Referenz r, dem ersten Beispiel. Die Funktionsweise erschließt sich am besten, wenn Sie das gesamte Programm im Einzelschrittmodus debuggen.

Zuweisungskompatibilität

Durch Anwendung der eben demonstrierten Vererbung werden in der Praxis von einer Basisklasse mehrere unterschiedliche Sub-Klassen abgeleitet. Dabei stellt jede Klasse prinzipiell einen eigenen Datentyp dar. Die bei Zuweisungen von Objekten stattfindenden Typprüfungen sind allerdings nicht völlig mit den Prüfungen vergleichbar, die bei Zuweisungen zwischen elementaren Feldern durchgeführt werden. Objekte unterschiedlicher Klassen können daher unter bestimmten Voraussetzungen an Objektreferenzen eines anderen Typs zugewiesen werden. Allgemein gilt, dass ein Referenzfeld auf alle Objekte zeigen kann, die zur Klasse des Referenzfeldes gehören sowie auf alle Objekte, die von dieser Basisklasse abgeleitet sind. Das gilt unabhängig von der Vererbungstiefe.

Das nachfolgende Programm soll die Zuweisungsmöglichkeiten und die damit verbundenen Effekte darstellen. Ausgangspunkt ist die Klasse c1, die eine einfache Zeichenkette aufbewahrt. Der Wert dieses Attributs kann über die Methode set_value gesetzt und mit get_value ausgelesen werden. Von dieser Klasse werden die beiden Klassen c11 und c12 abgeleitet. Beide definieren die set_value-Methode neu. Innerhalb der neuen Methoden wird die Zeichenkette

entweder in Groß- oder in Kleinbuchstaben gewandelt. Außerdem definiert die Klasse `c12` eine weitere Methode `clear_value`.

Innerhalb des Datendeklarationsteils werden für jede der drei Klassen jeweils eine Objektreferenz erstellt sowie zwei Zeichenkettenfelder deklariert. Zu Beginn des Programms werden drei Objekte instanziiert und getestet. Sie erhalten drei Zeichenketten, jeweils eine in Originalform und eine komplett in Groß- oder Kleinbuchstaben. Bis zu diesem Punkt bietet das Programm noch keine neuen Erkenntnisse. Hier zunächst der Quelltext:

```
 1 REPORT yz442030.
 2 TYPES mychar(60).
 3
 4 CLASS c1 DEFINITION.
 5   PUBLIC SECTION.
 6     METHODS: set_value IMPORTING imp TYPE mychar,
 7              get_value EXPORTING exp TYPE mychar.
 8   PROTECTED SECTION.
 9     DATA value TYPE mychar.
10 ENDCLASS.
11
12 CLASS c1 IMPLEMENTATION.
13   METHOD set_value.
14     value = imp.
15   ENDMETHOD.
16
17   METHOD get_value.
18     exp = value.
19   ENDMETHOD.
20
21 ENDCLASS.
22
23 CLASS c11 DEFINITION INHERITING FROM c1.
24   PUBLIC SECTION.
25     METHODS: set_value REDEFINITION.
26 ENDCLASS.
27
28 CLASS c11 IMPLEMENTATION.
29   METHOD set_value.
30     CALL METHOD super->set_value EXPORTING imp = imp.
31     TRANSLATE value TO LOWER CASE.
32   ENDMETHOD.
33 ENDCLASS.
34
35 CLASS c12 DEFINITION INHERITING FROM c1.
36   PUBLIC SECTION.
37     METHODS: set_value REDEFINITION,
```

```
38              clear_value.
39 ENDCLASS.
40
41 CLASS c12 IMPLEMENTATION.
42   METHOD set_value.
43     CALL METHOD super->set_value EXPORTING imp = imp.
44     TRANSLATE value TO UPPER CASE.
45   ENDMETHOD.
46
47   METHOD clear_value.
48     CLEAR value.
49   ENDMETHOD.
50 ENDCLASS.
51
52 DATA: r1  TYPE REF TO c1,
53       r11 TYPE REF TO c11,
54       r12 TYPE REF TO c12,
55       cimp TYPE mychar VALUE 'abcDEF',
56       cres TYPE mychar.
57
58 START-OF-SELECTION.
59   CREATE OBJECT r1.
60   CALL METHOD r1->set_value EXPORTING imp = cimp.
61   CALL METHOD r1->get_value IMPORTING exp = cres.
62   WRITE: / cres.
63
64   CREATE OBJECT r11.
65   CALL METHOD r11->set_value EXPORTING imp = cimp.
66   CALL METHOD r11->get_value IMPORTING exp = cres.
67   WRITE: / cres.
68
69   CREATE OBJECT r12.
70   CALL METHOD r12->set_value EXPORTING imp = cimp.
71   CALL METHOD r12->get_value IMPORTING exp = cres.
72   WRITE: / cres.
73
74   r1 = r11.
75   CALL METHOD r1->set_value EXPORTING imp = cimp.
76   CALL METHOD r1->get_value IMPORTING exp = cres.
77   WRITE: / cres.
78
79   r1 = r12.
80   CALL METHOD r1->set_value EXPORTING imp = cimp.
81   CALL METHOD r1->get_value IMPORTING exp = cres.
82   WRITE: / cres.
83
84 * r1 and r12 point to the same object!!!
```

```
85   CALL METHOD r12->set_value
86      EXPORTING imp = 'An other string'.
87   CALL METHOD r1->get_value IMPORTING exp = cres.
88   WRITE: / cres.
89   CALL METHOD r12->clear_value.
90
91 * not possible:
92 *  call method r1->clear_value.
93 *  r11 = r1.
94 *  r12 = r1.
95 *  r11 = r12.
```

Die erste bedeutsame Anweisung ist in Zeile 74 zu finden. Mit dieser Anweisung wird der Objektreferenz `r1` der Inhalt der Objektreferenz `r11` zugewiesen. Das bedeutet nicht, dass der Inhalt des Objekts der Klasse `c11` in das Objekt der Klasse `c1` kopiert wird, sondern dass jetzt die Objektreferenz auf das existierende Objekt der Klasse `r11` zeigt. Die Referenzen `r1` und `r11` zeigen jetzt auf ein und dasselbe Objekt! Das zuvor erzeugte Objekt der Klasse `r1` geht verloren, da keine Referenz mehr existiert, die auf dieses Objekt zeigt. Ohne eine solche Referenz kann nicht mehr auf ein Objekt zugegriffen werden.

Da `r1` jetzt auf ein Objekt der Klasse `c11` zeigt, werden beim nachfolgenden Methodenaufruf (Zeile 75 und 76) auch die Methoden dieses Objekts ausgeführt, die Zeichenkette also in Kleinbuchstaben gewandelt. Auf dieselbe Art und Weise kann `r1` natürlich auch auf ein Objekt der Klasse `c12` zeigen, wie die nachfolgenden Anweisungen (Zeilen 79 – 82) zeigen. Da `r1` und `r12` nun wieder auf dasselbe Objekt zeigen, ist es egal, welche Objektreferenz beim Aufruf der Methoden verwendet wird. Auch dafür existiert im Listing ein Beispiel (Zeilen 85 – 87).

Die Zuweisungen von `r11` bzw. `r12` zu `r1` sind möglich, weil die Klassen `c11` und `c12` von der Klasse `c1` erben. Dabei können sie die Definition von `c1` erweitern, aber nicht einschränken. Die in der Definition der Klasse `c1` enthaltenen Elemente sind somit auf jeden Fall in den Sub-Klassen enthalten und können daher auch über Objektreferenzen angesprochen werden, die nur die Elemente der Basisklasse kennen. Das bedeutet zwangsläufig auch, dass über die Objektreferenz `r1` keine Elemente angesprochen werden können, die erst in den abgeleiteten Klassen definiert werden. Im Beispiel ist die Methode `clear_value` der Klasse `c12` ein solches Objekt. Obwohl die Objektreferenz `r1` nach der Zuweisung von `r12` auf ein Objekt der Klasse `c12` zeigt, kann über `r1` die Methode `clear_value` nicht angesprochen werden. Ein derartiger Versuch führt zu einem Syntaxfehler.

Die Typprüfungen erfolgen übrigens nicht auf der Basis einfacher Namensvergleiche, sondern berücksichtigen auch die Vererbungshierarchie. Aus diesem Grund führt auch die Anweisung

```
r11 = r12.
```

zu einem Fehler, obwohl in `r12` alle Elemente enthalten sind, die laut Definition der für den Typ von `r11` maßgeblichen Klasse `c11` enthalten sind.

Auch die Zuweisungen

```
r11 = r1.
```

bzw.

```
r12 = r1.
```

sind nicht korrekt. Da vererbte Klassen die Definition der Basisklasse erweitern können, würden die beiden Anweisungen den Zugriff auf Elemente erlauben, über die das zugewiesene Objekt nicht verfügt. Versuche dieser Art müssen natürlich unterbunden werden.

Vereinfachte Parameterübergabe

Beim Aufruf der Methoden wurden bisher die an Funktionsbausteine erinnernden Konventionen eingehalten. Für Methoden existieren aber einige Erleichterungen, die nachfolgend am praktischen Beispiel demonstriert werden sollen. Dazu wird zwar wieder eine Klasse verwendet, die eine arithmetische Operation ausführt, allerdings kommt keine Vererbung zum Einsatz. Die Klasse `calc3` wird komplett neu implementiert und besitzt andere Schnittstellen als die Klassen `calc1` und `calc2`.

```
 1 REPORT  yz442040.
 2 CLASS calc3 DEFINITION.
 3   PUBLIC SECTION.
 4     METHODS:
 5       constructor
 6         IMPORTING value1 TYPE f
 7                   value2 TYPE f,
 8       set_op
 9         IMPORTING op TYPE c,
10       result
11         RETURNING value(result) TYPE f.
12   PROTECTED SECTION.
13     DATA: op1 TYPE f,
14           op2 TYPE f,
15           op  TYPE c.
16 ENDCLASS.
17
18 CLASS calc3 IMPLEMENTATION.
19   METHOD constructor.
20     op1 = value1.
21     op2 = value2.
```

```
22   ENDMETHOD.
23
24   METHOD result.
25     CASE op.
26       WHEN '+'.
27         result = op1 + op2.
28         CLEAR: op1, op2.
29       WHEN '-'.
30         result = op1 - op2.
31         CLEAR: op1, op2.
32       WHEN '*'.
33         result = op1 * op2.
34         CLEAR: op1, op2.
35       WHEN '/'.
36         IF op2 <> 0.
37           result = op1 / op2.
38           CLEAR: op1, op2.
39         ENDIF.
40     ENDCASE.
41   ENDMETHOD.
42
43   METHOD set_op.
44     me->op = op.
45   ENDMETHOD.
46
47 ENDCLASS.
48
49 PARAMETERS: a(20),
50             b(20),
51             op.
52 DATA: r  TYPE REF TO calc3,
53       f1 TYPE f,
54       f2 TYPE f.
55
56 START-OF-SELECTION.
57   TRANSLATE a USING ',.'.
58   TRANSLATE b USING ',.'.
59   f1 = a.
60   f2 = b.
61
62   CREATE OBJECT r EXPORTING value1 = f1
63                             value2 = f2.
64   CALL METHOD r->set_op( op ).
65
66   f1 = r->result( ).
67
68  WRITE: / f1 EXPONENT 0.
```

Die neue Klasse unterscheidet sich von den vorangegangenen zunächst durch ihre Funktionsweise. Die beiden Operanden werden immer noch dem Konstruktor übergeben. Allerdings steht zum Setzen des Operationszeichens eine eigene Methode `set_op` zur Verfügung. Sie legt das Operationszeichen in einem Attribut der Klasse ab. Das zusätzliche Attribut für das Operationszeichen ist ebenfalls eine Erweiterung gegenüber den vorangegangenen Varianten. Das Resultat der Operation wird schließlich durch die Methode `result` geliefert, die über keine Eingabeparameter verfügt. Sie liefert nur einen einzigen Parameter zurück und benutzt dazu nicht die `EXPORTING`-, sondern die `RETURNING`-Anweisung.

Die Quelltexte des Konstruktors und der `result`-Methode liefern keine neuen Kenntnisse, da sie den bereits bekannten Methoden entsprechen. Nur innerhalb der `set_op`-Methode (Zeile 44) taucht eine neue Syntax-Variante auf. Der Eingabeparameter der Methode sowie das Attribut des Objekts besitzen den selben Namen `op`. Der Eingabeparameter verschattet dadurch das Attribut. Dieses ist somit nicht ohne weiteres verfügbar. Der Zugriff auf das Attribut ist nur durch Verwendung der Referenz `me` möglich. Diese Referenz zeigt stets auf das aktuelle Objekt und ermöglich so den gezielten Zugriff auf verschattete Elemente.

Die nächste syntaktische Besonderheit ist in Zeile 64 zu finden. Da die Methode `set_op` nur über einen einzigen Parameter verfügt, kann der Wert für diesen Parameter ohne `EXPORTING`-Klausel und ohne Namensreferenz übergeben werden, da eine eindeutige Zuordnung problemlos möglich ist. Beachten Sie bei dieser Form des Aufrufs aber, dass jeweils ein Leerzeichen zwischen den Klammern und dem formalen Parameter erforderlich ist.

Noch einfacher – für den ABAP-Programmierer aber gewöhnungsbedürftig – ist der Aufruf der `result`-Methode in Zeile 66. Diese Methode besitzt keine Eingabeparameter und liefert ihr Ergebnis in einem `RETURNING`-Parameter zurück. Durch die fehlenden Eingabeparameter kann die Parameterliste beim Aufruf leer bleiben. Es sind lediglich die beiden runden Klammern zu notieren, die aber unbedingt durch ein Leerzeichen voneinander getrennt werden müssen! Die Rückgabe des Ergebnisses über einen `RETURNING`-Parameter eröffnet die Möglichkeit, in bestimmten Fällen auf die `CALL METHOD`-Anweisung zu verzichten. Die Zuweisung des Ergebnisses zu einem anderen Feld zählt zu diesen Ausnahmen. In Zeile 64 wird der `RETURNING`-Parameter daher in das Feld `f1` geschrieben.

Interfaces

In den bisherigen Beispielen wurden die Schnittstellen der Klassen direkt in der Klasse selbst definiert oder durch den Vererbungsmechanismus von einer übergeordneten Klasse vorgegeben. Dadurch entsteht ein Klassenbaum, für dessen Elemente Zuweisungskompatibilität nur aufwärts innerhalb eines Zweiges besteht. Mitunter ist es wünschenswert oder notwendig, völlig unterschiedliche Klassen mit gleichnamigen Methoden zu versehen, die intern unterschiedlich funktionieren, nach außen hin aber eine identische Schnittstelle besitzen und vergleichbare Aufgaben erfüllen. Dieses Verhalten wird als Polymorphismus

bezeichnet. In der Praxis könnten derartige Methoden die Attribute eines Objekts in die Datenbank schreiben oder aber einen Dialog aufrufen, der das Editieren der Attribute ermöglicht. Diese Form der Programmierung soll es erleichtern, generische Anwendungen zu schreiben, die mit Objekten unterschiedlicher Klassen arbeiten können. Das Problem besteht darin, dass Sie für jedes zu bearbeitenden Objekt eine Objektreferenz benötigen, deren Typ der jeweiligen Klasse entspricht. Sollten die betroffenen Klassen in unterschiedlichen Zweigen des Klassenbaums liegen und keinen gemeinsamen Vorgänger besitzen, der die gewünschte Methode in seiner Schnittstelle anbietet, benötigen Sie für jede in Frage kommende Klasse eine eigene Referenzvariable. Das kann sehr problematisch werden, wenn viele Klassen betroffen sind oder aber Klassen behandelt werden sollen, die noch gar nicht existieren. Generische Anwendungen können Sie auf diese Weise nicht schreiben.

Das folgende Beispiel demonstriert den Einsatz eines Interface. Dieses Interface definiert nur eine einzige Methode mit Namen `show`. Diese Methode soll später in den konkreten Objekten benutzt werden, um den Namen und den Typ eines Entwicklungsobjekts auszugeben. Das Interface wird von zwei Klassen (`cprog` und `ctable`) implementiert, die zueinander nicht zuweisungskompatibel sind. Eine Klasse soll die Namen von Programmen aufnehmen, die andere den Namen von Tabellen. Zusätzlich zu der über das Interface definierten Methode besitzt jede dieser beiden Klassen eine weitere Methode, mit der die Namen gesetzt werden können.

Die Aufgabe des folgenden Programms besteh darin, aus der Tabelle `TADIR` einige Programm- und Tabellennamen zu ermitteln und für jeden der gelesenen Datensätze ein Objekt der passenden Klasse zu instanziieren. Diese Objekte sollen in einer internen Tabelle gespeichert werden. Anschließend soll in einer Schleife über die interne Tabelle jedes Objekt angesprochen werden und den Namen und den Typ des ihm zugeordneten Objekts ausgeben. Nachfolgend zunächst der Quelltext des Programms.

```
 1 REPORT  yz442050.
 2 INTERFACE ishow.
 3   METHODS show.
 4 ENDINTERFACE.
 5
 6 CLASS cprog DEFINITION.
 7  PUBLIC SECTION.
 8    METHODS:
 9      set_progname
10        IMPORTING prog TYPE sobj_name.
11    INTERFACES ishow.
12  PROTECTED SECTION.
13    DATA: progname TYPE programm.
14 ENDCLASS.
15
```

```
16 CLASS ctable DEFINITION.
17   PUBLIC SECTION.
18     METHODS:
19       set_tabname
20         IMPORTING table TYPE sobj_name.
21     INTERFACES ishow.
22   PROTECTED SECTION.
23     DATA: tablename TYPE tabname.
24 ENDCLASS.
25
26 CLASS cprog IMPLEMENTATION.
27   METHOD set_progname.
28     progname = prog.
29   ENDMETHOD.
30
31   METHOD ishow~show.
32     WRITE: / 'Program: ', progname.
33   ENDMETHOD.
34 ENDCLASS.
35
36 CLASS ctable IMPLEMENTATION.
37   METHOD set_tabname.
38     tablename = table.
39   ENDMETHOD.
40
41   METHOD ishow~show.
42     WRITE: / 'Table  : ', tablename.
43   ENDMETHOD.
44 ENDCLASS.
45
46 TYPES:
47   t_objref TYPE REF TO ishow,
48   ti_objects
49     TYPE STANDARD TABLE OF t_objref
50     INITIAL SIZE 0.
51
52 DATA:
53   ref_prog TYPE REF TO cprog,
54   ref_tabl TYPE REF TO ctable,
55   ref_obj  TYPE REF TO ishow,
56   iobjects TYPE ti_objects WITH HEADER LINE.
57
58 TABLES:
59   tadir.
60
61 START-OF-SELECTION.
62   SELECT * FROM tadir
```

```
63     WHERE devclass = 'SBF_WEB'
64       AND ( object = 'PROG' OR object = 'TABL' )
65     ORDER BY obj_name.
66
67     IF tadir-object = 'PROG'.
68       CREATE OBJECT ref_prog.
69       CALL METHOD ref_prog->set_progname
70         EXPORTING prog = tadir-obj_name.
71       ref_obj = ref_prog.
72     ELSE.
73       CREATE OBJECT ref_tabl.
74       CALL METHOD ref_tabl->set_tabname
75         EXPORTING table = tadir-obj_name.
76
77       ref_obj = ref_tabl.
78     ENDIF.
79     APPEND ref_obj TO iobjects.
80
81   ENDSELECT.
82
83  LOOP AT iobjects.
84    ref_obj = iobjects.
85    CALL METHOD iobjects->show( ).
86  ENDLOOP.
```

Im Deklarationsteil der Anwendung werden drei Objektreferenzen deklariert, jeweils eine für die beiden Klassen und eine für das Interface. Obwohl die Deklarationen gleich aussehen, können Sie später nur die auf Klassen gerichteten Referenzfelder benutzen, um Objekte zu instanziieren. Diese beiden Referenzfelder sind zueinander nicht zuweisungskompatibel. Allerdings können Sie den Inhalt beider Felder der Interface-Referenz `ref_obj` zuweisen, da beide Klassen die durch das Interface vorgegebenen Schnittstelle implementieren. Über die Interface-Referenz sind dann aber nur die Teile der Objekte zugänglich, die durch das Interface definiert werden. Das ist in diesem Fall nur die Methode `show`. Das reicht für die zu erfüllende Aufgabe aber auch aus. In der `LOOP`-Schleife über die interne Tabelle können Sie ohne Kenntnis des Objekttyps die Methode `show` aufrufen. Da hinter dieser Referenz aber das komplette Objekt liegt, kann die aufgerufenen Methode die korrekten Daten ausgeben.

Events

Bisher wurden die Objekte direkt angesprochen, um ihre Funktionalität zu nutzen. Dies erfordert, dass stets eine Referenz auf ein Objekt zeigen muss. Falls eine spezielle Funktion ausgelöst werden soll, muss das anzusprechende Objekt bekannt sein. Da objektorientierte Modelle sehr komplex werden können, würde das ebenso komplexe Datenstrukturen zur Verwaltung der Objekte erfor-

dern. Aber gerade diese komplexen Datenstrukturen sollen durch die objektorientierte Programmierung vereinfacht werden. Aus diesem Grund bieten die Events die Möglichkeit, Objekte problemlos miteinander kommunizieren zu lassen. Grundgedanke ist es, dass ein Objekt eine Nachricht aussendet, ohne den konkreten Empfänger zu kennen. Für jede mögliche Nachricht wird ein empfangendes Objekt erstellt, das auf die gewünschte Weise reagiert.

Der Event-Mechanismus wird beispielsweise sehr häufig in grafischen Oberflächen eingesetzt. Dort können Menüeinträge einen Event auslösen, der durch einen Event-Handler bearbeitet wird.

Das folgende Beispiel demonstriert die Anweisungen, die zum Aussenden und Behandeln eines Events erforderlich sind.

```
 1 REPORT yz442060.
 2 TYPES mychar TYPE tadir-obj_name.
 3 CLASS cevent DEFINITION.
 4  PUBLIC SECTION.
 5    EVENTS: modify_content EXPORTING value(upp) TYPE c,
 6            show_content.
 7    METHODS: create_event_modify IMPORTING value(upp) TYPE c,
 8             create_event_show.
 9 ENDCLASS.
10
11 CLASS cevent IMPLEMENTATION.
12   METHOD create_event_modify.
13     RAISE EVENT modify_content EXPORTING upp = upp.
14   ENDMETHOD.
15
16   METHOD create_event_show.
17     RAISE EVENT show_content.
18   ENDMETHOD.
19 ENDCLASS.
20
21 CLASS cstring DEFINITION.
22   PUBLIC SECTION.
23     METHODS: set_value IMPORTING imp TYPE mychar,
24              modify FOR EVENT modify_content OF cevent
25                IMPORTING upp,
26              show   FOR EVENT show_content   OF cevent.
27   PRIVATE SECTION.
28     DATA value TYPE mychar.
29 ENDCLASS.
30
31 CLASS cstring IMPLEMENTATION.
32   METHOD set_value.
33     value = imp.
34   ENDMETHOD.
```

```
35
36   METHOD modify.
37     IF upp = 'X'.
38       TRANSLATE value TO UPPER CASE.
39     ELSE.
40       TRANSLATE value TO LOWER CASE.
41     ENDIF.
42   ENDMETHOD.
43
44   METHOD show.
45     WRITE: / value.
46   ENDMETHOD.
47 ENDCLASS.
48
49 DATA: re TYPE REF TO cevent,
50       rs TYPE REF TO cstring.
51
52 TABLES tadir.
53
54 START-OF-SELECTION.
55   CREATE OBJECT re.
56   SELECT * FROM tadir
57     WHERE devclass = 'SBF_WEB'
58       AND object   = 'DTEL'.
59
60     CREATE OBJECT rs.
61     CALL METHOD rs->set_value
62       EXPORTING imp = tadir-obj_name.
63     SET HANDLER rs->modify FOR re.
64     SET HANDLER rs->show FOR re.
65   ENDSELECT.
66
67   CALL METHOD re->create_event_modify EXPORTING upp = ' '.
68   CALL METHOD re->create_event_show.
69   CALL METHOD re->create_event_modify EXPORTING upp = 'X'.
70   CALL METHOD re->create_event_show.
```

Events können nur innerhalb einer Klasse ausgelöst werden. Aus diesem Grund wird zunächst eine Klasse `cevent` definiert. Die `EVENTS`-Anweisung dieser Klasse deklariert die Events `modify_content` und `show_content`. Der `modify_content`-Event verfügt dabei über einen Parameter. Die Events sind innerhalb der Klasse, in der sie definiert werden, auszulösen. Dies erfolgt in diesem Beispiel durch zwei speziell dafür vorgesehene Methoden `create_event_modify` und `create_event_show`. Es ist nicht notwendig, Events immer in speziell dazu geschriebenen Methoden auszulösen. Wesentlich häufiger kommt es vor, dass Events innerhalb von Methoden ausgelöst werden, die auch andere Funktionalität enthalten.

Als Empfänger der Events sollen zwei Methoden der Klasse `cstring` dienen. Die Objekte dieser Klasse nehmen eine Zeichenkette auf, deren Schreibweise über einen der Events modifiziert werden kann. Außerdem sollen die Objekte auf Anforderung ihren Inhalt auf dem Bildschirm ausgeben können.

Damit eine Methode als Bearbeitungsfunktion für einen Event dienen kann, muss sie zunächst als so genannter Event-Handler deklariert werden. Das erfolgt durch den Zusatz `FOR EVENT` in der Methodendeklaration. Dabei müssen der Name des Events und die Klasse, zu der dieser Event gehört, angegeben werden. Wenn ein Event Parameter besitzt, müssen diese in der behandelnden Methode natürlich ebenfalls deklariert werden. Dazu dient eine `IMPORTING`-Anweisung, die aber als Option zur `FOR EVENT`-Anweisung zu sehen ist und nicht der normalen `IMPORTING`-Anweisung der Methodendklaration entspricht. Die Parameterliste dieser `IMPORTING`-Anweisung muss der Parameterliste des Events entsprechen. Im Beispiel dient der Parameter `upp` des Events dazu, die Konvertierung der Zeichenkette in Großbuchstaben auszulösen.

Die eben beschriebenen Angaben dienen zunächst nur dazu, die Behandlung des Events prinzipiell zu ermöglichen. Sie legen fest, dass eine Methode einer Klasse als Behandlungsroutine für einen Event dienen kann, dies aber nicht zwingend ist. Zur Laufzeit muss daher noch festgelegt werden, welche Methode auf welche Nachricht reagiert. Dabei müssen Sender und Empfänger konkret, also als Objekt, benannt werden. Das bedeutet, dass in Abhängigkeit vom sendenden Objekt auf einen Event unterschiedliche Empfänger reagieren können. Diese Forderung kann abgemildert werden, indem nicht zwischen unterschiedlichen Sendern eines Events unterschieden wird (Zusatz `FOR ALL INSTANCES`).

Im Beispiel wird in Zeile 55 zunächst ein Objekt der Klasse `cevent` instanziiert. Dieses Objekt dient später zum Auslösen der Events. Im Anschluss daran werden einige Datenelemente einer ausgewählten Entwicklungsklasse gelesen (Zeile 56 – 58). Für jedes dieser Elemente wird ein Objekt der Klasse `cstring` erzeugt und mit dem Namen des Datenelementes belegt. In Zeile 63 und 64 erfolgt dann die konkrete Zuweisung der Ereignisbehandlungsmethoden zum auslösenden Ereignis. Beachten Sie bitte, dass die erzeugten Objekte nicht in einer internen Tabelle gesammelt werden, sondern die Objektreferenz immer wieder überschrieben wird.

Der Aufruf einer Methode der Klasse `cevent` in Zeile 67 löst schließlich den ersten Event aus. Da alle Instanzen der Klasse `cstring` diesen Event behandeln sollen, werden alle Zeichenketten in Kleinbuchstaben gewandelt. Der nachfolgende, in Zeile 68 ausgelöste Event fordert dann von allen `cstring`-Objekten die Ausgabe des Wertes auf dem Bildschirm. Die nachfolgenden beiden Kommandos demonstrieren das verfahren nochmals, wobei nun aber die Umwandlung in Großbuchstaben gefordert wird.

Klassenbasierte Ausnahmen

Die klassenbasierten Ausnahmen werden in drei Untergruppen eingeordnet. Für jede Untergruppe steht eine Basisklasse bereit, von der die erforderlichen eigenen Ausnahmen abgeleitet werden können. Ursprung aller Ausnahmeklassen ist die Klasse CX_SY_ROOT.

Die erste Untergruppe wird von der Basisklasse CX_STATIC_CHECK abgeleitet. Die Behandlung dieser Ausnahmen wird bereits vom Compiler überprüft. Voraussetzung für die statische Überprüfung ist, dass das Aussenden von derartigen Ausnahmen durch eine RAISING-Klausel realisiert werden muss.

Ausnahmen der Gruppe CX_DYNAMIC_CHECK bzw. deren Behandlung wird nicht vom Compiler überwacht. Diese Ausnahmen müssen nicht zwangsläufig durch eine RAISING-Klausel weitergereicht werden.

Die letzte Gruppe umfasst die Ausnahmen der Klasse CX_NO_CHECK. Diese werden weder statisch noch dynamisch überprüft und müssen somit auch nicht explizit durch eine RAISING-Klausel weitergegeben werden. Vielmehr reicht das Laufzeitsystem diese Ausnahmen selbstständig weiter, bis eine Behandlung erfolgt. Findet keine Behandlung statt, entsteht ein Laufzeitfehler.

Das erste Programmbeispiel demonstriert die prinzipielle Verfahrensweise zum Auslösen und Behandeln einer Ausnahme.

```
 1 REPORT yz440070.
 2
 3 CLASS c_myex1 DEFINITION INHERITING FROM cx_static_check.
 4 ENDCLASS.                    "c_myex1 DEFINITION
 5
 6 CLASS cexcept_1 DEFINITION.
 7   PUBLIC SECTION.
 8     METHODS:
 9       throw_exception IMPORTING param TYPE I
10                       RAISING c_myex1.
11 ENDCLASS.                    "cexcept_1 DEFINITION
12
13 CLASS cexcept_1 IMPLEMENTATION.
14   METHOD throw_exception.
15     WRITE: / param.
16     IF param = 3.
17       RAISE EXCEPTION TYPE c_myex1.
18     ENDIF.
19   ENDMETHOD.                    "throw_exception
20 ENDCLASS.                    "cexcept_1 IMPLEMENTATION
21
22 DATA: rexcept TYPE REF TO cexcept_1.
23
24 START-OF-SELECTION.
25   CREATE OBJECT rexcept.
26   DO 5 TIMES.
```

```
27      TRY.
28        CALL METHOD rexcept->throw_exception
29             EXPORTING
30               param = sy-index.
31      CATCH c_myex1.
32        WRITE: 'Exception occured: c_myex1'.
33      ENDTRY.
34    ENDDO.
```

Wenn eine eigene Ausnahmen ausgelöst werden soll, muss dazu in den meisten Fällen eine eigene Klasse definiert werden. Die eigene Ausnahme muss dazu von einer der beiden Klassen `CX_STATIC_CHECK` oder `CX_DYNAMIC_CHECK` abgeleitet werden. Sofern die eigene Ausnahmeklasse keine eigenen Methoden besitzen soll, reicht die Definition, ein Implementierungsteil wird nicht erforderlich (Zeile 3 und 4).

Das Auslösen einer klassenbasierten Ausnahme erfolgt in einem Objekt. In den Zeilen 6 bis 11 wird eine Klasse definiert, die über eine einzige Methode `throw_exception` verfügt. Die Implementierung dieser Klasse in den Zeilen 13 bis 20 zeigt, dass die Methode `throw_exception` eine Ausnahme erzeugt, wenn der übergebene Parameter den Wert 3 besitzt.

Um die Methode testen zu können, muss zunächst ein Objekt der Klasse `cexcept_1` erzeugt werden. Dieses wird dann innerhalb einer Schleife benutzt, um die Methode `throw_exception` mit unterschiedlichen Parametern aufrufen zu können. Damit die Ausnahme abgefangen werden kann, muss der Methodenaufruf in einen `TRY-ENDTRY`-Block eingeschlossen werden (27-33). Die Kommandos nach der `CATCH`-Anweisung werden nur ausgeführt, wenn die Ausnahme auftritt, also beim Aufruf der Methode mit dem Parameter 3.

Bild 4.1 zeigt das Ergebnis des Reports.

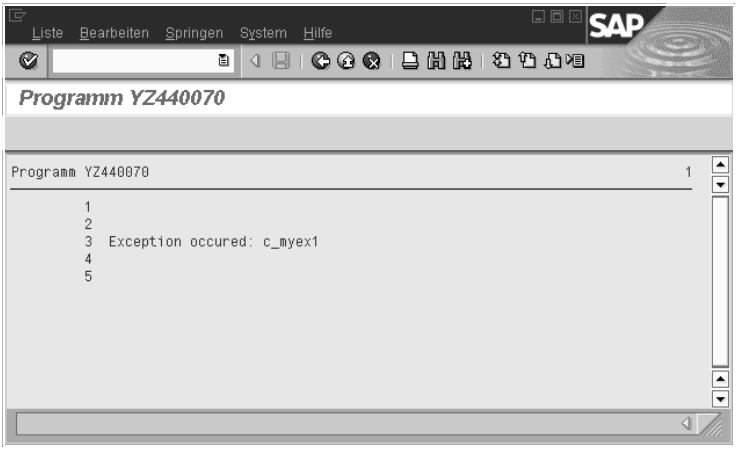

Abbildung 4.1 **© SAP AG**
Ausgabe des Reports zur Ausnahmebehandlung

Das zweite Beispiel zum Thema Ausnahmen demonstriert die Eigenschaften der Ausnahme-Klassen sowie die Möglichkeiten zu deren Anpassung an das konkrete Umfeld. So verfügen Ausnahmeklassen über einige Methoden, die von den Basisklassen geerbt werden. Außerdem können eigene Ausnahmeklassen mit Attributen versehen werden.

```
 1 REPORT yz440080.
 2
 3 CLASS c_myex2 DEFINITION INHERITING FROM cx_static_check.
 4   PUBLIC SECTION.
 5     DATA wrong_value TYPE i.
 6     METHODS constructor IMPORTING err TYPE i.
 7 ENDCLASS.                    "c_myex2 DEFINITION
 8
 9 CLASS c_myex2 IMPLEMENTATION.
10 * Im Konstruktor darf erst nach Aufrufs des Konstruktors
11 * der Oberklasse auf die eigene Instanz zugegriffen werden.
12
13   METHOD constructor.
14     CALL METHOD super->constructor.
15     wrong_value = err.
16   ENDMETHOD.                     "constructor
17
18 ENDCLASS.                    "c_myex2 IMPLEMENTATION
19
20 CLASS cexcept_2 DEFINITION.
21   PUBLIC SECTION.
22
23     METHODS:
24       throw_exception IMPORTING param TYPE I
25                       RAISING c_myex2.
26
27 ENDCLASS.                    "cexcept_2 DEFINITION
28
29 CLASS cexcept_2 IMPLEMENTATION.
30   METHOD throw_exception.
31     WRITE: / param.
32     IF param = 3.
33       RAISE EXCEPTION TYPE c_myex2 EXPORTING err = param.
34     ENDIF.
35   ENDMETHOD.                     "throw_exception
36
37 ENDCLASS.                    "cexcept_1 IMPLEMENTATION
38
39 DATA:
40   rextest TYPE REF TO cexcept_2,
41   exception TYPE REF TO c_myex2,
```

```
42   txt(255).
43
44 START-OF-SELECTION.
45
46   CREATE OBJECT rextest.
47
48   DO 5 TIMES.
49     TRY.
50       CALL METHOD rextest->throw_exception
51         EXPORTING
52           param = sy-index.
53     CATCH c_myex2 INTO exception.
54       WRITE: /.
55       txt = exception->get_text( ).
56       WRITE: / txt.
57       WRITE: / 'Auslösender Parameter-Wert:',
58                 exception->wrong_value.
59       WRITE: /.
60     ENDTRY.
61   ENDDO.
```

Zunächst wird wieder eine eigene Ausnahmeklasse definiert. Im Gegensatz zum ersten Beispiel erhält sie aber ein eigenes Attribut `wrong_value`. Damit beim Erzeugen des Ausnahmeobjekts mit dem Kommando `RAISE EXCEPTION` das Attribut mit einem Wert gefüllt werden kann, muss die Klasse außerdem einen Konstruktor erhalten. Ein Konstruktor ist eine Methode, die automatisch beim Erzeugen eines Objekts für genau dieses Objekt aufgerufen wird. Es dient zur Initialisierung dieses Objekts. In diesem Beispiel erwartet der Konstruktor einen Parameter `err`.

Da die neue Ausnahmeklasse eigene Methoden definiert, muss ein Implementationsteil folgen. Im Konstruktor wird der Parameter `err` entgegengenommen und in das Attribut `wrong_value` kopiert. Zuvor muss jedoch der Konstruktor der Superklasse aufgerufen werden. Das ist eine allgemeine Forderung in ABAP Objects und hat nichts mit den Ausnahmen im Speziellen zu tun. Innerhalb eines Konstruktors darf erst dann auf Instanzattribute und -methoden zugegriffen werden, wenn der Konstruktor der Superklasse aufgerufen wurde.

Auf Definition und Implementierung der Ausnahmeklasse folgt wieder der Programmcode einer Klasse, in der die Ausnahme testweise ausgelöst werden kann. Einziger Unterschied zum vorangegangenen Beispiel ist die Übergabe eines Parameters beim Erzeugen der Ausnahme. Dazu wird der `EXPORTING`-Zusatz des `RAISE EXCEPTION`-Kommandos benutzt.

Um das Attribut der Ausnahme nutzen zu können, muss die `CATCH`-Anweisung die Ausnahme nicht nur erkennen, sondern das konkrete Objekt entgegennehmen. Dazu muss eine passende Objektreferenz angelegt (Zeile 41) und im `INTO`-Zusatz der der `CATCH`-Anweisung benutzt werden (Zeile 53). Innerhalb des

CATCH-Blockes kann dann über die Objektreferenz auf die Attribute und Methoden der Ausnahme zugegriffen werden. In Zeile 55 wird zunächst eine Methode get_text() aufgerufen, die in allen Ausnahmen zur Verfügung steht, da sie in CX_ROOT, der Wurzelklasse aller Ausnahmen definiert ist. Diese Methode liefert einen beschreibenden Text zurück, der sich auf die konkrete Quelle der Ausnahme bezieht. In der Zeile 58 wird dann das neu definierte Attribut benutzt, um den Fehler auslösenden Parameter zu ermitteln.

Das letzte Beispiel demonstriert zwei weitere Aspekte der Ausnahmebehandlung. Im Mittelpunkt steht die Auswirkung der Klassenhierarchie der Ausnahmeklassen. Deren Berücksichtigung ermöglicht es, mehrere Ausnahmen mit einer gemeinsamen CATCH-Anweisung abzufangen. Außerdem bietet sich in diesem Beispiel die Möglichkeit, das Weiterreichen einer Ausnahme und die Wirkung des CLEANUP-Kommandos zu demonstrieren.

```
 1 REPORT yz440090.
 2
 3 CLASS c_myex1 DEFINITION INHERITING FROM cx_static_check.
 4 ENDCLASS.                    "c_myex1 DEFINITION
 5
 6 CLASS c_myex2 DEFINITION INHERITING FROM c_myex1.
 7 ENDCLASS.                    "c_myex2 DEFINITION
 8
 9 CLASS c_myex3 DEFINITION INHERITING FROM cx_static_check.
10 ENDCLASS.                    "c_myex3 DEFINITION
11
12 CLASS cexcept_1 DEFINITION.
13   PUBLIC SECTION.
14     METHODS:
15       throw_exception IMPORTING param TYPE I
16                       RAISING c_myex2 c_myex1 c_myex3 .
17 ENDCLASS.                    "cexcept_1 DEFINITION
18
19 CLASS cexcept_1 IMPLEMENTATION.
20   METHOD throw_exception.
21     WRITE /.
22     WRITE: / param.
23     IF param = 1.
24       RAISE EXCEPTION TYPE c_myex1.
25     ENDIF.
26     IF param = 2.
27       RAISE EXCEPTION TYPE c_myex2.
28     ENDIF.
29     IF param = 3.
30       RAISE EXCEPTION TYPE c_myex3.
31     ENDIF.
32   ENDMETHOD.                    "throw_exception
```

```
ENDCLASS.                    "cexcept_1 IMPLEMENTATION

CLASS cexcept_2 DEFINITION.
  PUBLIC SECTION.
    METHODS:
      test1 IMPORTING param TYPE I
            RAISING   c_myex1 c_myex3,
      test2 IMPORTING param TYPE I
            RAISING   c_myex3.
  PRIVATE SECTION.
    DATA: exref TYPE REF TO cexcept_1,
          exc   TYPE REF TO cx_static_check,
          txt(255).
ENDCLASS.                    "cexcept_2 DEFINITION

CLASS cexcept_2 IMPLEMENTATION.
  METHOD test1.
    CREATE OBJECT exref.
    TRY.
        CALL METHOD exref->throw_exception
          EXPORTING
            param = param.
      CATCH c_myex1 INTO exc.
        txt = exc->get_text( ).
        WRITE: / 'test1_1:', txt.
      CLEANUP.
        WRITE: / 'test1 cleanup'.
    ENDTRY.
  ENDMETHOD.

  METHOD test2.
    CREATE OBJECT exref.
    TRY.
        CALL METHOD exref->throw_exception
          EXPORTING
            param = param.
* wenn cmyex2 nicht vor c_myex1, dann compilerfehler
      CATCH c_myex2 INTO exc.
        txt = exc->get_text( ).
        WRITE: / 'test2_2:', txt.
      CATCH c_myex1 INTO exc.
        txt = exc->get_text( ).
        WRITE: / 'test2_1:', txt.
      CATCH c_myex3 INTO exc.
        WRITE: / 'test2_3: exception occured'.
        RAISE EXCEPTION exc.
    ENDTRY.
```

```
 80   ENDMETHOD.
 81 ENDCLASS.                    "cexcept_2 IMPLEMENTATION
 82
 83 DATA:
 84   rextest TYPE REF TO cexcept_2,
 85   exception TYPE REF TO cx_static_check,
 86   txt(255).
 87
 88 START-OF-SELECTION.
 89   CREATE OBJECT rextest.
 90
 91   DO 3 TIMES.
 92     TRY.
 93         CALL METHOD rextest->test1
 94           EXPORTING
 95             param = sy-index.
 96       CATCH cx_static_check INTO exception.
 97         txt = exception->get_text( ).
 98         WRITE: / 'Rahmenprogramm:', txt.
 99     ENDTRY.
100   ENDDO.
101
102   WRITE: / '========================================='.
103   DO 3 TIMES.
104     TRY.
105         CALL METHOD rextest->test2
106           EXPORTING
107             param = sy-index.
108       CATCH cx_static_check INTO exception.
109         txt = exception->get_text( ).
110         WRITE: / 'Rahmenprogramm:', txt.
111     ENDTRY.
112   ENDDO.
```

Zunächst werden drei Ausnahmeklassen angelegt, wobei die ersten beiden eine kleine Hierarchie bilden. Die einzige Methode der Klasse `cexcept_1` löst in Abhängigkeit vom Eingabeparameter jeweils eine dieser Ausnahmen aus. Damit das Weiterreichen von Ausnahmen demonstriert werden kann, erfolgt der Aufruf der `throw_exception`-Methode von `cexcept_1` über eine zweite Klasse. Diese stellt zwei Methoden bereit, in denen die von `cexcept_1` erzeugten Ausnahmen unterschiedlich behandelt werden.

Die Methode `test1` behandelt alle Ausnahmen vom Typ `c_myex1`. Da `c_myex2` von dieser Ausnahme abgeleitet wurde, besteht Zuweisungskompatibilität zu diesem Typ. Somit wird auch die auf dieser Klasse beruhende Ausnahmen durch den entsprechenden `CATCH`-Block behandelt (Zeile 55 – 57). Die Ausnahme `c_myex3` wird hingegen nicht behandelt. Allerdings wird sie in der `RAISE`-

Option der Methodendefinition von `test1` aufgeführt. Dies bedeutet, dass diese Ausnahme automatisch an den Aufrufer der Methode `test1` weitergeleitet wird, falls sie innerhalb von `test1` auftreten sollte. Allerdings wird für alle Ausnahmen, die nicht innerhalb des `TRY`-Blockes behandelt werden, der `CLEANUP`-Block aktiv. Die Reihenfolge der Ausgaben beim Aufruf von `test1` (Zeilen 91 – 100) verdeutlicht dies.

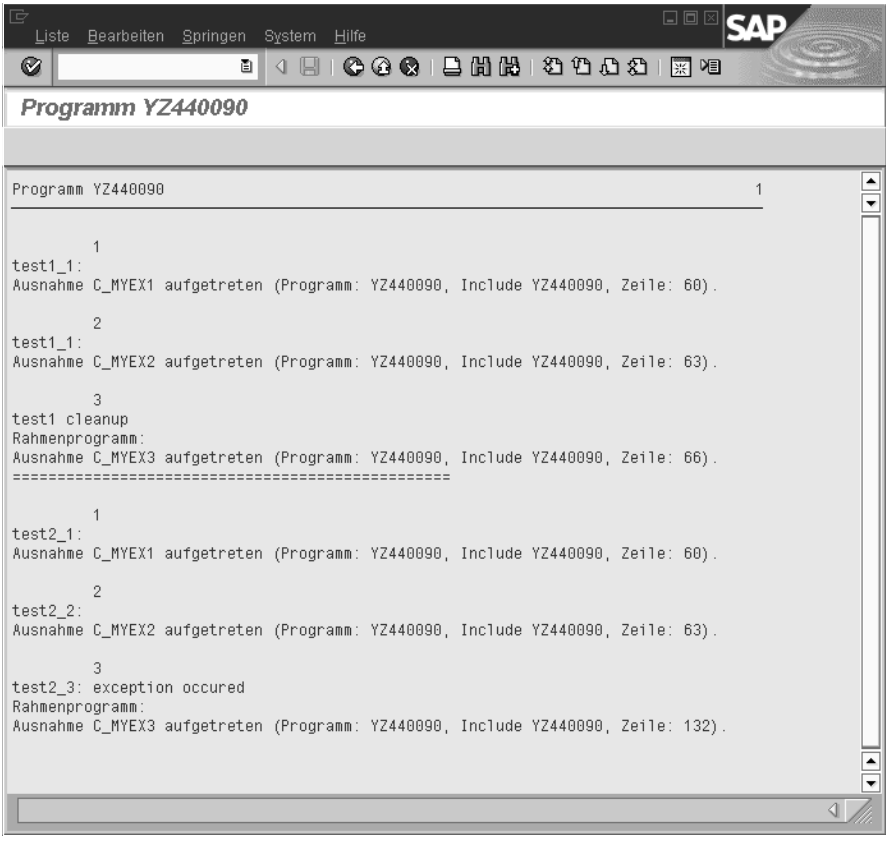

Abbildung 4.2 **© SAP AG**
Bildschirmausgabe des ReportsYZ440090

Die zweite Testmethode behandelt unter anderem die Ausnahme `c_myex2`. Bei der Programmierung der `CATCH`-Anweisungen muss die Auswirkung der Klassenhierarchie der Ausnahmen beachtet werden. So muss die `CATCH`-Anweisung für `c_myex2` unbedingt vor der für `c_myex1` stehen. Beim Auftreten einer Ausnahme werden die `CATCH`-Anweisungen sequenziell abgearbeitet. Sobald eine `CATCH`-Anweisung gefunden wird, die zum Typ der Ausnahme passt, werden die Anweisungen des `CATCH`-Blockes ausgeführt. Sollte die Behandlung von

c_myex1 vor c_myex2 stehen, würde wegen der Zuweisungskompatibilität von c_myex2 zu c_myex1 immer die Behandlungsroutine von c_myex1 ausgeführt werden. Die Reihenfolge der CATCH-Anweisungen wird bereits vom Compiler überprüft, so dass eine falsche Reihenfolge keine echte Fehlerquelle darstellt.

Am Beispiel der Ausnahme c_myex3 wird in der Methode test2 dann auch noch das explizite Weiterreichen einer Ausnahme demonstriert (Zeile 78).

Die Ausgaben des Reports beruhen auf der get_text-Methode der Ausnahmen, so dass leicht verfolgt werden kann, an welcher Stelle welche Ausnahme wirklich behandelt wird. Abbildung 4.2 zeigt die vom Report generierte Ausgabe.

Aufgaben:

1. Verändern Sie die Klasse calc1 so, dass die Eingabeparameter des Konstruktors optional sind. Erzeugen Sie zusätzliche Methoden, um die beiden Operanden unabhängig vom Konstruktor setzen zu können und testen Sie die neuen Methoden in einem eigenen Programm.
2. Erstellen Sie eine neue Klasse, in der eine Division durch Null eine Ausnahme auslöst.
3. Verändern Sie das Programm yz442060 so, dass zwei Objekte zum Auslösen von Ausnahmen bereitstehen und weisen Sie den cstring-Objekten abwechselnd eines der beiden Objekte als gültigen Sender für das modify-Ereignis zu. Testen Sie nun die Behandlung der Ereignisse.

4.3 Dynamische Dokumente

Beginnend mit der Version 4 wurden im R/3-System so genannte Custom Controls zur Verfügung gestellt. Diese Controls sind relativ komplexe Objekte, die im herkömmlichen SAPGUI bereitgestellt werden. Verfügbar sind beispielsweise Controls zur Darstellung von Bäumen, Editoren und HTML-Viewer. In [Riekert][1] finden sie ein ausführliches Beispiel zur Verwendung derartiger Controls.

Beginnend mit der Version 6 werden weitere Controls bereitgestellt, die aber auf einem völlig anderen Funktionsprinzip beruhen. Es handelt sich um die so genannten *Dynamischen Dokumente*. Ein dynamisches Dokument ist ein Container, der beliebige andere visuelle Elemente aufnehmen kann, z.B. Eingabefelder, Grafiken usw. Entscheidend ist, dass diese einen elementaren Charakter haben. Darüber hinaus weisen sie zwei Eigenschaften auf, die eine völlig neue Form der Oberflächenprogrammierung ermöglichen:

1. Riekert, Rainer: ABAP-Programmierung: Fortgeschrittene Programmiertechniken für ABAP, München, Addison Wesley 2001

❶ Die Elemente können zur Laufzeit erstellt und in den Container eingefügt werden.

❷ Die Elemente können durch das R/3-System automatisch in Elemente umgewandelt werden, die dem HTML-Standard entsprechen.

Alle Bestandteile eines dynamischen Dokuments, sowohl die Container als auch die elementaren Eingabe- und Bedienelemente, sind Objekte verschiedener Klassen. Ausgehend von der Wurzel, dem Container für das eigentliche Dokument, werden die Teilelemente in ein übergeordnetes Element eingefügt und von diesem verwaltet. Die wichtigsten verfügbaren Klassen für dynamische Dokumente finden Sie in Tabelle 4.5.

Klasse	Aufgabe
CL_DD_DOCUMENT	Container für gesamtes Dokument
CL_DD_FORM_AREA	Bereich eines Formulars
CL_DD_INPUT_ELEMENT	Eingabeelement
CL_DD_BUTTON_ELEMENT	Schaltfläche
CL_DD_EVENT_HANDLER	Event-Handler
CL_DD_SELECT_ELEMENT	Auswahl-Element
CL_DD_LINK_ELEMENT	Link-Element

Tabelle 4.3
Wichtige Klassen für dynamische Dokumente

Ein dynamisches Dokument wird in einem Dynpro angezeigt. Es wird dort in ein Custom Control eingefügt.

Das nachfolgende Beispiel demonstriert die prinzipielle Verfahrensweise beim Aufbau dynamischer Dokumente. Zwecks besserer Übersichtlichkeit ist es in mehrere Abschnitte unterteilt. Beim Programm handelt es sich wieder um einen einfachen Rechner, dessen Funktionalität dem des Beispiels aus dem Abschnitt über Dialoganwendungen entspricht. Bild 4.3 zeigt die Anwendung zur Laufzeit.

Um das Beispiel ausführen zu können, müssen Sie zunächst den Report `YZ440100` anlegen. In diesem Report muss später ein Dynpro mit der Nummer 100 erstellt werden. In diesem Dynpro wird lediglich ein Custom Control eingefügt. Es sollte etwa 70 Zeichen breit und 10 Zeilen hoch sein. Das Control erhält den Namen `CC1`. Bei Bedarf können Sie die Abmessungen später noch verändern. Außerdem muss ein Status `STAT1` erzeugt werden. In diesem Status sollte die Funktionstaste F3 mit dem Exit-Funktionscode `BACK` belegt werden. Andere Funktionen sind nicht notwendig.

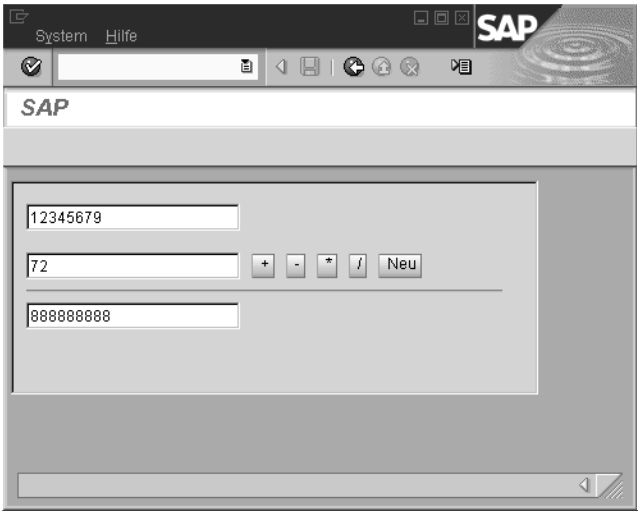

Abbildung 4.3 **© SAP AG**
Ansicht einer Demo-Anwendung für dynamische Dokumente

In jeder Anwendung, die dynamische Dokumente benutzen möchte, ist der Type-Pool SDYDO einzubinden. Er enthält wichtige Deklarationen, beispielsweise die Datentypen zur Typisierung der diversen Parameter.

```
REPORT yz440100.
TYPE-POOLS:
  sdydo.
```

Einige der Bestandteile des dynamischen Dokuments müssen als globale Variablen vorliegen. Im nachfolgenden Abschnitt der Anwendung werden zunächst die entsprechenden Objektreferenzen deklariert. Die Beschreibung der diversen Klassen erfolgt später.

```
DATA:
  gr_screen TYPE REF TO cl_dd_document,
  gc_initialize TYPE c VALUE 'X',
  gr_in1 TYPE REF TO cl_dd_input_element,
  gr_in2 TYPE REF TO cl_dd_input_element,
  gr_op TYPE REF TO cl_dd_input_element,
  gr_result TYPE REF TO cl_dd_input_element,
  gr_formarea TYPE REF TO cl_dd_form_area.
```

Ein dynamisches Dokument verwendet nicht mehr das herkömmliche Dynpro-Programmiermodell. Vielmehr werden durch die Bestandteile eines dynamischen Dokuments diverse Ereignisse ausgelöst, die von so genannten Event-Handlern verarbeitet werden müssen. Die Auswertung der Benutzer-Aktion und die Verknüpfung mit der eigentlichen Anwendungslogik liegt nun nicht

mehr im PAI-Teil eines Dynpros, sondern in einem oder mehreren Event-Handlern. Beim Event-Handler handelt es sich um eine Klasse, die zunächst zu definieren ist.

```
CLASS calculator DEFINITION.
  PUBLIC SECTION.
    METHODS:
      handle_button_add FOR EVENT clicked OF
      cl_dd_button_element

IMPORTING sender,
      handle_button_sub FOR EVENT clicked OF
      cl_dd_button_element

IMPORTING sender,
      handle_buttons_dot FOR EVENT clicked OF
      cl_dd_button_element

IMPORTING sender,
      handle_button_new FOR EVENT clicked OF
      cl_dd_button_element

IMPORTING sender.
  PRIVATE SECTION.
    DATA: i1 TYPE i,
          i2 TYPE i,
          res TYPE sdydo_value.
    METHODS:
      refresh_screen,
      get_values,
      set_result.

ENDCLASS.                    "calculator DEFINITION
```

Nach der Definition muss natürlich eine Objektreferenz für den Event-Handler erstellt und eine entsprechende Instanz angelegt werden.

```
DATA: my_calculator TYPE REF TO calculator.

CREATE OBJECT my_calculator.
```

Da das dynamische Dokument innerhalb eines Dynpros abläuft, kann nun ein Sprung zu diesem Dynpro erfolgen. Bedingt durch die Struktur eines Reports finden Sie die Module und Implenetatonsteile der beteiligten Klassen weiter unten.

```
CALL SCREEN 100.
```

Im PBO-Modul des Dynpros erfolgt die Initialisierung des dynamischen Dokuments. Da diese nur beim ersten Aufruf des Dynpros erfolgen darf, regelt das

Flag gc_initialize die Ausführung der Initialisierungsroutine. Innerhalb des PBO-Moduls wird zunächst ein Objekt für das dynamische Dokument erzeugt. Das Feld gr_screen zeigt auf das Objekt. Der eigentliche Aufbau des dynamischen Dokuments wurde in ein Unterprogramm create_mask verlagert. In diesem Unterprogramm werden diverse Unterobjekte in gr_screen eingefügt. Das Objekt gr_screen ist nur ein Container, der andere visuelle Elemente enthält. Durch den Aufruf der Methode merge_document wird das Dokument endgültig erstellt und mit display_document im Custom Control CC1 angezeigt.

```
MODULE status_0100 OUTPUT.
  IF gc_initialize = 'X'.
    SET PF-STATUS 'STAT1'.

    CREATE OBJECT gr_screen
      EXPORTING
        background_color = gr_screen->col_background_level2.

    PERFORM create_mask.

    CALL METHOD gr_screen->merge_document.

    CALL METHOD gr_screen->display_document
      EXPORTING
        container          = 'CC1'
      EXCEPTIONS
        html_display_error = 1.

    CLEAR gc_initialize.
  ENDIF.
ENDMODULE.                             " STATUS_0100  OUTPUT
```

Das PAI-Modul ist bei dieser Form der Anwendung lediglich für die Auswertung der Funktionstaste F3 zuständig. Sie dient dazu, das Programm zu beenden. Alle anderen Aktivitäten werden durch den bereits erwähnten Event-Handler ausgeführt.

```
MODULE user_command_0100 INPUT.
  CASE sy-ucomm(4).
    WHEN 'BACK'.
      LEAVE PROGRAM.
  ENDCASE.
ENDMODULE.                             " USER_COMMAND_0100
INPUT
```

Der Aufbau des dynamischen Dokuments ist recht aufwändig, da er durch reine Programmierung erfolgt. Im Moment existiert noch kein Werkzeug, mit dem ein dynamisches Dokument auf visueller Basis erstellt werden könnte. Zu Beginn des Unterprogramms create_mask erfolgt die Deklaration einiger lokaler

Objektreferenzen. In diesem speziellen Fall sind es Felder, die auf die 5 Drucktasten verweisen.

```
FORM create_mask.
DATA:
  r_bu_add TYPE REF TO cl_dd_button_element,
  r_bu_sub TYPE REF TO cl_dd_button_element,
  r_bu_mul TYPE REF TO cl_dd_button_element,
  r_bu_div TYPE REF TO cl_dd_button_element,
  r_bu_new TYPE REF TO cl_dd_button_element.
```

Nun wird das dynamische Dokument durch Aufruf der Methode `add_form` dazu veranlasst, ein Formular zu erzeugen. Die Referenz auf dieses Formular wird im globalen Feld `gr_formarea` zurückgegeben.

```
  CALL METHOD gr_screen->add_form
    IMPORTING
      formarea = gr_formarea.
```

Ebenso wie `gr_screen` ein Formular erzeugen kann, verfügt ein Formular über Methoden, mit denen es die für ein Formular benötigten Elemente erstellen kann. Für das recht einfache Formular dieses Beispiels handelt es sich lediglich um Eingabefelder und Drucktasten, die über die Methoden `add_input_element` und `add_button` angelegt werden. Auch diese Methoden liefern die Referenz auf das angelegte Element als Export-Parameter zurück.

```
  CALL METHOD gr_formarea->add_input_element
    EXPORTING
      value         = ''
      name          = 'I1'
      size          = 30
      maxlength     = 30
      tooltip       = 'Input 1'
    IMPORTING
      input_element = gr_in1.
```

Erwähnenswert sind noch die Methoden `new_line` und `line_with_layout`. Die erste Methode fügt einfach einen Zeilenvorschub ein. Die zweite beginnt bzw. beendet einen Anweisungsblock. Alle Elemente in diesem Block erscheinen später in einer Zeile. Damit innerhalb der Zeile eine Ausrichtung möglich ist, können Sie mit der Methode `add_gap` eine definierte Anzahl von Leerzeichen einfügen.

```
  gr_formarea->new_line( ).
  gr_formarea->line_with_layout( start = 'X' ).

  CALL METHOD gr_formarea->add_input_element
    EXPORTING
      value         = ''
```

```
    name          = 'I2'
    size          = 30
    maxlength     = 30
    tooltip       = 'Input 2'
  IMPORTING
    input_element = gr_in2.

gr_formarea->add_gap( width = 3 ).

CALL METHOD gr_formarea->add_button
  EXPORTING
    label   = ' + '
    tooltip = 'Addition'
    name    = 'BADD'
  IMPORTING
    button  = r_bu_add.

gr_formarea->add_gap( width = 3 ).

CALL METHOD gr_formarea->add_button
  EXPORTING
    label   = ' - '
    tooltip = 'Subtraktion'
    name    = 'BSUB'
  IMPORTING
    button  = r_bu_sub.

gr_formarea->add_gap( width = 3 ).

CALL METHOD gr_formarea->add_button
  EXPORTING
    label   = ' * '
    tooltip = 'Multiplikation'
    name    = 'BMUL'
  IMPORTING
    button  = r_bu_mul.

gr_formarea->add_gap( width = 3 ).

CALL METHOD gr_formarea->add_button
  EXPORTING
    label   = ' / '
    tooltip = 'Division'
    name    = 'BDIV'
  IMPORTING
    button  = r_bu_div.
```

```
  gr_formarea->add_gap( width = 3 ).

  CALL METHOD gr_formarea->add_button
    EXPORTING
      label   = ' Neu '
      tooltip = 'Neu'
      name    = 'BNEW'
    IMPORTING
      button  = r_bu_new.

  gr_formarea->line_with_layout( end = 'X' ).

  gr_formarea->underline( ).

  CALL METHOD gr_formarea->add_input_element
    EXPORTING
      value         = ''
      name          = 'OUT'
      size          = 30
      maxlength     = 30
      tooltip       = 'Result'
    IMPORTING
      input_element = gr_result.
```

Zum Abschluss müssen den Drucktasten noch die Event-Handler zugewiesen werden. Zwecks späterer Demonstration diverser Eigenschaften der Event-Handler besitzen die beiden Drucktasten für Multiplikation und Division einen gemeinsamen Event-Handler, die anderen drei Tasten verfügen jeweils über eine separate Event-Behandlung.

```
  SET HANDLER my_calculator->handle_button_add  FOR r_bu_add.
  SET HANDLER my_calculator->handle_button_sub  FOR r_bu_sub.
  SET HANDLER my_calculator->handle_buttons_dot FOR r_bu_mul.
  SET HANDLER my_calculator->handle_buttons_dot FOR r_bu_div.
  SET HANDLER my_calculator->handle_button_new  FOR r_bu_new.
ENDFORM.                    "create_mask
```

Die eigentliche Anwendungslogik befindet sich in der `calculator`-Klasse, die auch die Methoden zum Event-Handling enthält. Die Funktionalität der Event-Handler ist relativ ähnlich. Meist wird zunächst mit der Methode `get_values` der Inhalt der beiden Eingabefelder in die Felder `i1` und `i2` gelesen. Anschließend wird die Berechnung durchgeführt und das Ergebnis mit `set_result` in das Ergebnisfeld geschrieben. Zum Abschluss wird ein Neuaufbau des Bildschirms notwendig, damit die geänderten Werte auch angezeigt werden.

```
CLASS calculator IMPLEMENTATION.

  METHOD handle_button_add.
```

```
    get_values( ).
    res = i1 + i2.
    set_result( ).
    refresh_screen( ).
  ENDMETHOD.                    "handle_button_add

  METHOD handle_button_sub.
    get_values( ).
    res = i1 - i2.
    set_result( ).
    refresh_screen( ).
  ENDMETHOD.                    "handle_button_sub
```

Geringfügig anders ist die Methode für die Behandlung der Multiplikations- und Divisionstaste aufgebaut. Dem Event-Handler wird vom Laufzeitsystem stets ein Parameter übergeben, über den das rufende Objekt ermittelt werden kann. In diesem Fall wird anhand des Namens der betätigten Taste die jeweils korrekte Rechenoperation ausgewählt.

```
  METHOD handle_buttons_dot.
    get_values( ).
    IF sender->name = 'BMUL'.
      res = i1 * i2.
    ELSE.
      IF sender->name = 'BDIV'.
        IF i2 <> 0.
          res = i1 / i2.
        ELSE.
          res = 'ERROR'.
        ENDIF.
      ENDIF.
    ENDIF.

    set_result( ).
    refresh_screen( ).
  ENDMETHOD.                    "handle_buttons_dot

  METHOD handle_button_new.
    gr_in1->set_value( ' ' ).
    gr_in2->set_value( ' ' ).
    gr_result->set_value( ' ' ).
    refresh_screen( ).
  ENDMETHOD.                    "handle_button_new
```

Die Methode `refresh_screen` baut den Bildschirm, genauer den Bereich des dynamischen Dokuments, neu auf. Sie benutzt dazu die bereits erwähnten Methoden `merge_document` und `display_document`. Die letztgenannte wird allerdings mit anderen Parametern aufgerufen. Diese Methoden sorgen dafür,

dass lediglich die Aktualisierung des Inhalts des bereits existierenden Dokuments erfolgt.

```
METHOD refresh_screen.
  CALL METHOD gr_screen->merge_document.
  CALL METHOD gr_screen->display_document
    EXPORTING
      reuse_control      = 'X'
      reuse_registration = 'X'.
ENDMETHOD.                    "refresh_screen
```

Etwas ungewöhnlich erscheint die `get_values`-Methode. Obwohl in die beiden Eingabefelder ein Wert eingetragen wurde, der auch korrekt über das Attribut `value` gelesen werden kann, muss dieser Wert nochmals explizit mit `set_value` in das Eingabeelement geschrieben werden, damit das Eingabefeld nach dem Neuaufbau der Maske mit eben diesem Wert vorbelegt ist.

```
METHOD get_values.
  i1 = gr_in1->value.
  i2 = gr_in2->value.
  gr_in1->set_value( gr_in1->value ).
  gr_in2->set_value( gr_in2->value ).
ENDMETHOD.                    "get_values

METHOD set_result.
  SHIFT res LEFT DELETING LEADING ' ' IN CHARACTER MODE.
  gr_result->set_value( res ).
ENDMETHOD.                    "set_result

ENDCLASS.                   "calculator IMPLEMENTATION
```

4.4 Der Class Builder

Klassen können nicht nur mittels direkter Programmierung innerhalb eines Programms erstellt werden. Diese Variante ist lediglich von eingeschränktem Nutzen, da die so angelegten Klassen nur lokal innerhalb dieses Programms verfügbar sind. Wesentlich effektiver ist es, Klassen ähnlich wie Funktionsbausteine global verfügbar zu machen. Dies erfolgt mittels des so genannten *Class Builder*, einem dialogorientierten Werkzeug zur Bearbeitung globaler Klassen. Der Class Builder wird über den Transaktionscode `SE24` oder vom SAP-Menü aus über die Menüfunktion ENTWICKLUNG | CLASS BUILDER aufgerufen. Bild 4.4 zeigt das Startbild dieser Transaktion.

Nach Eingabe des Namens einer neu anzulegenden Klasse und betätigen der ANLEGEN-Schaltfläche erscheint ein Popup (siehe Abbildung 4.5), in dem ausgewählt werden muss, ob es sich beim anzulegenden Element um eine echte Klasse oder ein Interface handelt.

Abbildung 4.4 **© SAP AG**
Startbild des Class Builder

In einem weiteren Popup (Abbildung 4.6) geben Sie zusätzliche Informationen über das anzulegende Objekt an. Über den Klassentyp legen Sie fest, ob es sich um eine gewöhnliche Klasse, eine Ausnahmeklasse oder eine Klasse mit persistenten Attributen handelt. Über das Symbol neben dem Klassennahmen definieren Sie eine eventuelle Vererbung. Außerdem können Sie wählen, ob von dieser Klasse weitere abgeleitet werden dürfen (Flag: FINAL) und ob die Klasse ausprogrammiert oder nur modelliert werden soll.

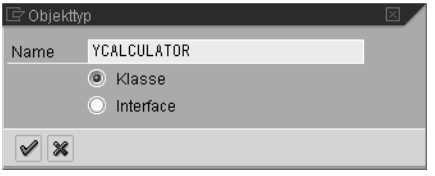

Abbildung 4.5 **© SAP AG**
Art des anzulegenden Objekts bestimmen

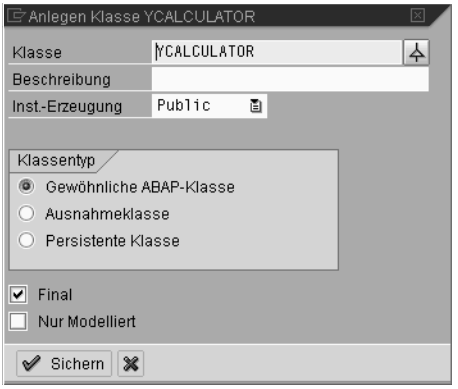

Abbildung 4.6 **© SAP AG**
Eigenschaften einer Klasse pflegen

Nach Bestätigen der Angaben in diesem Popup erscheint die eigentliche Pflegeoberfläche des Class Builder. Sie besteht aus einem Tab Strip Control mit mehreren Registerkarten. Der zur Verfügung stehende Platz ist jeweils mit einer umfangreichen Tabelle gefüllt (siehe Abbildung 4.7).

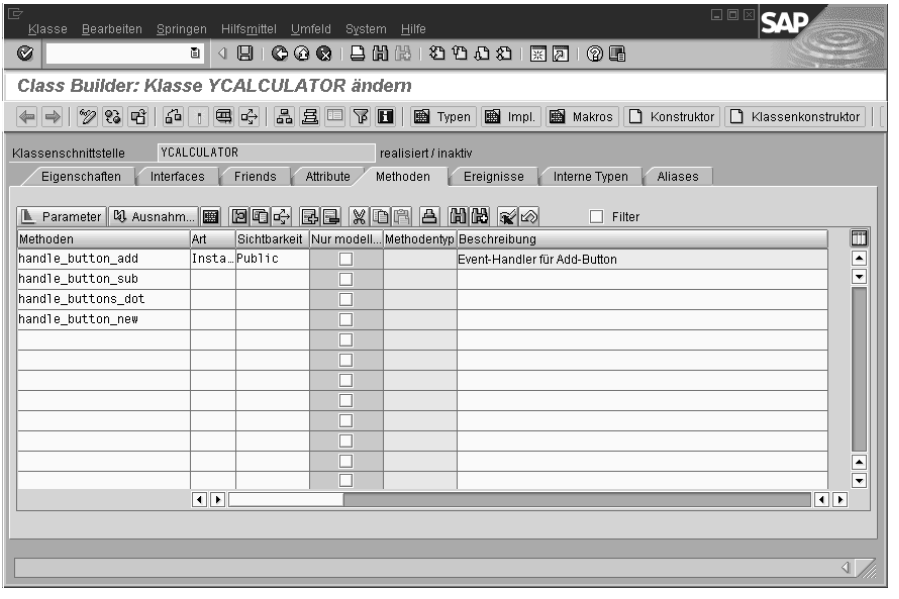

Abbildung 4.7 **© SAP AG**
Der Class Builder

Beim Start des Class Builder ist üblicherweise die Registerkarte für die Methoden aktiv. Sie können hier neue Methoden anlegen, in dem Sie einen neuen Eintrag in die Tabelle einfügen. In den weiteren Spalten legen Sie fest, ob es sich um Instanz- oder Klassenmethoden handelt, definieren die Sichtbarkeit und pflegen ggf. eine Beschreibung. Durch einen Doppelklick auf einen Methodennamen öffnen Sie einen Editor zur Pflege des Quelltextes.

Um die Parameter einer Methode zu definieren, platzieren Sie den Cursor auf dem Namen der entsprechenden Methode und betätigen die Drucktaste Parameter, die Sie direkt über der Überschriftenzeile der Tabelle finden. Abbildung 4.8 zeigt die entsprechende Maske.

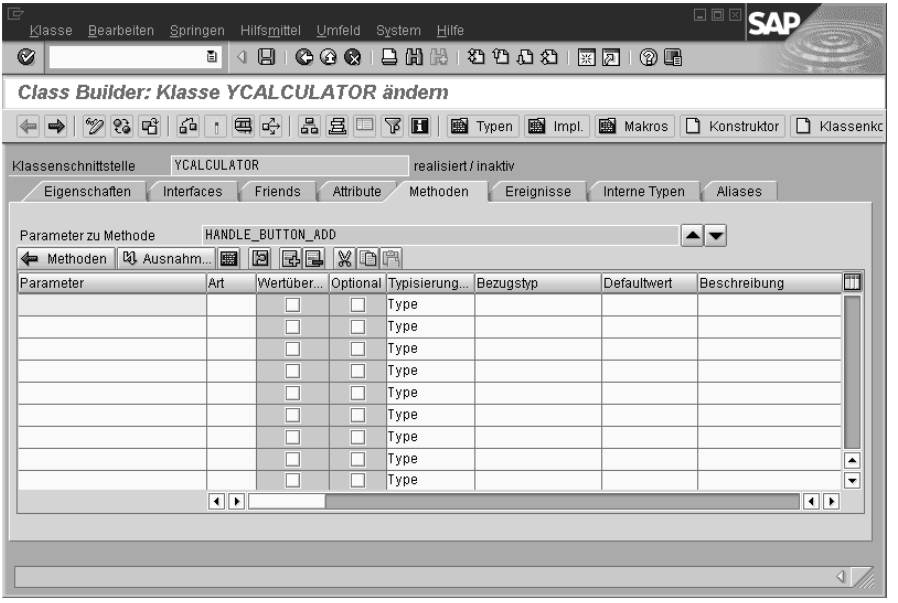

Abbildung 4.8 **© SAP AG**
Parameter einer Methode pflegen

Der Aufbau der Tabelle sowie deren Funktionalität erinnert stark an die Pflege der Parameter eines Funktionsbausteins.

Die Rückkehr zur Übersichtstabelle für die Methoden erfolgt mit der Schaltfläche METHODEN.

Durch Auswahl der anderen Registerkarten gelangen Sie zur Pflege der restlichen Elemente einer Klasse. Die Namen der Registerkarten, die Überschriften der Tabellenspalten sowie die teilweise vorhandenen Eingabehilfen bieten dem ABAP OO-Kundigen ausreichend Informationen zur Bedienung dieses Werkzeugs.

5 Pflege der Data-Dictionary-Elemente

Neben der Programmierarbeit im engeren Sinne, also dem Aufschreiben von Quelltext, erfordert die Erstellung einer ABAP-Anwendung sehr oft das Erzeugen von Elementen im Data Dictionary. Spezielle Teilaufgaben, so z.B. die Realisierung von Eingabehilfen oder Pflegebausteinen, können ohne Programmierarbeit nur durch Pflegen von Dictionary-Elementen realisiert werden. Dieses Kapitel beschreibt die Dictionary-Elemente mit ihren speziellen Eigenschaften sowie die Werkzeuge zur Pflege der Objekte. Die in diesem Kapitel vorgestellten Beispiele bzw. erzeugten Objekte bilden die Grundlage für andere Beispiele in diesem Buch, insbesondere die umfassende Demoanwendung in Kapitel 7. Es ist daher empfehlenswert, alle Beispiele und Übungen auszuführen. Die Struktur der einzelnen Abschnitte dieses Kapitels entspricht weitgehend dem Aufbau vorangegangener Kapitel. Nach einer praktischen Demonstration folgt ein umfangreicher Teil mit theoretischen Erläuterungen.

Alle nachfolgend beschriebenen Werkzeuge sind vom Hauptmenü der ABAP Workbench über die Menüfunktion ENTWICKLUNG | DICTIONARY oder den Transaktionscode SE11 zugänglich. Es erscheint zunächst das Einstiegsbild der Dictionary-Pflege (Abbildung 5.1).

In diesem Einstiegsbild wird per Auswahlfeld das anzulegende Element bestimmt und im zugehörigen Eingabefeld der Name des zu bearbeitenden Objekts eingetragen. Drei Drucktasten im Dynpro verzweigen zu den typischen Bearbeitungsarten Anlegen, Ändern oder Anzeigen. Objekte unterschiedlichen Typs können durchaus übereinstimmende Namen haben. Dies gilt allerdings nicht für alle Typen. Views, Strukturen, Tabellen und Datenelemente besitzen einen gemeinsamen Namensraum. Objekte einer dieser vier Gruppen müssen einen typübergreifenden eindeutigen Namen besitzen. Diese Eindeutigkeit ist Voraussetzung dafür, diese Elemente auch als Typbezeichner in Datendeklarationen benutzen zu können.

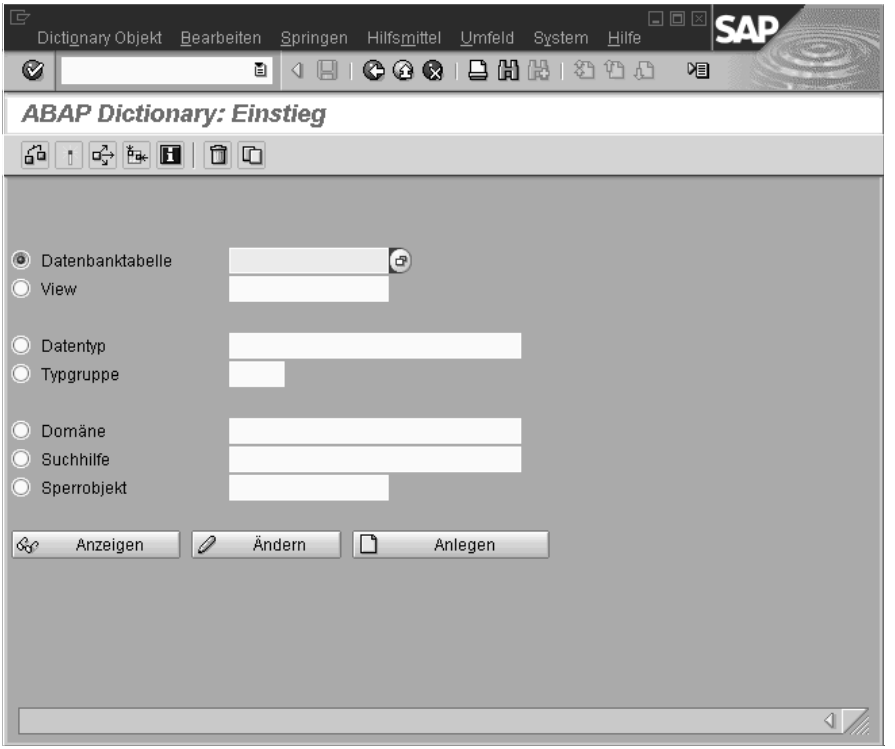

Abbildung 5.1 **© SAP AG**
Einstiegsbild zur Dictionary-Pflege

Die in diesem Kapitel erzeugten Elemente tragen das Zeichen Z in ihrem Namen. Dieses Zeichen können Sie bei Bedarf ersetzen, um mehreren Programmierern an einem System die parallele Arbeit zu ermöglichen.

5.1 Domänen, Datenelemente, Tabellen und Strukturen

Die prinzipiellen Zusammenhänge zwischen Domänen, Datenelementen und Tabellen wurden bereits in Kapitel 2 beschrieben. An dieser Stelle soll auf eine Wiederholung dieser Informationen verzichtet werden. Die in diesem Abschnitt beschriebenen Dictionary-Elemente dienen zum Aufbau zweier sehr einfacher Tabellen. In diesen Tabellen sollen Kennbuchstaben für verschiedene Wirtschaftsbranchen und die sprachabhängigen Langtexte zu diesen Kennbuchstaben abgelegt werden.

5.1.1 Domänen

Die Grundlage für die Definition von Tabellenfeldern sind Domänen. Sie stellen die direkte Verbindung zur Datenbank dar. Sie enthalten die technischen Eigenschaften wie den Datentyp oder die Länge. Darüber hinaus kann durch zusätzliche Angaben der durch den Datentyp und die Feldlänge vorgegebene Wertevorrat weiter eingeschränkt werden. Wegen der automatischen Wertprüfungen in Verbindung mit Festwerten oder Wertetabellen und den damit verbundenen Fremdschlüsselbeziehungen haben Domänen ihren Charakter als rein technisches Beschreibungselement zum Teil verloren. Indirekt tragen auch viele Domänen anwendungsspezifische Informationen. Es ist daher üblich und zum Teil sogar notwendig, Domänen nicht nur nach rein technischen Gesichtspunkten, sondern auch im Hinblick auf ihre spätere Verwendung in Anwendungen zu entwerfen. Die Gründe dafür werden später, nach Erzeugen der beiden oben erwähnten Tabellen, ersichtlich.

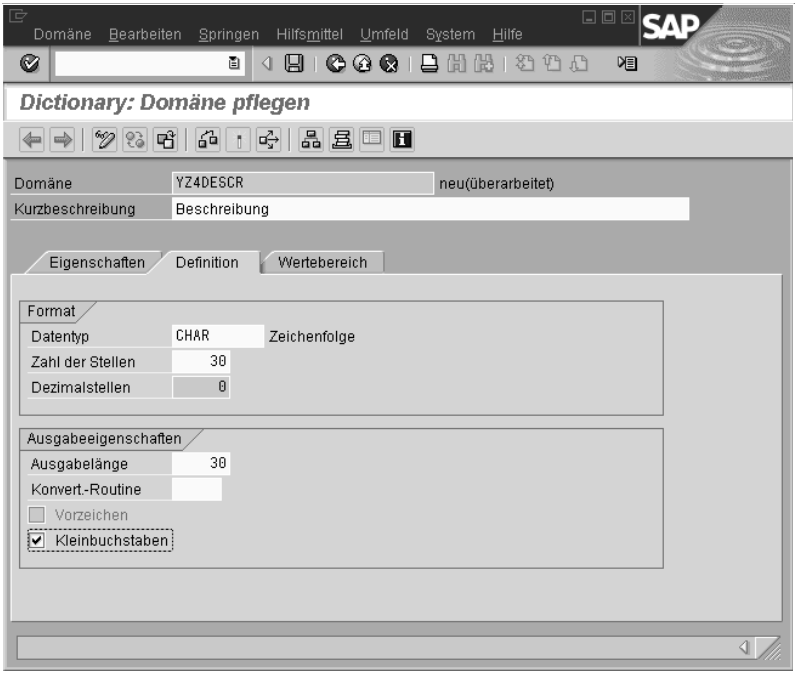

Abbildung 5.2 **© SAP AG**
Pflegebild für eine Domäne

Als erste Domäne wird YZ4DESCR erzeugt. Das auf ihr beruhende Tabellenfeld soll später eine Bezeichnung, also einen erläuternden Text, aufnehmen. Um eine Domäne anzulegen, ist im Eingabefeld des Dictionary-Pflegewerkzeugs der Name der Domäne einzutragen, das Auswahlfeld DOMÄNE zu markieren und

die Schaltfläche ANLEGEN zu betätigen. Es erscheint das in Abbildung 5.2 dargestellte Dynpro, in dem in einem einzelnen Eingabefeld für die Kurzbeschreibung auf drei Registerkarten eines Tab Strips alle wichtigen Angaben für eine Domäne erfasst werden können.

Wie für fast alle anderen Objekte der Entwicklungsumgebung muss auch für Domänen zunächst eine Kurzbeschreibung eingetragen werden. Sie dient nur zur Information, der Inhalt ist daher unkritisch.

Nach dem Start der Domänenpflege ist die Registerkarte DEFINITION aktiv. Auf dieser Karte werden die Angaben zum Daten- und Ausgabeformat erfasst. Von besonderer Bedeutung sind die beiden Felder DATENTYP und ZAHL DER STELLEN. Die neu anzulegende Domäne erhält, wie aus Abbildung 5.2 ersichtlich, den Datentyp `CHAR` und die Länge 30. Die anderen Felder können leer bleiben.

Um die automatische Umwandlung des Feldinhaltes in Großbuchstaben zu unterdrücken, wird das Ankreuzfeld KLEINBUCHSTABEN markiert. Die Eingaben werden gesichert. Dabei wird das Feld AUSGABELÄNGE automatisch ergänzt. Bei anderen Datentypen werden unter Umständen weitere Felder automatisch belegt. Für die Domäne `YZ4DESCR` sind keine weiteren Angaben erforderlich. Der Bereich EIGENSCHAFTEN enthält lediglich Ausgabefelder, die z. B. über den Zeitpunkt der letzten Änderung informieren. Die Registerkarte WERTEBEREICH ermöglicht es, den zulässigen Wertebereich für Tabellenfelder, die auf dieser Domäne beruhen, einzuschränken. Dies wird bei der nächsten anzulegenden Domäne demonstriert. Zunächst muss jedoch die Erstellung der Domäne `YZ4DESCR` abgeschlossen werden.

Eine neue Domäne bzw. Änderungen einer vorhandenen werden erst nach der Aktivierung der Domäne wirksam. Diese Aktivierung erfolgt durch Wahl der entsprechenden Menüfunktion oder durch eine Drucktaste. Falls eine Domäne, die bereits in Datenfeldern und Tabellen benutzt wird, nach Änderungen nochmals aktiviert werden muss, werden auch alle abhängigen Objekte (Tabellen, Datenelemente etc.) aktiviert, da die neuen Eigenschaften dort ebenfalls wirksam werden müssen. Die Aktivierung der abhängigen Objekte kann extrem lange dauern, falls es viele abhängige Elemente gibt. Bei der Entwicklung einer neuen Anwendung sollte die Datenstruktur daher von Vornherein feststehen, um die Zahl der nachträglichen Änderungen und damit der zeitaufwändigen Nachgenerierungen möglichst gering zu halten. Es gibt allerdings Situationen, in denen eine schrittweise Bearbeitung von Domänen mit mehrfacher Aktivierung unumgänglich ist.

Mit der nun anzulegenden Domäne `YZ4BRA` soll die Definition von Domänenfestwerten erläutert werden. Diese Festwerte werden bei Eingaben in Dynpros geprüft, abweichende Eingaben sind nicht möglich. Zur Erleichterung der Eingabe erzeugt das System für Domäne mit Wertprüfung automatisch eine Eingabehilfe, die den verfügbaren Wertevorrat in einer Auswahlliste anzeigt. Legen Sie die neue Domäne mit dem Datentyp `CHAR` und der Länge 1 an. Das Flag für die Groß-/Kleinschreibung wird diesmal nicht aktiviert, so dass nur Großbuch-

staben eingetragen werden können. Die Angaben werden auf dieselbe Weise wie für die erste Domäne erfasst und abgespeichert. Anschließend wechseln Sie zur Registerkarte Wertebereich. Abbildung 5.3 zeigt diesen Teil des Dynpros. Es besteht im Wesentlichen aus zwei Tabellen.

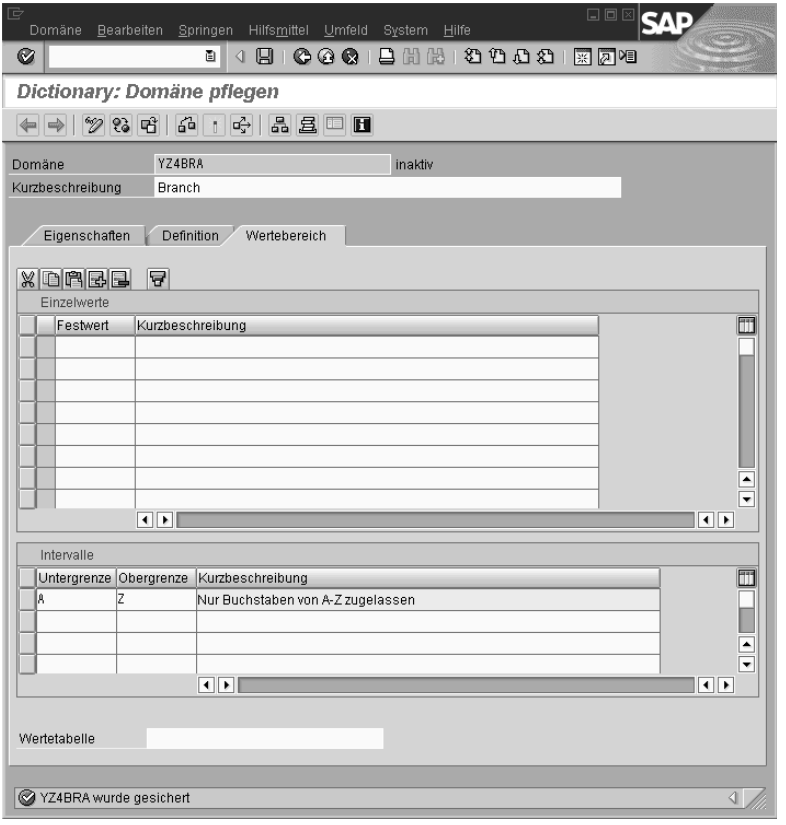

Abbildung 5.3 **© SAP AG**
Pflege der Festwerte für eine Domäne

In den beiden Tabellen werden entweder Festwerte oder Bereiche erfasst. Im jeweils letzten Feld der Tabellen kann für jeden Wert oder jeden Bereich eine kurze Bezeichnung oder Erläuterung eingetragen werden. Diese Bezeichnung erscheint später auch in der automatisch erzeugten Eingabehilfe. Sie sollte also, falls sie benutzt wird, möglichst aussagekräftig sein. Die Einstellung für die Domäne aus Abbildung 5.3 sorgt dafür, dass später nur die Buchstaben A bis Z in ein Datenfeld eingetragen werden können, das auf dieser Domäne beruht. Nach Eingabe der Festwerte wird die Domäne aktiviert. Damit sind die Domänen für die ersten beiden Beispieltabellen angelegt. Der Rest dieses Abschnitts beschreibt einige wichtige Attribute der Domänen.

Format

Bei der Pflege einer Domäne wird dieser ein Format oder Datentyp zugewiesen. Die für Domänen verfügbaren Datentypen entsprechen weder den ABAP-Datentypen noch denen des Datenbanksystems. Sie stellen vielmehr eine Zwischenschicht dar, von der aus Umwandlungen in beide Richtungen möglich sind. Sie können vom Dictionary leicht in die konkreten Datentypen des Datenbanksystems oder einen ABAP-Datentyp umgesetzt werden. Da die Domänen die Verbindung zum Datenbanksystem, genauer gesagt die Verbindung zu einzelnen Feldern einer Datenbanktabelle übernehmen, lehnen sich die Domänen-Datentypen an die Möglichkeiten der Datenbanksysteme an.

Der Datentyp der Domäne bestimmt auch die wesentlichen Eigenschaften des Dictionary-Felds, das später mit Hilfe der Domäne erzeugt wird. Unter anderem verfügen einige der Datentypen über eine Ausgabemaske zur Darstellung des Wertes auf dem Bildschirm. Die für Domänen verfügbaren Datentypen sind in Tabelle 5.1 aufgeführt. Man spricht daher anstelle von Domänen-Datentypen auch von einem *externen Format*. Bei der Auswahl des externen Formats muss die Länge oder der Wertebereich des später in diesem Feld aufzubewahrenden Werts berücksichtigt werden, da die verschiedenen externen Formate (Typen) über unterschiedliche Längen und Wertebereiche verfügen. Beispielsweise stehen die externen Formate `INT1`, `INT2` und `INT4` zur Verfügung, um Integer-Werte mit einer Länge von 1, 2 oder 4 Byte aufzunehmen. Alle diese Formate werden in einem ABAP-Programm in den ABAP-Datentyp I umgesetzt.

Datentyp	Beschreibung	Maske	Stellenzahl	Dezimalstellen/ Vorschlag	Ausgabelänge/ Vorschlag
ACCP	Buchungsperiode JJJJMM	____.__	6		beliebig/6
CHAR	Zeichenketten		<=255		beliebig/ Stellenzahl
CLNT	Mandant		3		3
CUKY	Währungsschlüssel, wird von CURR-Feldern referiert		5		5
CURR	Währungsfeld, abgelegt als DEC	a, b	<=17	<= Stellenzahl/2	beliebig/ Stellenzahl + Trennzeichen + Vorzeichen

Tabelle 5.1
Datentypen für Domänen, Angaben bei Verwendung in Tabellen

Datentyp	Beschreibung	Maske	Stellenzahl	Dezimalstellen/ Vorschlag	Ausgabelänge/ Vorschlag
DATS	Datumsfeld (JJJJMMTT), abgelegt als CHAR(8)	__.__.____	8		10
DEC	Rechen- oder Betragsfeld mit Dezimal- und Vorzeichen	a	<=17		beliebig/ Stellenzahl + Trennzeichen + Vorzeichen
FLTP	Gleitpunktzahl mit 8 Byte Genauigkeit		16	16	beliebig/22 + Vorzeichen
INT1	1-Byte Integer, Dezimalzahl<=254		3		beliebig/3
INT2	2-Byte Integer, nur für Längenfeld vor LCHR oder LRAW		5		beliebig/5 + Vorzeichen
INT4	4-Byte Integer, Dezimalzahl mit Vorzeichen		10		beliebig/10 + Vorzeichen
LANG	Sprachkennzeichen		1		1
LCHR	lange Zeichenfolge, benötigt voranstehendes INT2-Feld		>255		beliebig/ Stellenzahl
LRAW	lange Bytefolge, benötigt voranstehendes INT2-Feld		>255		beliebig/ Stellenzahl
NUMC	Zeichenkette nur mit Ziffern		Beliebig		beliebig/ Stellenzahl
PREC	Genauigkeit eines QUAN-Feldes				
QUAN	Mengenfeld, zeigt auf ein Einheitenfeld mit Format UNIT	a	<=17	<= Stellenzahl	beliebig/ Stellenzahl + Trennzeichen + Vorzeichen

Tabelle 5.1
Datentypen für Domänen, Angaben bei Verwendung in Tabellen (Fortsetzung)

Datentyp	Beschreibung	Maske	Stellenzahl	Dezimalstellen/ Vorschlag	Ausgabelänge/ Vorschlag
RAW	uninterpretierte Folge von Bytes		Beliebig		beliebig/ doppelte Stellenzahl
RAWSTRING	Variabel lange Folge von Bytes		Beliebig		Beliebig/15
STRING	Variabel lange Zeichenfolge		Beliebig		Beliebig/15
TIMS	Zeitfeld (HHMMSS), abgelegt als Char(6)	__:__:__	6		beliebig/8
VARC	lange Zeichenkette, ab Rel. 3.0 nicht mehr unterstützt		Beliebig		Beliebig
UNIT	Einheitenschlüssel für QUAN-Felder		2 oder 3		Stellenzahl

Tabelle 5.1
Datentypen für Domänen, Angaben bei Verwendung in Tabellen (Fortsetzung)
(a: Tausender- und Dezimalpunkte werden vom System gesetzt,
b: Zahl der Dezimalstellen abhängig von Währung)

Einige Datentypen besitzen eine vorgegebene Länge, für andere Typen kann die Feldlänge in gewissen Grenzen frei gewählt werden. Ein Eintrag in das Feld ZAHL DER STELLEN ist wegen der Eingabeprüfung des Dynpros auf jeden Fall erforderlich, auch wenn später beim Abspeichern der Werte oder bei Betätigen der Datenfreigabetaste die Feldlänge durch das System automatisch auf einen Vorgabewert gesetzt oder korrigiert wird. Sofern Sie den Datentyp über die Eingabehilfe auswählen, wird die korrekte Stellenzahl, sofern sie bestimmt werden kann, automatisch gesetzt.

Alle in Domänen verwendeten Datentypen werden durch das Data-Dictionary später auf die ABAP-Datentypen abgebildet. Dabei können verschiedene externe Formate in denselben ABAP-Datentyp transformiert werden. Falls ein programminternes Datenfeld den Inhalt eines Tabellenfeldes aufnehmen soll, muss es den Typ und die Länge besitzen, in den der externe Datentyp umgewandelt wird. Am einfachsten ist dies durch Deklaration des Datenfeldes mit Hilfe der `LIKE`-Anweisung möglich. Falls diese Anweisung nicht verwendet werden soll, liefert Tabelle 5.2 einige Informationen über die Transformationen externer Datentypen zu ABAP-Typen.

Externer Datentyp	Darstellung in ABAP
ACCP	N(6)
CHAR n	C(n)
CLNT	C(3)
CUKY	C(5)
CURR n,m,s	P((n+2)/2) DECIMALS m (NO-SIGN)
DATS	D(8)
DEC n, m, s	P((n+2)/2) DECIMALS m (NO-SIGN)
FLTP	F(8)
INT1	I
INT2	I
INT4	I
LANG	C(1)
LCHR	C(n)
LRAW	X(n)
NUMC n	N(n)
PREC	
QUAN n, m, s	P((n+2)/2) DECIMALS m (NO-SIGN)
RAW n	X(n)
RAWSTRING n	XSTRING n
STRING x	STRING n
TIMS	T(6)
VARC n	C(n)
UNIT	C(n)

Tabelle 5.2
Darstellung der externen Formate in ABAP (n = Stellen, m = Nachkommastellen, s = 1 für Vorzeichen, sonst 0)

Wertebereich

Beim Erfassen von Werten in Dynpros kann das System Felder auf zulässige Eingaben prüfen. Zum Teil erfolgt dies durch entsprechende Anweisungen in der Ablauflogik, vor allem dann, wenn die Prüfungen etwas umfangreicher sind. Die Prüfung gegen eine vorgegebene Wertemenge kann aber auch automatisch durchgeführt werden. Als Nebeneffekt steht für alle Felder, für die derartige automatische Prüfungen stattfinden, auch eine Eingabehilfe bereit, in der die zulässige Wertemenge angezeigt wird. Von besonderer Bedeutung ist dieser Prüfmechanismus für spezielle, vom System bereitgestellte Pflegebausteine (z.B. Transaktionen SM30, SM31, SM32). Mit diesen Werkzeugen können Tabelleninhalte gepflegt werden, ohne dass dazu eigene Programme geschrieben werden müssen. Der Einsatz dieser Pflegewerkzeuge erfolgt vor allem beim Customizing.

Möglich werden die automatischen Prüfungen durch Definition der Wertemenge in einer Domäne. Dies kann zum einen durch Definition von Festwerten erfolgen, zum anderen durch Angabe einer Wertetabelle. Diese Wertetabelle muss ein Schlüsselfeld besitzen, das auf der jeweiligen Domäne beruht. Bei späteren Eingaben sind nur die Werte zugelassen, die in der Wertetabelle im entsprechenden Feld enthalten sind. Die Angabe einer Wertetabelle allein reicht allerdings nicht für eine Prüfung aus. Dazu sind zusätzlich noch so genannte Fremdschlüssel zu definieren. Da dieses Thema sehr komplex ist, wird es in Abschnitt 5.1.5 detailliert erläutert.

Die Festwerte einer Domäne sowie der Name der Wertetabelle werden in einem eigenen Bereich des Dynpro bearbeitet. Eine Tabelle kann nur für eine einzige Domäne Wertetabelle sein. Falls die im Feld WERTETABELLE eingetragene Tabelle bereits für eine andere Domäne Wertetabelle ist, erfolgt eine Fehlermeldung beim Aktivieren.

Ausgabeeigenschaften

In der Feldgruppe AUSGABEEIGENSCHAFTEN innerhalb der Registerkarte DEFINITION werden Attribute angegeben, die Einfluss auf die Darstellung des Feldes in einer Liste oder einem Dynpro haben. Einige Datentypen nehmen gebrochene Zahlen auf. Im Feld DEZIMALSTELLEN kann angegeben werden, wie viele Dezimalstellen zugelassen sein sollen. Für einige externe Formate, beispielsweise FLTP für Fließkommazahlen, wird dieser Wert fest vorgegeben, für andere kann er frei gewählt werden. Wenn für das Vorzeichen eine Stelle reserviert werden soll, so ist das Flag VORZEICHEN zu markieren. Abhängig von der internen Darstellung eines Wertes sowie den für Dezimalstellen und Vorzeichen erforderlichen Stellen kann die zur Ausgabe eines Wertes erforderliche Stellenzahl von der Feldlänge abweichen. Die Ausgabelänge wird beim Ändern der anderen Werte vom System automatisch berechnet und als Vorgabe im entsprechenden Eingabefeld bereitgestellt. Dieser Wert kann manuell überschrieben werden. Weicht der Wert im Feld AUSGABELÄNGE von der berechneten Länge ab, so erfolgt in den meisten Fällen eine Warnung. Bei Bedarf kann die Ausgabelänge

also abweichend von der realen Länge gewählt werden. Es kann beispielsweise bei sehr langen Char-Feldern sinnvoll sein, die Ausgabelänge geringer zu wählen als die Länge des Datenbankfelds. Einige Standardwerkzeuge wie die der Data Browser (Transaktion SE16) oder die beiden Werkzeuge zur Tabellenpflege (SM30, SM31 und SM32) greifen auf die Angaben im Dictionary zu, um Ausgabelisten oder Pflegedynpros zu erzeugen. Diese Werkzeuge würden versuchen, sehr lange Felder in ihrer Originallänge auf dem Bildschirm anzuzeigen, was zu einem sehr unübersichtlichen oder missverständlichen Bild führen kann. Wenn in der Domäne eine kürzere Ausgabelänge für solche Felder gesetzt wird, können derartige unerwünschte Nebeneffekte gemildert werden.

Im Abschnitt über Reports, speziell über die Varianten des Kommandos WRITE, wurden die Konvertierungsexits erwähnt. Dabei handelt es sich um Funktionsbausteine, die den Wert eines Feldes vor der Ausgabe bzw. nach der Eingabe umwandeln. Für jede Domäne kann ein derartiger Konvertierungsexit angegeben werden. Der variable Teil des Namens eines Konvertierungsexits ist fünf Zeichen lang. Er kann im Feld KONVERT-ROUTINE eingetragen werden. Von derartigen Konvertierungsroutinen wird allerdings nur relativ selten Gebrauch gemacht. Wesentlich wichtiger ist das letzte Flag KLEINBUCHSTABEN. Dieses Flag ist normalerweise nicht aktiv. In diesem Zustand bewirkt es bei Eingaben über Dynpros die automatische Umwandlung aller Buchstaben in Großbuchstaben. Diese Eigenschaft ist vor allem für Schlüsselfelder von Bedeutung, um deren einheitliche Schreibweise zu erzwingen. Erst wenn das Flag aktiviert wird, findet diese Umwandlung nicht statt, die Zeichen werden in der Schreibweise, in der sie eingegeben wurden, auch abgespeichert.

Die Konvertierung in Großbuchstaben findet nicht beim Schreiben des Wertes in die Datenbank statt, sondern innerhalb der Dynproverarbeitung. Es ist daher auch bei deaktiviertem Kleinschreibungsflag möglich, Werte in die Datenbank zu schreiben, die Kleinbuchstaben enthalten, sofern diese Werte nicht über Dynpros eingegeben, sondern beispielsweise im Programm hart kodiert werden. Dieses Flag schützt also nicht vor Programmierfehlern.

5.1.2 Datenelemente

Nach der Deklaration der Domänen können die Datenelemente erzeugt werden. Die Datenelemente greifen auf die technischen Angaben einer Domäne zurück und ergänzen diese mit betriebswirtschaftlichen oder programmspezifischen Informationen. Datenelemente und Domänen besitzen getrennte Namensräume. Sie können daher identische Namen besitzen. Da Domänen und Datenelemente einander oft eindeutig zugeordnet sind, tragen identische Namen hier ausnahmsweise zur Erhöhung der Übersichtlichkeit bei.

Angelegt werden Datenelemente ähnlich wie Domänen vom Grundbild des Dictionary-Pflegewerkzeuges aus. Im Gegensatz zu früheren Versionen der R/3-Software sind Datenelemente kein völlig eigenständiges Objekt, sondern

gehören zur Obergruppe DATENTYP. Nach Eintragen des Namens YZ4DESCR im Eingabefeld und Markieren des DATENTYP-Feldes kann die Drucktaste ANLEGEN betätigt werden. Zunächst ist in einem kleinen Popup (Abbildung 5.4) festzulegen, ob ein einfacher Typ (Datenelement), eine Struktur (Feldleiste) oder ein Tabellentyp (Typbeschreibung für interne Tabelle) angelegt werden soll.

Abbildung 5.4 **© SAP AG**
Auswahl der konkreten Typeigenschaft

Danach erscheint das Pflegebild für Datenelemente (Abbildung 5.5). Es ähnelt in seinem Aufbau dem Pflegebild für Domänen.

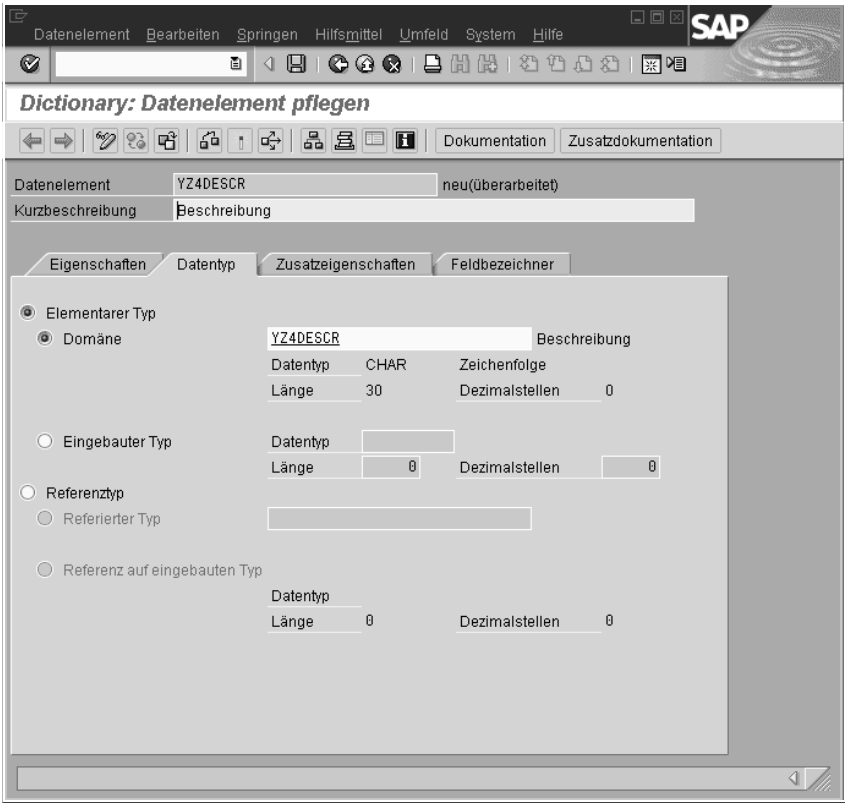

Abbildung 5.5 **© SAP AG**
Pflege eines Datenelements

Zunächst ist wieder die obligatorische Kurzbezeichnung zu erfassen. Die eigentliche Aufgabe beim Erzeugen eines Datenelementes besteht in der Zuordnung eines technischen Datentyps. Seit Einzug der objektorientierten Programmierung in ABAP können Datenelemente einer von zwei unterschiedlichen Datentypgruppen angehören. Zur ersten Gruppe gehören die einfachen Felder. Sie nehmen einen einzigen unstrukturierten Wert eines bestimmten Typs auf. Diesen Datentyp bestimmen Sie entweder durch eine Domäne oder aber durch die direkte Verwendung eines der bereits bei den Domänen erläuterten eingebauten Datentyps. Dabei geht aber der Vorteil der Wertebereichsprüfung verloren. Aus diesen beiden Varianten wählen Sie, indem Sie das Auswahlfeld ELEMENTARER TYP sowie eines der beiden Auswahlfelder DOMÄNE oder EINGEBAUTER TYP betätigen.

Die zweite Auswahlmöglichkeit steht im Zusammenhang mit der objektorientierten Programmierung. Durch Auswahl des Feldes REFERENZTYP legen Sie fest, dass das so definierte Datenelement eine Referenz auf eine Klasse oder ein Interface aufnehmen soll. In diesem Fall ist im Eingabefeld REFERENZ AUF der Name einer Klasse oder eines Interface einzutragen.

Da Datenelemente die semantische Bedeutung des Feldes bestimmen, müssen Sie auch noch Feldtexte pflegen, die in Dynpros als Schlüsseltext benutzt werden. Diese Bezeichner erscheinen als Schlüsselfelder in Dynpros oder Spaltenüberschrift in automatisch generierten Listen. Sie müssen daher korrekt und möglichst aussagekräftig sein. Letztlich entscheiden die Kurzbeschreibung und die Feldbezeichner über die Interpretation des Feldes in Anwendungen. Sie geben Auskunft darüber, welche Informationen im entsprechenden Tabellenfeld gespeichert werden sollen. Zur Pflege der Bezeichner wählen Sie die Registerkarte FELDBEZEICHNER (siehe Abbildung 5.6).

In diesem Bild tragen Sie die gewünschten Texte ein und legen außerdem die maximale Länge der Texte fest. Ältere Versionen verwendeten hier standardmäßig vorgegebene Werte.

Nach Eingabe aller Attribute wird das Datenelement aktiviert. Ebenso wie bei Domänen werden Änderungen an Datenelementen erst nach einer Aktivierung wirksam. Falls das Datenelement bereits benutzt wird, werden automatisch auch alle abhängigen Dictionary-Objekte aktiviert.

Nachdem das Datenelement `YZ4DESCR` erfolgreich angelegt wurde, wird der Vorgang für das Datenelement `YZ4BRA` wiederholt. Es wird mit der Domäne `YZ4BRA` verbunden und erhält die Bezeichnungen *Branche*.

Auf der Registerkarte ZUSATZEIGENSCHAFTEN erscheinen noch einige Eingabefelder, die in professionellen Anwendungen von erheblicher Bedeutung sein können. Das SAP-System stellt einige Verfahren zum anwendungsübergreifenden Datenaustausch über globale Speicherbereiche zur Verfügung. Eine dieser Möglichkeiten ist die Verwendung von Get/Set-Parametern. Ein solcher Parameter erhält einen eindeutigen Bezeichner.

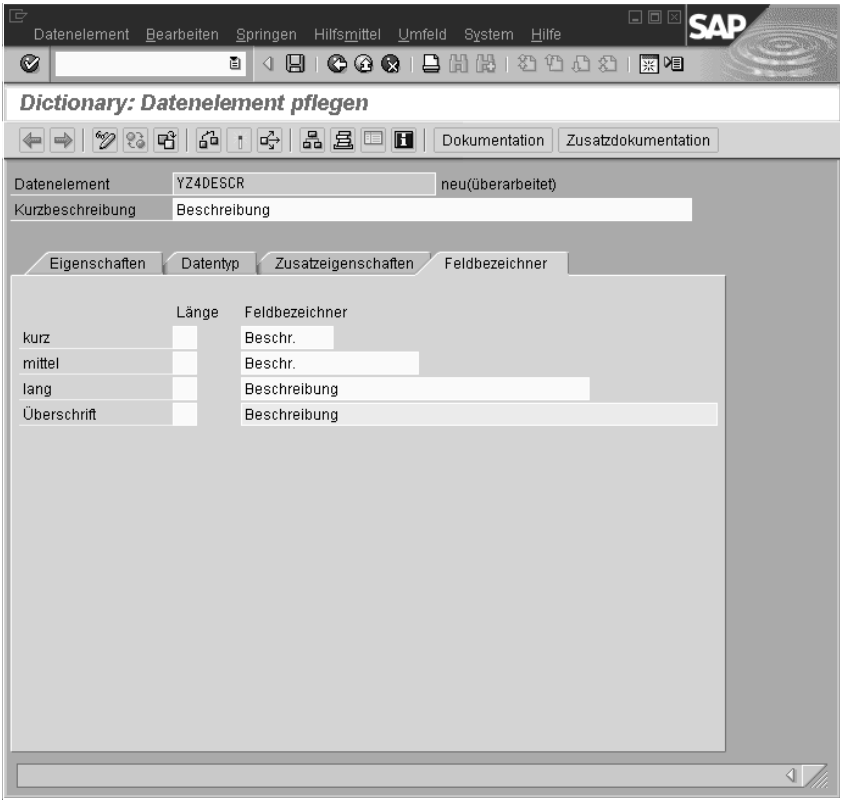

Abbildung 5.6 **© SAP AG**
Pflege der Bezeichner

Mittels spezieller ABAP-Anweisungen (`SET PARAMETER`, `GET PARAMETER`) wird ein Wert in einen solchen Parameter geschrieben oder aus ihm gelesen. Der Inhalt eines Parameters bleibt während der gesamten Sitzung eines Anwenders, also vom An- bis zum Abmelden, erhalten. Dynprofeldern kann als Attribut ein solcher Parameter zugewiesen werden. Bei der Abarbeitung des Dynpros wird das Feld dann automatisch mit dem Inhalt des Parameters gefüllt oder der Wert des Feldes in den Parameter geschrieben. Dies erspart unter Umständen lästige Eingaben in diversen Transaktionen. Beispielsweise speichert das System den Namen eines von Ihnen bearbeiteten Programms in einem solchen Parameter. Beim direkten Aufruf der verschiedenen Pflegewerkzeuge, etwa des Screen Painters oder des Menu Painters, wird das Eingabefeld für den Programmnamen automatisch gefüllt.

Falls einem Datenelement ein solcher Parameter zugewiesen wurde, wird dieser Parameter einem auf dem entsprechenden Datenelement beruhenden Dynprofeld automatisch zugewiesen.

Das SAP-System verfügt über die Möglichkeit, Datenänderungen selektiv protokollieren zu können. Dazu sind mit einer speziellen Transaktion so genannte Änderungsbelegobjekte zu erzeugen und einige zusätzliche Funktionen in Programme einzubinden. Diese Funktionen erkennen automatisch Änderungen eines Datensatzes und erzeugen in diesem Fall einen Änderungsbeleg. Dabei werden allerdings nur Änderungen in Datenfeldern berücksichtigt, in denen das Flag *Änderungsbeleg* gesetzt ist.

Datenelemente können mit einer Suchhilfe verbunden werden. Bei Verwendung des Datenelements in einem Dynpro ist dann immer diese Suchhilfe aktiv, unabhängig davon, um welches konkrete Tabellenfeld es sich handelt. Die Angaben für die Suchhilfe erfolgen in den beiden Eingabefeldern NAME und PARAMETER innerhalb des Teilbereichs SUCHHILFE.

5.1.3 Tabellen

Nach Bereitstellen aller erforderlichen Datenelemente und Domänen kann eine Tabelle erzeugt werden. Tabellen sind im Gegensatz zu Datenelementen und Domänen nicht nur Definitionen im Data-Dictionary, sondern real auf der Datenbank existierende Objekte. Bei der Erzeugung von Tabellen sind daher zusätzliche Arbeitsschritte erforderlich, deren Demonstration im Mittelpunkt des folgenden Abschnitts stehen soll. Erzeugt werden zwei Tabellen zur Aufnahme eines Kennbuchstabens für Wirtschaftsbranchen und eines sprachabhängigen Textes zu diesen Kennbuchstaben. Im später zu beschreibenden Beispiel dienen diese Tabellen zur Validierung der Eingaben in eine dritte Tabelle, in der Stammdaten für Aktien erfasst werden.

Tabellenstruktur

Zunächst müssen die allgemeinen Attribute und die Struktur der Tabelle definiert werden. Dazu wird im Grundbild der Dictionary-Pflege das Auswahlfeld DATENBANKTABELLE markiert, der Tabellenname (`YZ4B`) eingetragen und wieder die Drucktaste ANLEGEN betätigt. Es erscheint ein Dynpro, dessen Gestaltung als Tab Strip an die bereits vorgestellten Pflegedynpros erinnert, in dem neben einigen anderen Attributen auch die Datenfelder der Tabelle gepflegt werden müssen.

Zunächst ist wie gewohnt eine Kurzbeschreibung einzutragen. Den konkreten Text können Sie beliebig wählen. Er sollte aber auf die Aufgabe der Tabelle, zulässige Branchen aufzunehmen, hindeuten.

In der aktuellen Karteikarte AUSLIEFERUNG UND PFLEGE (Abbildung 5.7) müssen zunächst zwei grundlegende Eigenschaften gepflegt werden. Im Feld AUSLIEFERUNGSKLASSE ist eine Eingabe unbedingt erforderlich. Der Inhalt dieses Feldes ist im Rahmen der hier erläuterten Beispiele relativ bedeutungslos. In produktiven SAP-Anwendungen hat die Auslieferungsklasse allerdings Einfluss darauf,

ob und in welchem Umfang in diese Tabelle Einträge aus anderen Systemen importiert werden dürfen. Für YZ4B und die danach anzulegende Tabelle YZ4BT können Sie die Auslieferungsklasse C (Customizing-Tabelle) benutzen.

Bereits mehrfach erwähnt wurde, dass für Tabellen spezielle Pflegeprogramme generiert werden können. Dies ist allerdings nur möglich, wenn im Feld DATA-BROWSER/TABELLENSICHT-PFLEGE ein passender Wert gesetzt ist. Da die beiden Tabellen YZ4B und YZ4BT später mit solchen Werkzeugen gepflegt werden sollen, ist bei beiden Tabellen die Einstellung *Anzeige/Pflege erlaubt* auszuwählen. Die Bearbeitung der Tabellen in selbst programmierten Dialoganwendungen ist von diesem Flag nicht betroffen.

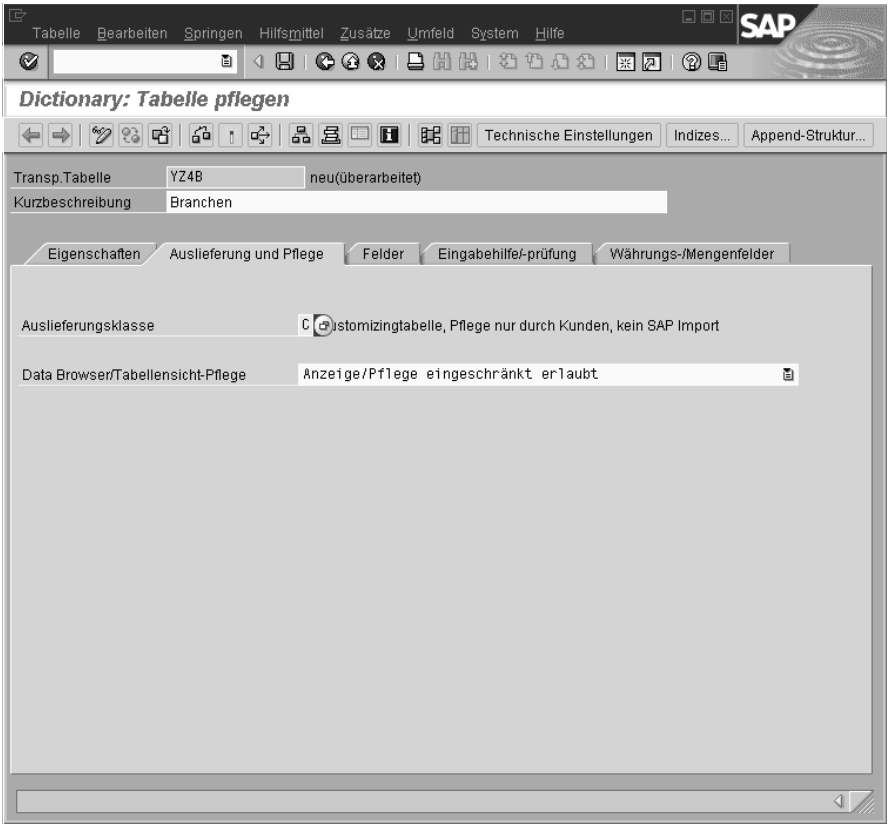

Abbildung 5.7 **© SAP AG**
Auslieferungs- und Pflegeeigenschaften der Tabelle definieren

Nach diesen Vorarbeiten kann die Datenstruktur festgelegt werden. Wechseln Sie dazu zum Register FELDER. Dort steht eine Tabelle zur Verfügung, der in einem Arbeitsgang die Definition mehrerer Felder ermöglicht. In der Spalte FELDER wird der Name des anzulegenden Feldes eingetragen. Es sollten, abgesehen

von der Länge, dieselben Konventionen wie bei den Namen programminterner Datenfelder beachtet werden. Feldnamen müssen innerhalb einer Tabelle natürlich eindeutig sein. Falls das Feld ein Schlüsselfeld sein soll, ist das Flag KEY zu markieren. Die Schlüsselfelder müssen am Anfang der Tabelle stehen. Die Beschreibung der Eigenschaften des Datenfelds erfolgt durch Angabe eines Datenelementes in der dritten eingabebereiten Spalte.

An dieser Stelle ist eine Anmerkung erforderlich. Ab Release 3.0C kann der Bezug auf ein Datenelement durch die direkte Eingabe eines Datentyps ersetzt werden. Dies ist für Hilfsfelder sinnvoll, die nie in einem Dynpro oder Report erscheinen sollen, da für derartige Felder weder Überschriften, Bezeichnungen oder Fremdschlüsselbeziehungen gepflegt werden müssen. Die direkte Eingabe der Datentypen ist erst möglich, nachdem der Pflegemodus für die Tabellenfelder mit der Menüfunktion BEARBEITEN | EINGEBAUTER TYP umgeschaltet wurde. Im weiteren Verlauf wird von dieser Möglichkeit allerdings kein Gebrauch gemacht. Von der direkten Typeingabe wird mit BEARBEITEN | DATENELEMENT wieder zur Definition der Spalteneigenschaften per Datenelement zurückgeschaltet. Außerdem stehen für die genannten Menüfunktionen entsprechende Drucktasten auf der Registerkarte zur Feldpflege bereit.

Die erste Tabelle `YZ4B` soll lediglich zwei Felder aufnehmen. Das SAP-System ist mandantenfähig. Die erzeugten Entwicklungsobjekte, gleich ob Programme, Tabellen oder andere Elemente, sind aber im gesamten System gültig. Dies erfordert, dass in Tabellen, die mandantenbezogene Daten aufnehmen sollen, ein Feld zur Aufnahme des Mandanten zur Verfügung steht. Das Data-Dictionary verarbeitet dieses Feld bei Zugriffen auf die Tabelle automatisch. In Anwendungen ist es nicht erforderlich, dieses Feld in irgendeiner Weise auszuwerten. Es muss in mandantenabhängigen Tabellen aber immer als das erste Feld einer Tabelle angelegt werden. Dieses Feld heißt immer `MANDT`. Es ist ein Schlüsselfeld und baut auf dem gleichnamigen Datenelement `MANDT` auf. Dieses Datenelement ist bereits im SAP-Standard enthalten. Es kann und darf nicht selbst angelegt oder bearbeitet werden.

Das zweite Feld der Tabelle ist das Feld `BRANCH`. Es soll ebenfalls ein Schlüsselfeld sein und das Datenelement `YZ4BRA` benutzen. Nach Eingabe der Werte und Betätigen der Datenfreigabetaste oder dem Sichern der Eingaben ermittelt das System anhand der Datenelemente einige wichtige Angaben und füllt damit die restlichen Spalten der Tabellendefinition. Das Pflegedynpro sollte nun etwa das in Abbildung 5.8 dargestellte Aussehen haben.

Beim Abspeichern erfragt das System ggf. Eigenschaften zur späteren Erweiterbarkeit der Tabelle (siehe Abbildung 5.9). In diesem Beispiel ist die korrekte Auswahl nebensächlich, es kann daher die Einstellung *beliebig erweiterbar* benutzt werden.

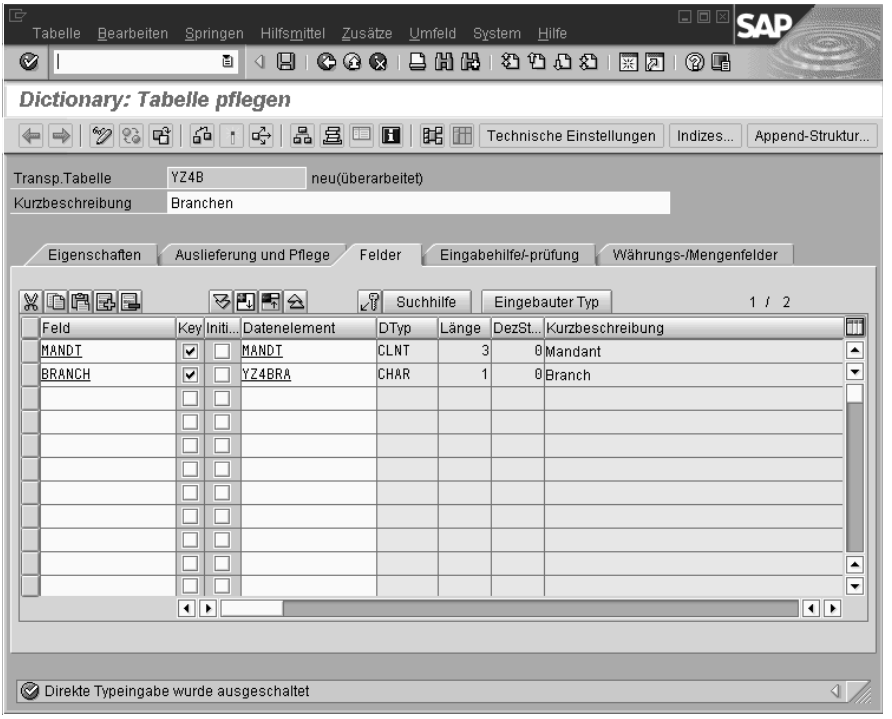

Abbildung 5.8 **© SAP AG**
Pflege der Datenfelder einer Datenbanktabelle

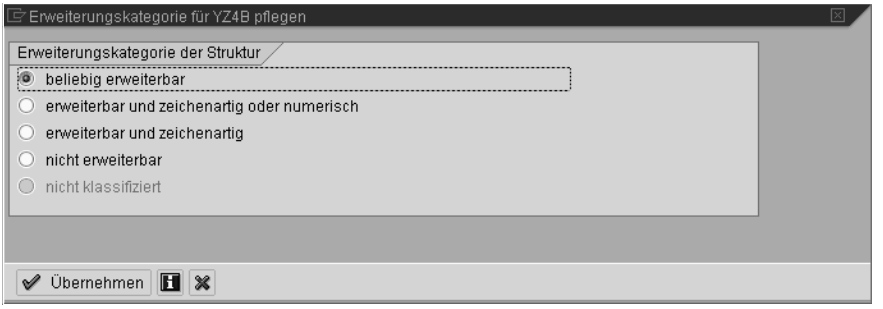

Abbildung 5.9 **© SAP AG**
Eigenschaften zur Erweiterbarkeit pflegen

Währungs- und Mengenfelder

Tabellenfelder, die Währungsbeträge oder Mengenangaben enthalten (Typen `CURR` und `QUAN`), werden auf spezielle Weise behandelt. In zentralen Tabellen (z.B. `TCURC` und `T006`) werden Angaben zu den Währungen und Maßeinheiten

angelegt, wobei ein systemweit eindeutiger Schlüssel für die Währung oder die Maßeinheit vergeben wird. Diese Angaben werden benutzt, um ein Feld mit Währungsbeträgen oder Maßeinheiten näher zu beschreiben. Jedem Feld des Typs CURR bzw. QUAN muss daher ein weiteres Feld, das so genannte Referenzfeld, zugeordnet werden, in dem zur Laufzeit einer Anwendung ein Währungs- oder Einheitenschlüssel verfügbar ist. Dieses Feld kann in der eigenen Tabelle liegen, es kann sich aber auch in einer anderen Tabelle befinden. Bei der Darstellung eines solchen Feldes in einem Dynpro wird der Währungs- oder Einheitenschlüssel ermittelt und das Feld entsprechend formatiert. Die Zuordnung des Referenzfeldes erfolgt innerhalb der dritten Registerkarte. Die Eingabefelder für Referenztabelle und Referenzfeld sind nur für Felder der genannten Typen eingabebereit.

Technische Einstellungen

Die relationalen Datenbanksysteme legen Tabellen nicht wahllos auf dem Massenspeicher an, sondern versuchen, Platzbedarf und Zugriffsgeschwindigkeit zu optimieren. Dazu erwarten einige der Datenbanksysteme spezielle Angaben seitens des Programmierers. Diese und einige weitere Angaben gehören zu den so genannten technischen Einstellungen einer Tabelle. Sie werden in einem zweiten Dynpro erfasst (Abbildung 5.10). Dieses ist entweder über die Drucktaste TECHNISCHE EINSTELLUNGEN oder die Menüfunktion SPRINGEN | TECHNISCHE EINSTELLUNGEN zugänglich. In einigen Fällen, z. B. beim Versuch, eine Tabelle ohne technische Einstellungen zu aktivieren, springt das System auch automatisch in dieses Dynpro.

Die Angaben in diesem Dynpro sind nur erforderlich, damit das Datenbanksystem den Zugriff auf die Tabelle optimieren kann. Die technischen Einstellungen haben keinen Einfluss auf die prinzipielle Funktionsfähigkeit der Tabelle, sondern nur auf die Performance des Tabellenzugriffs. Sie sind daher unkritisch im Sinne der reinen Funktionsfähigkeit einer Anwendung, nicht aber der Performance.

Da sich die möglichen Werte für die technischen Einstellungen je nach verwendetem Datenbanksystem voneinander unterscheiden können, sollte die Eingabehilfe benutzt und aus den dort angebotenen Werten ein sinnvoller Eintrag herausgesucht werden.

Von Bedeutung sind vor allem drei Werte. Der erste, die DATENART, liefert eine Aussage über die Zugriffshäufigkeit. Im Beispiel wird APPL2 für Customizing-Daten gewählt, auf die selten zugegriffen wird. Das zweite Feld GRÖSSENKATEGORIE teilt der Datenbank die zu erwartende Anzahl von Datensätzen mit. Diese kann dann zusammenhängenden Speicherplatz für die Tabelle reservieren. Da in unserem Beispiel nur mit sehr wenigen Datensätzen gearbeitet wird, reicht der niedrigste Wert völlig aus. Die dritte erforderliche Angabe betrifft die Pufferungsart. Standardmäßig wird vom System die Auswahl NICHT GEPUFFERT markiert. Diese Auswahl kann beibehalten werden.

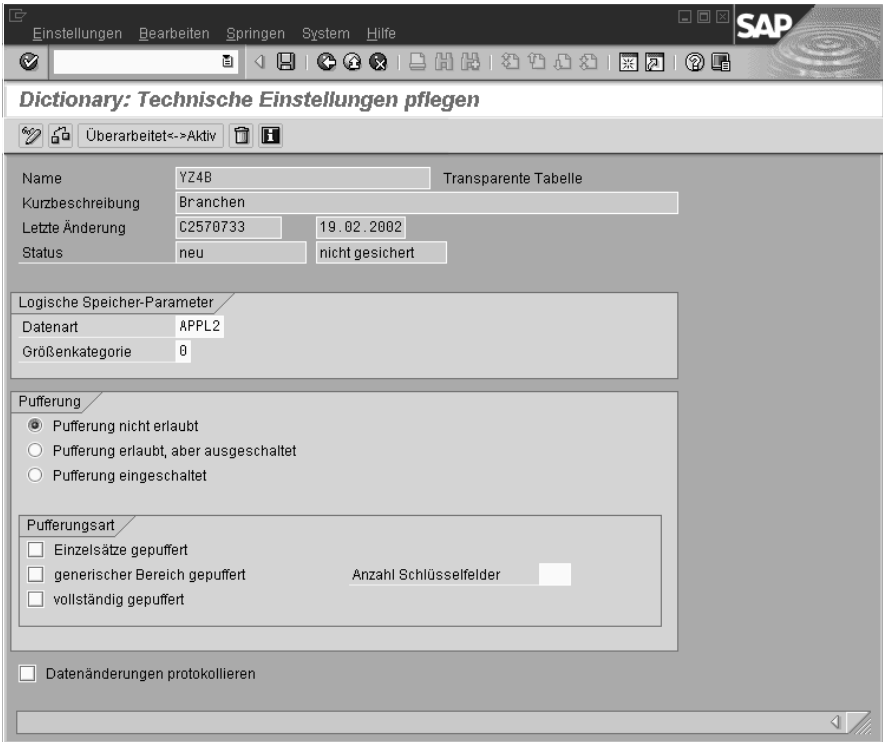

Abbildung 5.10 **© SAP AG**
Pflege der Datenfelder einer Datenbanktabelle

Zur Pufferung sind einige weitergehende Informationen notwendig. Das SAP-System kann ganze Tabellen oder ausgewählte Teile davon in einem internen Puffer halten und so den Zugriff auf die Daten beschleunigen. Die Pufferung ist nur sinnvoll, wenn innerhalb einer Transaktion auf Datensätze mehrfach lesend zugegriffen wird. Die Wirksamkeit der Pufferung wird verschlechtert, wenn Schreibvorgänge erfolgen. Beim Lesen eines Satzes einer Datenbanktabelle werden entsprechend der eingestellten Pufferungsart automatisch weitere Sätze gelesen und in einen Puffer gestellt. Wenn die vollständige Pufferung ausgewählt wurde, wird die Tabelle entweder komplett in den Puffer geladen oder gar nicht, falls der Speicherplatz nicht reicht. Diese Pufferungsart wird nur für sehr kleine Tabellen bis etwa 30 Kbyte Umfang empfohlen.

Bei der generischen Pufferung werden alle Datensätze mit identischem Schlüssel gepuffert. Daher ist bei dieser Pufferungsart die Anzahl der Schlüsselfelder zu benennen, die bei der Pufferung ausgewertet werden sollen. Diese muss mindestens um den Wert 1 geringer sein als die Gesamtzahl der Schlüsselfelder, sonst würde die Pufferung der ersten Variante entsprechen. Sinnvoll ist eine derartige Pufferung beispielsweise für sprachabhängige Tabellen. Wenn eine

Tabelle generisch gepuffert werden soll, muss die Reihenfolge der Schlüsselfelder entsprechend optimiert werden. Die vorderen Teile des Schlüssels sollten für die nachfolgend zu lesenden Sätze möglichst konstant sein, damit der Puffer nicht immer wieder neu gefüllt werden muss. In der Praxis bedeutet dies beispielsweise, dass bei sprachabhängigen Tabellen das Feld mit dem Sprachenkennzeichen unmittelbar nach dem Mandantenfeld folgen sollte.

Bei der Einzelpufferung werden die gelesenen Datensätze in einem Puffer aufbewahrt. Diese Pufferungsart lohnt sich nur, wenn auf relativ wenige Datensätze mit völlig unterschiedlichen Schlüsseln sehr häufig lesend zugegriffen wird.

Nach dem Bearbeiten der technischen Einstellungen können diese abgespeichert werden. Mit der Funktionstaste F3 oder dem entsprechenden Symbol ist die Rückkehr zum zentralen Pflegedynpro der Tabellenpflege möglich. Dort wird die Tabelle aktiviert. Beim Aktivieren wird die Tabelle sowohl im Data Dictionary als aktiv gekennzeichnet als auch als reale Tabelle auf der Datenbank angelegt.

Nach der erfolgreichen Bearbeitung der Tabelle YZ4B kann die zweite Tabelle YZ4BT angelegt werden. Sie soll zu jeder Branche einen erläuternden Text aufnehmen, der für jede Anmeldesprache getrennt gepflegt werden kann. Tabelle 5.3 zeigt die in die Tabelle aufzunehmenden Felder und Datenelemente.

Feldname	Key	Datenelement
MANDT	X	MANDT
LANGU	X	SPRAS
BRANCH	X	YZ4BRA
DESCR		YZ4DESCR

Tabelle 5.3
Struktur der Tabelle YZZBT

Das Datenelement SPRAS ist ebenso wie MANDT ein bereits vom SAP-Standard vorgegebenes Element. Die Verwendung dieses Datenelementes und eines darauf aufbauenden Sprachenfelds ist für Tabellen mit sprachabhängigen Texten zwingend erforderlich. Das Sprachenfeld muss natürlich zum Schlüssel gehören, es ist aber nicht Bedingung, dass es nach dem Mandanten an zweiter Stelle steht. Die technischen Einstellungen werden ebenso wie für die Tabelle YZ4B gepflegt, wobei dieselben Werte benutzt werden können. Die Tabelle wird daraufhin aktiviert. Damit stehen zwei Tabellen zur weiteren Bearbeitung zur Verfügung.

Die Bearbeitung von Tabellendefinitionen im Dictionary ist relativ einfach. Die meisten der zur Verfügung stehenden Menüfunktionen erschließen sich durch ihren Namen oder durch einfaches Ausprobieren. Einige Details der Eigenschaften von Tabellen sollen die bisherigen Informationen ergänzen.

Wenn Tabellen durch neue Felder erweitert werden sollen, erfolgt dies üblicherweise durch Einfügen der neuen Felder in die Feldliste. Nach Möglichkeit sollten Felder immer am Ende der Tabelle angefügt werden. In diesem Fall ist es für das Datenbanksystem einfacher, die Struktur der Datenbanktabelle anzupassen. Im Tabellen-Pflegebild steht für das Einfügen neuer Tabellenfelder die Drucktaste ZEILE EINFÜGEN zur Verfügung.

Fremdschlüssel definieren

Beide Tabellen sind zwar noch nicht technisch, aber logisch eng miteinander verbunden. In einer späteren Anwendung soll der Anwender gleichzeitig in der Tabelle `YZ4B` einen Wert erfassen und in `YZ4BT` die entsprechende sprachabhängige Erläuterung eingeben können. Dies soll nicht durch eine selbst geschriebene Dialogtransaktion, sondern durch die vom System bereitgestellte Tabellenpflege erfolgen. Dazu ist später für die Tabelle `YZ4B` ein Pflegebaustein zu generieren. Damit dieser Baustein die beiden Tabellen korrekt miteinander verknüpfen kann, muss ein so genannter Fremdschlüssel angelegt werden. Ein Fremdschlüssel definiert Abhängigkeiten von Tabellen. Er wird von verschiedenen Werkzeugen automatisch ausgewertet. Die theoretischen Grundlagen von Fremdschlüsseln erläutert ein eigener Abschnitt. Hier soll nur demonstriert werden, wie die beiden Tabellen `YZ4B` und `YZ4BT` über einen Fremdschlüssel miteinander verbunden werden.

Die Verknüpfung der beiden Tabellen erfolgt über das Tabellenfeld `BRANCH`. Die Texte in `YZ4BT` sollen ja einer Branche und damit einem Datensatz aus `YZ4B` zugeordnet werden. Die eigentliche Verbindung wird über die Domäne des Datenbankfelds hergestellt.

Die Texttabelle ist die untergeordnete Tabelle. In ihr sollen nur Texte zu Branchen enthalten sein, die in `YZ4B` bereits existieren. In der Domäne `YZ4BRA` wird daher neben den Festwerten eine Wertetabelle eingetragen. Diese Wertetabelle ist `YZ4B`. Um die Wertetabelle einzutragen, kann die Domäne natürlich vom Grundbild der Dictionary-Pflege aus aufgerufen werden. Aber wie bei der Programmpflege ist auch in der Dictionary-Pflege ein ausgefeilter Navigationsmechanismus verfügbar. Ein Doppelklick auf den Namen eines Datenelements innerhalb der Feldliste der Tabellenstruktur springt direkt zum Pflegedynpro dieses Datenelementes. Von dort aus ist nach einem erneuten Doppelklick auf den Namen der Domäne das Pflegen dieser Domäne möglich. Zur Sicherheit gegen irrtümliche Eingaben arbeiten die aufgerufenen Pflegetransaktionen zunächst nur im Anzeigen-Modus. Soll eines der Objekte wirklich bearbeitet werden, so muss durch eine Drucktaste oder eine Menüfunktion der Ändern-Modus eingeschaltet werden.

Unabhängig davon, auf welchem Weg Sie zur Pflege der Domäne gelangen, tragen Sie im Feld WERTETABELLE die Tabelle YZ4B ein. Anschließend wird die Domäne aktiviert. Das Eintragen der Wertetabelle hat noch keinen direkten Einfluss auf die Prüfungen für die Domäne. Damit wird nur festgelegt, gegen welche Tabelle überhaupt geprüft werden soll. Die Prüfung selbst wird erst dann ausgeführt, wenn später für das zu prüfende Tabellenfeld eine Fremdschlüsselbeziehung definiert wird.

Das Zufügen einer Wertetabelle kann einer der Gründe dafür sein, eine aktive Domäne nochmals zu bearbeiten. Da die Domäne selbst in der Wertetabelle benutzt wird, ergibt sich eine zyklische Abhängigkeit. In die Domäne kann die Wertetabelle erst eingetragen werden, wenn die Tabelle YZ4B existiert, diese wiederum kann erst angelegt werden, wenn die Domäne YZ4BRA vorhanden ist.

Das Vorhandensein einer Prüftabelle kennzeichnet das System innerhalb des Pflegewerkzeugs für Tabellen in älteren Versionen des R/3-Systems mit einem Stern in der Spalte PRÜFTABELLE. Dieser Stern deutet darauf hin, dass für die zu Grunde liegende Domäne eine Wertetabelle existiert, aber noch kein Fremdschlüssel definiert wurde. Der Fremdschlüssel wird immer in der abhängigen Tabelle, hier also YZ4BT, angelegt. Dazu stellen Sie innerhalb der Tabellenpflege-Transaktion den Cursor in die Zeile für das Feld BRANCH und betätigen die Drucktaste FREMDSCHLÜSSEL innerhalb der Karteikarte. Da noch kein Fremdschlüssel existiert, erscheint ein Popup, mit dessen Hilfe ein Vorschlag für den Fremdschlüssel erstellt werden kann. Dieses Popup ist mit der Drucktaste JA zu bestätigen.

In einem zweiten Popup (Abbildung 5.11) bietet das System den Fremdschlüssel zur Bearbeitung an. Der Vorschlag wird vom System anhand der Schlüsselfelder und der Domäne des Feldes, für das der Fremdschlüssel erstellt werden soll, ermittelt. Das Flag PRÜFUNG ERWÜNSCHT innerhalb des Rahmens DYNPRO-PRÜFUNG ist zu markieren, falls es nicht schon automatisch gesetzt ist. In der Gruppe SEMANTISCHE EIGENSCHAFTEN wird das Auswahlfeld SCHLÜSSELFELDER EINER TEXTTABELLE markiert. Diese Auswahl ist entscheidend dafür, dass die später zu erzeugenden Pflegebausteine das Sprachenfeld der Tabelle YZ4BT automatisch auswerten. Die Fremdschlüsseldefinition kann nun mit der Drucktaste ÜBERNEHMEN in die Tabellendefinition eingefügt werden. Anstelle des Sterns in der Spalte PRÜFTABELLE erscheint nun der Name der Tabelle, gegen die das Feld geprüft werden soll. Anschließend wird die Tabelle aktiviert.

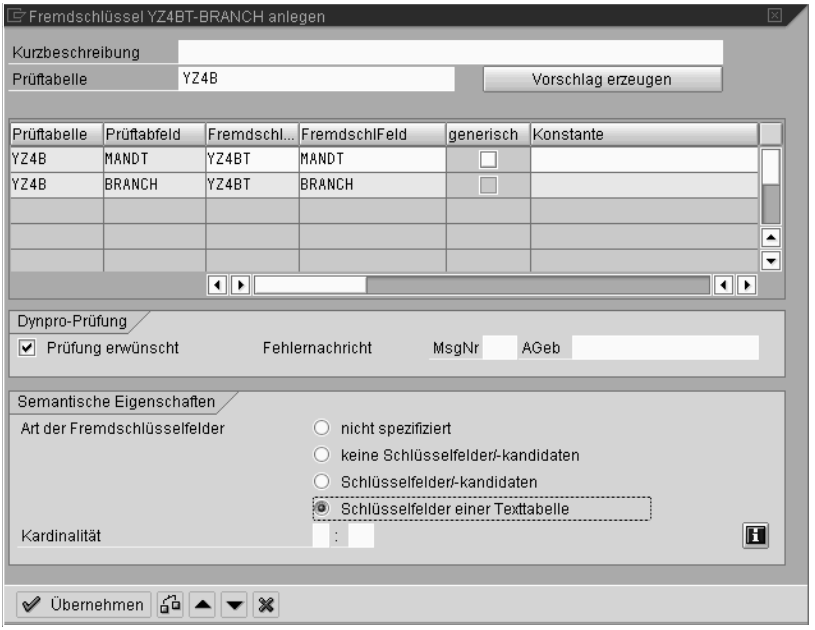

Abbildung 5.11 **© SAP AG**
Vorschlag für einen neuen Fremdschlüssel

Generierte Tabellenpflege

Mit der Definition des Fremdschlüssels sind die Voraussetzungen zum Anlegen des Pflegebausteins für Tabelle `YZ4B` geschaffen. Dieser Pflegebaustein ist die Grundlage für die Transaktionen `SM30`, `SM31` und `SM32`, mit denen Sie Tabelleninhalte pflegen können. Die Pflegebausteine werden vom Grundbild der Tabellenpflege aus erzeugt. Dazu dient die Menüfunktion HILFSMITTEL | TABELLENPFLEGEGENERATOR. Sie ist nur dann aktiv und damit ausführbar, wenn die Tabelle im Ändern-Modus bearbeitet wird. Der Tabellenpflegegenerator ist ab SAP-Release 3.0 ein relativ komplexes Werkzeug. Er erzeugt im Gegensatz zu den Werkzeugen früherer Releasestände keinen Modul-Pool mit einem einfachen Dynpro, sondern eine komplette Funktionsgruppe. Diese kann manuell nachbearbeitet und an spezielle Wüsche des Anwenders angepasst werden. Nach Änderungen an der Tabelle können ausgewählte Teile der Funktionsgruppe neu generiert werden, wobei manuelle Ergänzungen unter Umständen erhalten bleiben.

Um einen einfachen Pflegebaustein für die Tabelle `YZ4B` und die dazugehörigen Texte zu generieren, sind nur wenige Eingaben im Dynpro des Tabellenpflegegenerators (Abbildung 5.12) erforderlich.

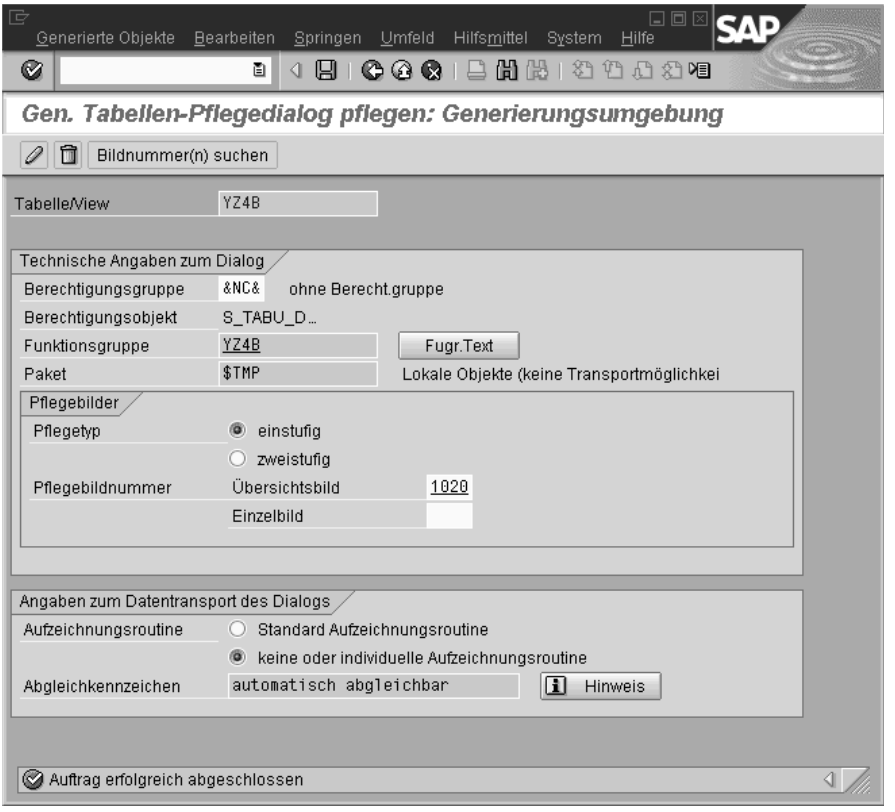

Abbildung 5.12 **© SAP AG**
Pflegedynpro des Tabellenpflegegenerators

Im Feld FUNKTIONSGRUPPE ist der Name einer Funktionsgruppe einzutragen, in der alle generierten Objekte zum Pflegen der Tabelle angelegt werden. Im Beispiel wird als Name YZ4B gewählt. Beim Feld PFLEGETYP ist die Auswahl EINSTUFIG zu markieren. Als Dynpronummer des Pflegebilds (Pflegedynpros) wird 1020 benutzt. Das einzige Feld, für das an dieser Stelle keine allgemein gültige Aussage zum Inhalt möglich ist, ist das Feld BERECHTIGUNGSGRUPPE. Einem Pflegebaustein zur Tabellenpflege muss eine Berechtigungsgruppe zugeordnet werden, die nur einem speziellen Anwenderkreis die Benutzung dieses Werkzeugs gestattet. Mit hoher Wahrscheinlichkeit wird in der Eingabehilfe für die Berechtigungsgruppe der Eintrag „&NC&“ angeboten, der für „Keine Berechtigungsgruppe“ steht. Falls dieser Eintrag nicht verfügbar ist oder später Probleme verursacht, müssen Sie sich bei Ihrem Systemverwalter nach einer passenden Berechtigungsgruppe erkundigen oder eine solche anlegen lassen.

Nach Eingabe aller Werte können Sie die Pflegebausteine mit der Drucktaste ANLEGEN oder der Menüfunktion GENERIERTE OBJEKTE | ANLEGEN erzeugen lassen. Eine Ausschrift in der Statuszeile des Bildschirmfensters informiert über die ordnungsgemäße Ausführung des Auftrages. Anschließend kann direkt aus dem Dynpro des Tabellenpflegegenerators heraus die soeben generierte Tabellenpflege aufgerufen werden. Dazu ist lediglich der Transaktionscode `/nSM31` im OK-Feld einzugeben. Dadurch wird die laufende Anwendung beendet und das Grundbild der Tabellenpflege aufgerufen (Abbildung 5.13).

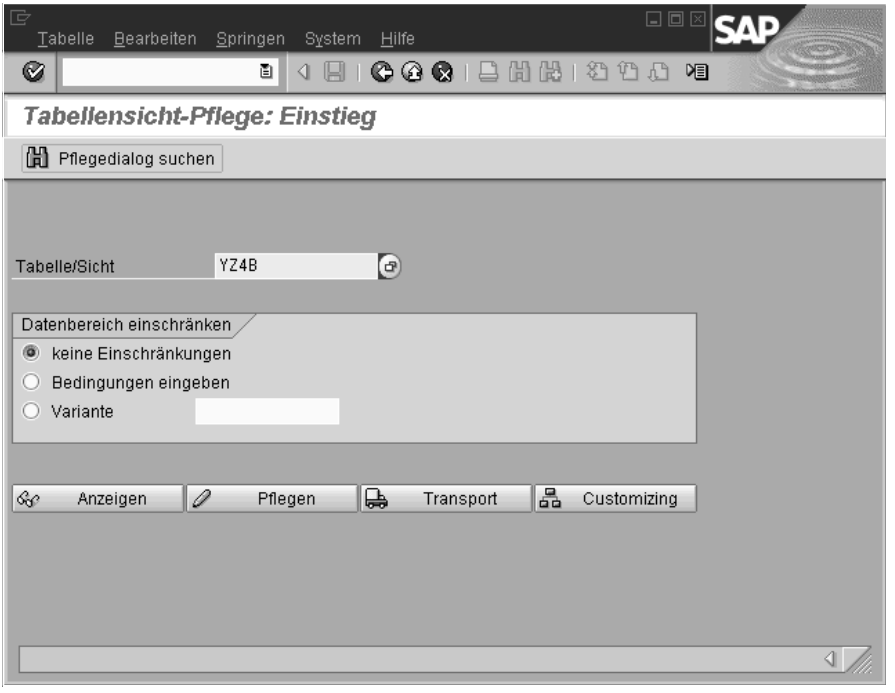

Abbildung 5.13 **© SAP AG**
Grundbild der Tabellenpflege

In diesem Dynpro ist lediglich der Name der zu pflegenden Tabelle einzutragen und per Drucktaste die Bearbeitungsart PFLEGEN zu wählen. Die Transaktion ruft dann die zur Bearbeitung erforderlichen Pflegefunktionen auf. Sofern diese nicht manuell verändert wurden, stellen sie den Inhalt der Tabelle in Listenform dar. Mit der Menüfunktion BEARBEITEN | NEUE EINTRÄGE oder der Drucktaste NEUE EINTRÄGE wird eine Liste bereitgestellt, in der neue Datensätze erfasst werden können. Abbildung 5.14 zeigt das Dynpro nach der Eingabe von zwei Datensätzen. Nach der Eingabe der Daten müssen die Änderungen abgespeichert werden.

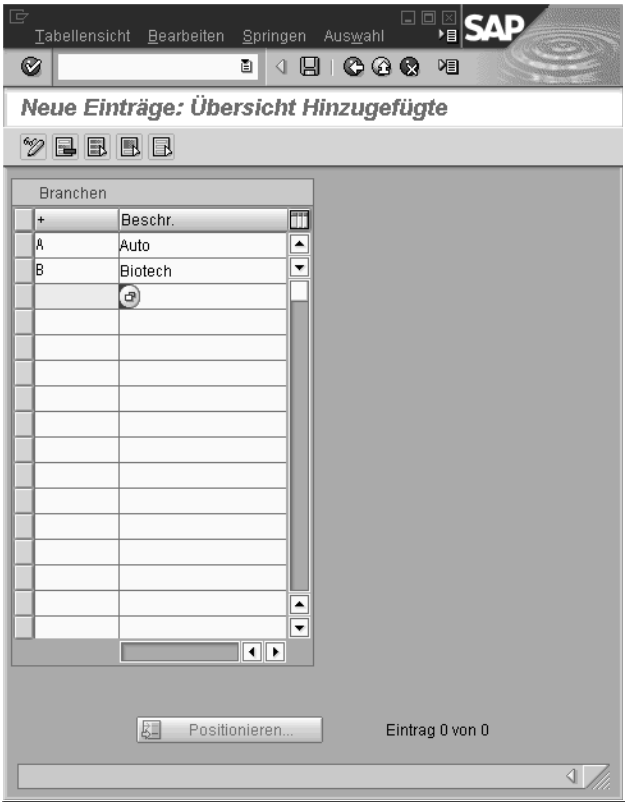

Abbildung 5.14 **© SAP AG**
Erfassen neuer Datensätze mit der Tabellenpflege

Im Pflegedynpro wird für das Eingabefeld BRANCHEN eine Eingabehilfe bereitgestellt. Nach Aufruf der Eingabehilfe (mit F4 oder dem Symbol rechts neben dem Eingabefeld) erscheint ein Popup, das auf den zulässigen Wertebereich verweist. Dazu werden die in der Domäne YZ4BRA definierten Festwerte ausgewertet.

Die Pflegetransaktionen erfordern in ihrem Startbild die Eingabe des Namens der zu bearbeitenden Tabelle. Da die Werkzeuge vor allem innerhalb des Customizing eingesetzt werden, bietet sich der Einsatz einer Parametertransaktion zum direkten Sprung in das Bearbeitungsbild der Tabelle an. Diese Transaktion kann in ein Menü eingetragen werden, wodurch ohne zusätzliche Eingaben eine ganz spezielle Tabelle gezielt gepflegt werden kann. Das folgende Beispiel beschreibt die Verwendung zusammen mit der SM30.

Zunächst ist eine neue Transaktion anzulegen, die der Übersichtlichkeit halber den Namen YZ4B erhält. Im nachfolgenden Popup (Abbildung 5.15) ist die Transaktionsart PARAMETERTRANSAKTION zu wählen.

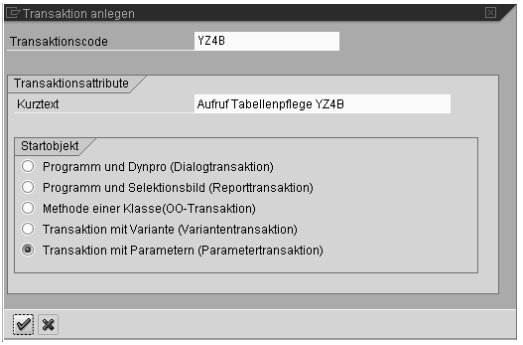

Abbildung 5.15 **© SAP AG**
Anlegen einer Parametertransaktion

Das darauf folgende Popup (Abbildung 5.16) ist etwas komplexer aufgebaut als das für die Bearbeitung einfacher Dialogtransaktionen.

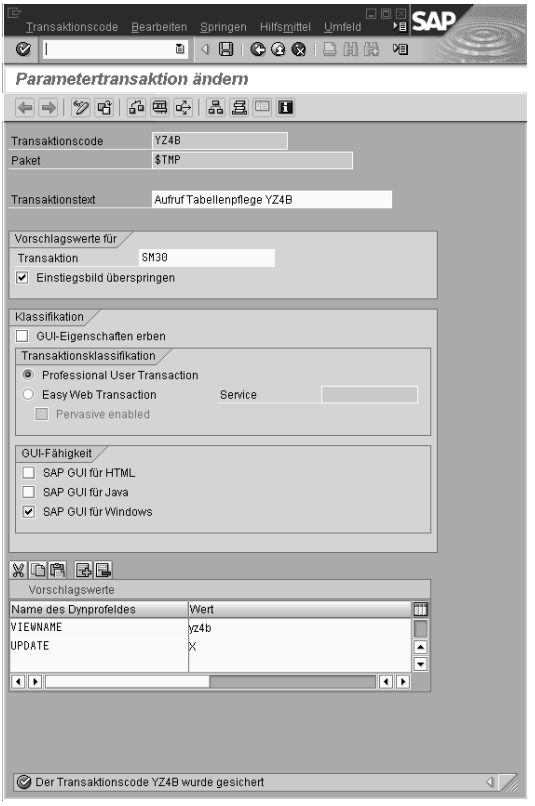

Abbildung 5.16 **© SAP AG**
Anlegen einer Parametertransaktion

Eine Parametertransaktion kann entweder eine Transaktion oder direkt ein Dynpro eines Modul-Pools aufrufen. Die letztgenannte Möglichkeit ist interessant, wenn für den Modul-Pool oder das Programm noch keine Transaktion existiert oder aber der Start über ein alternatives Dynpro erfolgen soll. Im vorliegenden Fall soll eine existierende Transaktion aufgerufen werden: Das Auswahlfeld TRANSAKTION wird markiert und der auszuführende Transaktionscode im daneben liegenden Eingabefeld eingetragen. Da direkt in die Tabellenpflege eingestiegen werden soll, ist das Eingangsbild der Transaktion SM30 zu überspringen. Dazu ist das Flag EINSTIEGSBILD ÜBERSPRINGEN unter dem Eingabefeld des Transaktionscodes zu markieren. Damit die gerufene Transaktion korrekt arbeiten kann, müssen ihr natürlich die Werte, die sonst im Eingangsbild erfasst werden müssen, auf andere Weise übergeben werden. Genau das ist der Sinn einer Parametertransaktion. Im unteren Drittel des Popups stehen Eingabefelder bereit, in denen die zu übergebenden Parameter eingetragen werden können. In der linken Spalte wird dazu der Name eines Dynprofelds des Einstiegsbilds der aufzurufenden Transaktion eingetragen, in der rechten Spalte der zu übergebende Wert. Nachdem die Parameter gemäß Abbildung 5.16 eingetragen wurden, kann die neue Transaktion gesichert und sofort ausgeführt werden.

Der Entwickler einer Parametertransaktion muss die aufzurufende Transaktion, zumindest aber den Aufbau des Startdynpros, genau kennen. Ohne diese Informationen ist die Erstellung einer korrekten Parametertransaktion nicht möglich. Die Informationen erhält man entweder durch eine entsprechende Dokumentation oder durch die Auswertung des Quelltextes der aufzurufenden Anwendung. Eine erste Hilfestellung bietet die Funktionstaste F1. Wenn der Kursor auf einem Dynprofeld steht, blendet der Druck auf diese Taste für die meisten Dynprofelder einen Hilfetext (Abbildung 5.17) ein.

Abbildung 5.17 **© SAP AG**
Hilfetext zum Dynprofeld

Innerhalb dieses Popups steht eine Drucktaste TECHNISCHE INFO zur Verfügung, über die Sie zu einem weiteren Popup (Abbildung 5.18) gelangen, dem Sie technische Angaben zum Dynprofeld, z.B. dessen Namen oder das dahinter liegende Tabellenfeld, entnehmen können.

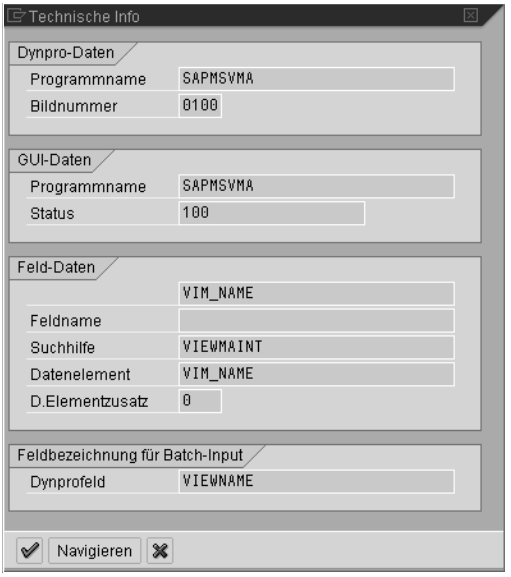

Abbildung 5.18 **© SAP AG**
Technische Informationen zum Dynprofeld

Speziell im Fall der Transaktion SM30 reicht die Analyse des Startdynpros allerdings nicht aus. Über eine Parametertransaktion können nur Felder des Startdynpros gefüllt, aber keine Drucktasten betätigt werden. Daher existieren im Startdynpro der Transaktion SM30 versteckte Eingabefelder, z.B. das Feld UPDATE. Vergleichen Sie dazu das Dynpro 100 des Programms SAPMSVMA. Zum Zeitpunkt PAI wertet die Ablauflogik nicht nur den ausgelösten Funktionscode, sondern auch den Zustand dieser Eingabefelder aus. Der Eintrag eines „X" im Feld Update wird von der Ablauflogik ebenso behandelt wie die Betätigung der Drucktaste PFLEGEN. Derartige Interna müssen dem Entwickler einer Parametertransaktion bekannt sein.

Include- und Append-Strukturen

Nicht in jedem Fall sollen oder können neue Felder direkt an die Originaltabelle angefügt werden. Zwei andere Möglichkeiten stellen die Include- und die Append-Strukturen dar. Mit Include-Strukturen werden vorhandene Strukturdefinitionen in andere Tabellen oder Dictionary-Strukturen eingefügt. Damit ist es möglich, umfangreichere Definitionen mehrfach zu verwenden. Dies erspart unter Umständen viel Schreibarbeit. Soll eine Teilstruktur an mehreren Stellen verwendet werden, so erleichtert eine Include-Struktur Änderungen. Diese müssen dann nur noch an einer Stelle gepflegt werden. Includes stehen unter allen Releaseständen zur Verfügung.

Während der Bearbeitung einer Tabelle wird eine vorhandene Struktur mit der Menüfunktion BEARBEITEN | INCLUDES | EINFÜGEN in die aktuelle Tabelle eingefügt. In der Tabelle erscheint als Feldname der Begriff .INCLUDE, anstelle des Datenelements steht der Name der eingebundenen Struktur. Der Name des einzufügenden Objekts wird in einem kleinen Popup (siehe Abbildung 5.19) erfasst, in dem zusätzlich noch ein Gruppenname und ein Namenssuffix eingetragen werden können.

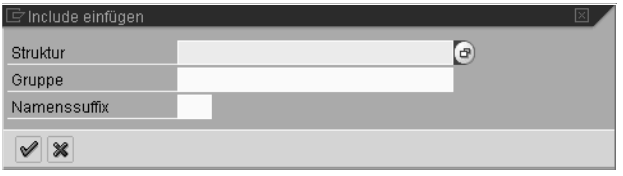

Abbildung 5.19 **© SAP AG**
Einfügen einer Include-Struktur in eine Tabellendefinition

Diese beiden Felder entsprechen in ihrer Wirkung den Zusätzen zum INCLUDE-Kommando (siehe Abschnitt 3.2). Der Gruppenname ermöglicht es, auf die eingefügte Substruktur als eigenständiges Teilobjekt zuzugreifen. Der Namenssuffix vermeidet Probleme, die durch identische Feldnamen in den verschiedenen Teilstrukturen entstehen könnten.

In der Regel kann das Feld für den Namenssuffix frei bleiben. Sollten sich allerdings beim Einfügen von Includes Probleme mit doppelten Feldnamen ergeben, so können diese durch ein Suffix behoben werden. Diese Zeichenkette wird an die Namen aller per Include eingefügten Felder angehängt. Dadurch entfallen Doppeldeutigkeiten, allerdings ändern sich Feldnamen, was bei der Programmierung von Anwendungen, die auf eine derartige Tabelle zugreifen, berücksichtigt werden muss.

Includes übernehmen eine weitere wichtige Aufgabe im Zusammenhang mit Kundenmodifikationen. Mitunter sollen Anwendungen an spezielle Bedürfnisse eines Kunden angepasst werden. Dazu ist es oft erforderlich, Tabellen mit zusätzlichen Feldern zu versehen. Wenn in das so modifizierte System neue SAP-Software eingespielt und dabei die beim Kunden modifizierte Tabelle von SAP nochmals ausgeliefert wird, werden die Änderungen des Kunden normalerweise überschrieben. Spezielle Pflegewerkzeuge ermöglichen während eines solchen Upgrades, Kundenmodifikationen einzupflegen und Datenverluste zu vermeiden. Diese Aufgabe kann erleichtert werden, wenn die vom Kunden anzufügenden Felder in ein Include verlagert werden. Während des Upgrades muss dann lediglich das Include – und nicht jedes Feld einzeln – neu eingebunden werden.

Ab Release 3.0 kann das eben beschriebene Problem noch eleganter mit den Append-Strukturen gelöst werden. Eine Append-Struktur ist eine Tabellenstruktur, die in ihren Verwaltungsdaten einen Verweis auf eine übergeordnete

Tabelle enthält. Beim Aktivieren der übergeordneten Tabelle, die selbst keinen Verweis auf die Append-Strukturen enthalten muss, werden alle Append-Strukturen automatisch der übergeordneten Tabelle zugeordnet. Da sich die untergeordneten Strukturen automatisch an die übergeordnete Tabelle anhängen, selbst wenn deren Struktur durch ein SAP-Upgrade geändert wurde, entfällt die Notwendigkeit eines manuellen Abgleichs. Append-Strukturen werden ähnlich wie Includes innerhalb der Tabellenpflege mit der Menüfunktion SPRINGEN | APPEND-STRUKTUREN erzeugt.

5.1.4 Tabellen für Datencluster

Wie bereits erwähnt, müssen Tabellen, in denen Datencluster abgelegt werden sollen, einen speziellen Aufbau besitzen. Sie werden zwar als normale transparente Tabellen angelegt und können mit den entsprechenden Kommandos (z.B. `SELECT`) bearbeitet werden, ihre eigentliche Leistungsfähigkeit entfalten sie aber erst im Zusammenhang mit den Kommandos `EXPORT` und `IMPORT`. Diese Tabellen und das Zusammenspiel mit den erwähnten Kommandos sind ein gutes Beispiel für die enge Verflechtung der verschiedenen Objekte innerhalb der R/3-Entwicklungsumgebung.

Tabelle 5.4 zeigt den Aufbau einer solchen Tabelle. Falls die Tabelle mandantenabhängig sein soll, ist das erste Feld das Mandentenfeld `MANDT`. Das zweite Feld ist obligatorisch. Es muss den Namen `RELID` tragen sowie den Datentyp `CHAR` und die Länge 2 besitzen. In diesem Feld wird die Kennung für das so genannte Gebiet abgelegt. Es ist ein Schlüsselfeld. An dieses Feld schließen sich beliebige weitere Schlüsselfelder an, die alle vom Typ `CHAR` sein müssen. Durch diese Felder wird der eigentliche Schlüssel der Datencluster gebildet. Die Kommandos `IMPORT` bzw. `EXPORT` erwarten als Schlüssel allerdings nur ein einziges Feld. Der Inhalt dieses Feldes wird dann zeichenweise in die Schlüsselfelder übertragen. Das letzte Schlüsselfeld ist wiederum vorgegeben. Es bekommt den Namen `SRTF2` und muss ein Integer-Feld mit der Länge 4 sein.

Die Schlüsselfelder werden durch das Kommando `EXPORT` automatisch versorgt. Alle Schlüsselfelder außer dem Mandantenfeld und dem Feld `SRTF2` werden aus den bei `EXPORT` oder `IMPORT` angegebenen Werten gebildet. Das Feld `SRTF2` wird immer automatisch versorgt.

An die Schlüsselfelder schließen sich beliebige weitere Felder an. Diese Felder können Zusatzinformationen zu den im Datencluster gespeicherten Daten enthalten. Sie müssen mit den Standardkommandos für die Arbeit mit Tabellen (`SELECT`, `UPDATE`…) bearbeitet werden.

Den Abschluss der Tabelle bilden die beiden Felder `CLUSTR` und `CLUSTD`. Sie besitzen den Datentyp `INT2` bzw. `LRAW`. Das Feld `CLUSTD` nimmt die eigentlichen Daten auf. Da die Länge dieses Feldes natürlich begrenzt ist, werden die Daten bei Bedarf in mehreren aufeinander folgenden Datensätzen abgespeichert bzw.

aus mehreren Datensätzen gelesen. Dabei übernimmt das Feld SRTF2 die Aufgabe eines Zählers.

Tabellenfeld	Schlüssel	Typ	wird bei EXPORT/IMPORT versorgt durch:
MANDT	X	MANDT	Automatisches Mandantenhandling oder explizite Angabe mit CLIENT.
RELID	X	CHAR2	Gebiet
Beliebige Schlüsselfelder	X	CHAR	Schlüssel (als einzelnes CHAR-Feld ohne innere Struktur).
SRTF2	X	INT4	Wird automatisch gefüllt.
Beliebige Tabellenfelder			Werden nicht durch IMPORT/EXPORT behandelt.
CLUSTR		INT2	Interne Verwaltungsinformation; wird automatisch gefüllt.
CLUSTD		LRAW	Datenfeld für Clusterdaten; wird automatisch gefüllt.

Tabelle 5.4
Aufbau einer Tabelle für Datencluster

Falls in einer Tabelle für Datencluster zusätzliche Informationen in optionalen Feldern abgelegt werden, ist dies im allgemeinen nur einmal für jeden Datencluster notwendig, unabhängig davon, wie viele Datensätze für dessen Speicherung wirklich benötigt werden. In diesem Fall muss das Schlüsselfeld SRTF2 natürlich mit einem gültigen Wert versorgt werden. Dies wird in der Praxis immer der Wert 0 sein, da für jeden Datencluster mindestens ein Datensatz geschrieben wird. Ein Beispiel zur Struktur von Tabellen für Datencluster und zur Bearbeitung dieser Tabellen finden Sie im Abschnitt 3.10.3.

Indizes

Suchvorgänge in Tabellen können durch geeignete Indizierung erheblich beschleunigt werden. Ein Index enthält ausgewählte Felder einer Tabelle des Datenbanksystems, wobei die Datensätze stets sortiert werden. Die Suche nach einem Datensatz kann nun mittels geeigneter Verfahren, zum Beispiel der binären Suche, sehr schnell erfolgen. Im Index verweist ein Zeiger auf die Stelle der Datenbanktabelle, an welcher der zum Indexbegriff gehörende Datensatz gefunden werden kann. Indizes werden durch das Datenbanksystem automatisch benutzt. Beim Zugriff auf eine Tabelle, z. B. mit SELECT . . . WHERE, sucht das Daten-

banksystem einen zur WHERE-Klausel passenden Index. Falls ein solcher existiert, wird er benutzt, falls nicht, findet eine sequenzielle Suche in der Datenbanktabelle statt. Die Indizes werden beim Einfügen oder Löschen von Datensätzen automatisch durch das Datenbanksystem aktualisiert.

Zu jeder Datenbanktabelle legt das System einen so genannten *Primärindex* an, der alle Schlüsselfelder enthält. Darüber hinaus kann sowohl der Programmierer als auch der Anwender des SAP-Systems zusätzliche *Sekundärindizes* erzeugen. Dies erfolgt wieder vom Hauptbild der Tabellenpflege aus mit der Drucktaste INDIZES... oder der Menüfunktion SPRINGEN | INDIZES. Ein Index wird ab SAP-Release 3.0 durch den Tabellennamen und eine dreistellige Kennung identifiziert. Bei der Pflege eines Index aus der Tabellenpflege heraus steht der Name der Tabelle natürlich fest. In einem kleinen Popup (Abbildung 5.20) ist lediglich die dreistellige Indexkennung einzutragen.

Abbildung 5.20 **© SAP AG**
Auswahl des zu bearbeitenden Index

Die Kennung des Index dient lediglich zur Unterscheidung mehrerer Indizes innerhalb der Pflegewerkzeuge. Da Indizes vom Datenbanksystem automatisch benutzt werden, ist es nicht erforderlich bzw. nicht möglich, in Programmen gezielt auf einen Index zu verweisen.

Nach Eingabe einer Kennung und Bestätigen der Eingabe mit der Datenfreigabetaste kann der Index bearbeitet werden. Abbildung 5.21 zeigt das entsprechende Dynpro.

Selbstverständlich ist zunächst eine Kurzbeschreibung einzutragen. Ein Flag UNIQUE-INDEX legt fest, ob Einträge in der Indextabelle eindeutig sein sollen. Falls dieses Flag gesetzt wird, prüft die Datenbank nicht nur die Eindeutigkeit der Schlüsselfelder der Tabelle, sondern zusätzlich noch die Eindeutigkeit bezüglich der Indexdatei. Das bedeutet, dass ein Datensatz möglicherweise nicht in eine Tabelle aufgenommen werden kann, obwohl dies gemäß der Werte in seinen Schlüsselfeldern möglich sein müsste. Für Sekundärindizes ist diese Einstellung selten sinnvoll, sondern eher gefährlich, da durch eine derartige zusätzliche Datenprüfung die Funktionsfähigkeit vorhandener Anwendungen beeinträchtigt werden kann.

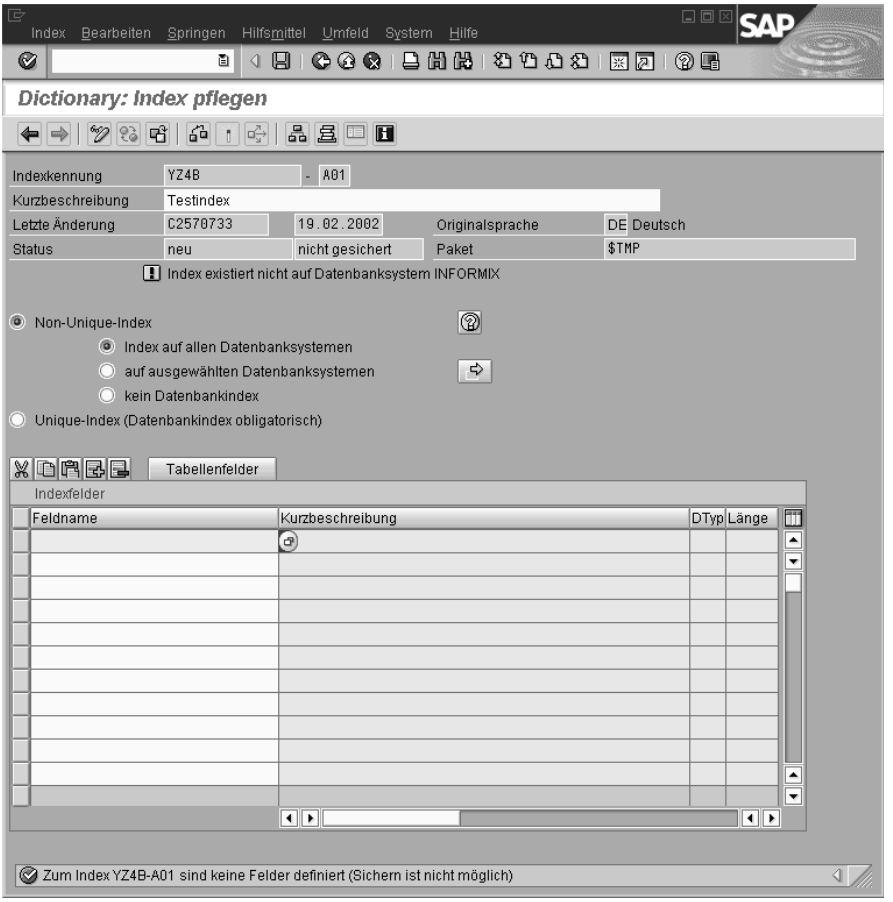

Abbildung 5.21 **© SAP AG**
Pflege eines Index

Im Table View-Bereich des Dynpros werden nun die Tabellenfelder eingetragen, die zur Indexbildung herangezogen werden sollen. Die Eingabehilfe-Taste an jedem Eingabefeld für den Feldnamen ruft eine Eingabehilfe auf, in der alle zu übernehmenden Felder angeboten und per Mausklick in die Indexdefinition übernommen werden können. Die Reihenfolge der Felder im Index kann von erheblicher Bedeutung für dessen Funktion sein.

Nach dem Eintragen der Felder muss der Index aktiviert werden. Während der Aktivierung wird er auch auf der Datenbank erzeugt. Mit der erfolgreichen Aktivierung ist die Bearbeitung eines Index abgeschlossen.

Die Sekundärindizes sollten speziell für laufzeitkritische Suchvorgänge maßgeschneidert werden. Das bedeutet, dass die Reihenfolge der Felder in einer `WHERE`-

Klausel und die Indexfelder aufeinander abgestimmt sein müssen. Das erste Indexfeld sollte bereits eine weitgehende Selektion ermöglichen. Es sollte also ein Feld sein, in dem sich der Inhalt häufig ändert. In der WHERE-Klausel sollten die Felder in der Reihenfolge aufgeführt werden, in der sie in der Indexdefinition stehen. Ein Index kann nur soweit ausgewertet werden, wie gültige Felder (aus Sicht des Index) in der WHERE-Klausel stehen.

Zur Zusammenarbeit von Index und WHERE-Klausel folgt ein Beispiel. Vorgegeben sei eine Tabelle mit den Feldern KUNNR (Schlüsselfeld), NAME, VORNAME, STRASSE, PLZ und ORT. Falls nun häufig anhand des Namens und des Ortes gesucht wird, sollte ein Sekundärindex über NAME und ORT gebildet werden. Die WHERE-Klausel enthält dann die Felder in exakt dieser Reihenfolge:

```
...
WHERE NAME = ...
  AND ORT = ...
...
```

Eine so aufgebaute WHERE-Klausel würde über den Index sehr effektiv die Datensätze mit dem gewünschten Namen und dann bei mehreren gefundenen Einträgen ebenfalls über den Index den gesuchten Ort ermitteln. Wäre die Reihenfolge der Felder in der WHERE-Klausel vertauscht, so könnte höchstwahrscheinlich kein Index benutzt werden. Falls hingegen der Index über die drei Felder NAME, VORNAME und ORT gebildet wird, so kann bei der Suche nur der Index bis zum Feld NAME verwendet werden. In den so identifizierten Datensätzen ist dann eine sequenzielle Suche nach dem gewünschten Ort notwendig, da für das Indexfeld VORNAME kein Wert in der WHERE-Klausel verfügbar ist.

Datenbank-Utility

In vielen Pflegewerkzeugen ist das so genannte *Datenbank-Utility* verfügbar. Es ist einheitlich über die Menüfunktion HILFSMITTEL | DATENBANK-UTILITY oder den Transaktionscode SE14 erreichbar. Die Werkzeuge des Dictionary bearbeiten zunächst nur die logischen Definitionen der verschiedenen Objekte. Diese müssen in vielen Fällen aber auch noch physisch in der Datenbank erzeugt werden. Beispielsweise muss nach der Definition der Struktur einer Tabelle diese auch noch als reale Tabelle des Datenbanksystems erzeugt werden, damit wirklich Daten gespeichert werden können. Das Dictionary an sich stellt ja lediglich die Beschreibung der Tabelle aus Sicht des SAP-Systems zur Verfügung. Das Erzeugen oder Löschen von Objekten in der Datenbank erfolgt mit dem Datenbank-Utility. Es ist inzwischen aber nur noch selten notwendig, dieses Werkzeug manuell aufzurufen, da es üblicherweise beim Aktivieren einer Tabelle automatisch gestartet wird und seine Arbeit im Hintergrund verrichtet. Gründe für den manuellen Aufruf können beispielsweise Probleme nach einem Import oder einem Upgrade sein.

Beim Aufruf aus den Dictionary-Werkzeugen heraus werden einige Parameter, z. B. der Name des Objekts, vorbelegt. Der Anwender kann dann im Datenbank-Utility verschiedene Aktivitäten (Anlegen, Löschen, Umsetzen) und unterschiedliche Ausführungsmodi (z. B. Online oder Batch-Verarbeitung) wählen. Einige der Pflegefunktionen, vor allem die zum Anlegen von neuen Objekten, rufen das Datenbank-Utility automatisch auf. Falls die manuelle Verwendung des Datenbank-Utilitys erforderlich wird, weisen die Aktivierungsprotokolle der diversen Elemente darauf hin. Besonders bedeutsam ist die Funktion UMSETZEN, die für Tabellen aufgerufen werden muss, deren Struktur geändert wurde. Beim Umsetzvorgang werden gleichzeitig die alten Daten in die neue Tabellenstruktur übernommen.

Falls Dictionary-Objekte, die auf der Datenbank existieren, gelöscht werden sollen, ist dies erst nach Löschen des Objekts von der Datenbank möglich. Dazu ist in früheren Versionen das Datenbank-Utility manuell aufzurufen. In der aktuellen Version 4.6 besteht diese Notwendigkeit nicht mehr.

Auslieferungsklasse

Die Tabellen im SAP-System enthalten völlig unterschiedliche Informationen. Einige nehmen die bei der täglichen Arbeit anfallenden Daten des Anwenders, die so genannten Bewegungsdaten auf, mit anderen wird das R/3-System im Customizing an die realen Erfordernisse des Anwenders angepasst. Weitere Tabellen enthalten Daten, die das SAP-System für seine Arbeit unbedingt benötigt, z. B. Berechtigungen, Umrechnungsfaktoren für Währungen etc. Zum Teil werden sogar einige Anwendungen direkt durch die Einträge in einigen Tabellen gesteuert. Ebenso vielfältig wie die Aufgaben der Tabellen sind auch die Anlässe und Methoden, ihren Inhalt zu modifizieren. Bei der Installation der SAP-Software werden viele der Tabellen mit Vorgabewerten gefüllt. Spätere Auslieferungen von Korrekturen oder neuen Releaseständen oder die Kopie von Mandanten müssen diese Inhalte möglicherweise überschreiben. Dabei muss aber verhindert werden, dass anwenderspezifische Einstellungen verändert oder gelöscht werden. Ebenso müssen Veränderungen von Systemtabellen durch den Anwender unterbunden werden, weil dadurch die Funktionsfähigkeit des Gesamtsystems infrage gestellt werden könnte. Den einzelnen Tabellen wird daher ein spezielles Attribut, die so genannte *Auslieferungsklasse*, zugeordnet. Dieses Attribut besagt, durch wen der Inhalt dieser Tabelle aktualisiert werden darf und in welcher Form sie durch SAP ausgeliefert wird. Tabelle 5.5 zeigt die verschiedenen Auslieferungsklassen.

Anwendungstabellen nehmen die Daten auf, die durch die diversen betriebswirtschaftlichen Anwendungen erzeugt werden. Derartige Tabellen werden durch SAP immer leer ausgeliefert. Nur der Kunde fügt Inhalte ein. Die Customizing-Tabellen dienen zur Anpassung des Systems an Kundenwünsche. Für derartige Tabellen werden bei der Installation des R/3-Systems Vorgaben ausgeliefert, später erfolgt die Pflege durch den Kunden.

Auslieferungsklasse	Beschreibung
A	Anwendungstabelle
C	Customizing-Tabelle
G	Geschützte Tabelle
L	Temporäre Daten
E	Allgemeine Systemtabelle
S	SAP-Systemtabelle
W	Tabelle für Systembetrieb

Tabelle 5.5
Auslieferungsklassen

Diese Tabelleninhalte werden durch SAP-Upgrades in der Regel nie geändert. Die geschützten Tabellen ähneln den Customizing-Tabellen, allerdings kann ein SAP-Upgrade Werte hinzufügen, aber keine vorhandenen Werte ändern. Die Tabellen für temporäre Daten ähneln in ihrem Verhalten den Anwendungstabellen. Sie werden leer ausgeliefert und durch SAP nie mit Werten gefüllt. Der Unterschied zu den Anwendungstabellen besteht darin, dass beim Kopieren der Daten eines Mandanten die Inhalte von Anwendungstabellen mitkopiert werden können, nicht aber die von temporären Tabellen. Die allgemeinen Systemtabellen besitzen einen Namensraum für Kunden. Dieser Namensraum wird durch einen SAP-Upgrade nicht beeinflusst, Einträge im SAP-Namensraum können aber geändert werden. Die SAP-Systemtabellen hingegen erlauben nur eine Pflege durch SAP. Sie werden bereits in gefülltem Zustand ausgeliefert. Während die Systemtabellen der Klassen S und E Informationen mit eher statischem Charakter aufnehmen, enthalten die Tabellen für den Systembetrieb (Klasse W) Daten, die durch diverse Transaktionen indirekt geändert werden. Sie enthalten Daten, die das R/3-System zur Aufrechterhaltung seiner Funktionalität benötigt, beispielsweise die Beschreibungen von Dictionary-Elementen oder die Quelltexte von Programmen. Die Inhalte dieser Tabellen unterliegen keinem besonderen Schutz.

5.1.5 Strukturen

Bei Dictionary-Strukturen handelt es sich im Prinzip um die Deklaration einer Feldleiste im Dictionary. Diese Struktur hat wie eine Tabelle einen Namen und eine Feldliste. Allerdings wird für eine Struktur keine Datenbanktabelle erzeugt. Das Pflegewerkzeug ähnelt daher weitgehend dem zur Bearbeitung von Tabellen, wobei aber einige datenbankbezogene Funktionen fehlen.

Im Zusammenhang mit der Verschmelzung von Dictionary-Elementen und globalen Datentypdeklarationen werden Strukturen in den neueren Versionen der R/3-Software als Bestandteile der Datentypen geführt. Im Einstiegsbild der Transaktion zur Dictionary-Pflege (SE11) existiert daher nicht mehr das Auswahlfeld STRUKTUR, sondern stattdessen das Feld DATENTYP. Erst wenn ein neuer Datentyp angelegt wird, muss in einem kleinen Popup (siehe Abbildung 5.22) genauer spezifiziert werden, welche Art von Datentyp angelegt werden soll.

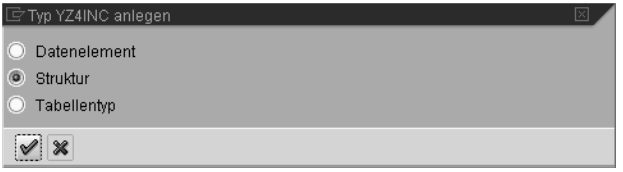

Abbildung 5.22 **© SAP AG**
Auswahl der Datentypart

Die Pflegeoberfläche für Strukturen ähnelt der von Datenbanktabellen. Bezüglich der Felder und deren Eigenschaften (z. B. Eingabehilfe und Prüfung) existieren nahezu identische Möglichkeiten zur Pflege. Lediglich die Einstellungen, die auf die Speicherung einer Tabelle in einem Datenbanksystem ausgerichtet sind (Primärschlüssel, Größenkategorie, technische Einstellungen), fehlen natürlich.

Strukturen können auf zwei unterschiedliche Arten benutzt werden. Sie können in einem Programm wie eine Feldleiste benutzt werden, es ist also möglich, ihnen Daten zuzuweisen. Dazu sind sie im Programm ähnlich wie Tabellen mit dem Schlüsselwort DATA oder TABLES zu deklarieren. Strukturen vereinen in sich den Vorteil einer Feldleiste (gemeinsame Behandlung mehrerer logisch zusammengehöriger Datenfelder) und eines Dictionary-Elements (Typisierung, Fremdschlüsselprüfungen, Feldbezeichner, einfache Übernahme in Dynpros).

Die zweite Verwendungsmöglichkeit der Strukturen besteht als Datentyp. Sie können eine Struktur benutzen, um in der DATA-Anweisung die Eigenschaften eines eigenen Datenobjekts mittels des TYPE-Zusatzes zu beschreiben. Falls eine Struktur nur in ihrer Eigenschaft als Datentyp benutzt wird, muss sie nicht mit TABLES deklariert werden.

Wenn in einem Dynpro Felder aus mehreren Tabellen gepflegt werden, erleichtert eine Struktur, die alle zu pflegenden Felder enthält, gegenüber der direkten Verwendung der Tabellen die Programmierung erheblich. Bei der Bearbeitung des Dynpros können alle erforderlichen Felder in einem Arbeitsgang in den Fullscreen des Dynpros eingefügt werden. Innerhalb der Anwendung erleichtert eine solche Struktur den Umgang mit den Daten. Zunächst ist das Programm übersichtlicher, da immer mit einer einzigen Struktur gearbeitet wird, um auf die Dynprodaten zuzugreifen. Die Übergabe der Daten an Prüfroutinen oder Verbucher ist ebenfalls überschaubarer, da nur ein Parameter benutzt werden muss.

5.1.6 Tabellentypen

Beim Anlegen von Datenelementen wurde bereits darauf hingewiesen, dass neben den einfachen auch strukturierte und Tabellentypen im Dictionary angelegt werden können. Innerhalb des Pflegewerkzeugs für Tabellentypen müssen die Struktur, die Zugriffsvariante sowie ggf. der Schlüssel für die interne Tabelle definiert werden. Die Angaben entsprechen denen, die auch bei der manuellen Definition eines Tabellentyps mit der Anweisung TYPES gemacht werden müssen.

Bei der Pflegetransaktion (siehe Abbildung 5.23) ist neben der obligatorischen Kurzbeschreibung zunächst die Struktur des Tabellentyps zu definieren. Dies erfolgt auf der Registerkarte ZEILENTYP. Dort können Sie die Struktur entweder durch Angabe eines existierenden einfachen oder strukturierten Typs, durch Direktangabe eines der eingebauten elementaren Typen oder durch Angabe eines Referenztyps definieren. Die letztgenannte Möglichkeit ist interessant, um Referenzen auf Objekte in internen Tabellen speichern zu können.

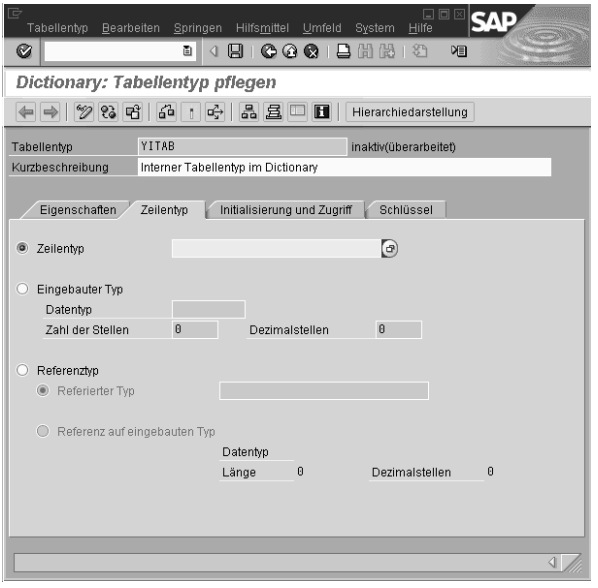

Abbildung 5.23 **© SAP AG**
Struktur des Tabellentyps festlegen

Auf der Registerkarte INITIALISIERUNG UND ZUGRIFF (Abbildung 5.24) legen Sie die Zugriffsvariante fest. Wie bereits im Abschnitt 3.2.7. beschrieben, existieren dazu die drei echten Varianten Standard-Tabelle, sortierte Tabelle und Hash-Tabelle. Außerdem können die beiden generischen Varianten „Index-Tabelle" und „nicht spezifiziert" gewählt werden. In diesem Fall kann natürlich kein Zeilentyp gepflegt werden.

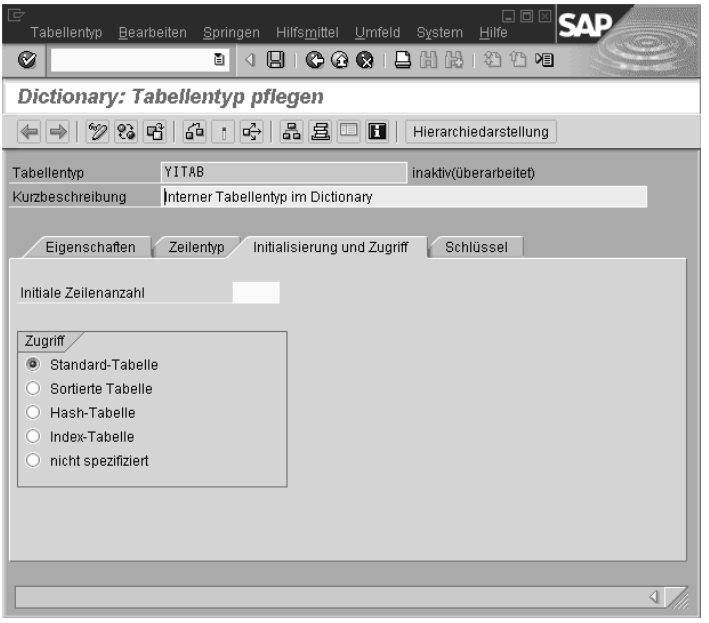

Abbildung 5.24 **© SAP AG**
Tabellenart auswählen

Die letzte Registerkarte SCHLÜSSEL (Abbildung 5.25) dient zur Definition eines Schlüssels. Innerhalb der Untergruppe SCHLÜSSELDEFINITION wird zunächst festgelegt, wie der Schlüssel aufgebaut sein soll. Ein STANDARDSCHLÜSSEL wird in Abhängigkeit vom Zeilentyp gebildet. Er besteht entweder aus allen zeichenartigen Feldern (bei herkömmlich strukturierten Zeilentypen) oder der gesamten Zeile (z. B. bei Objektreferenzen). Ein Schlüssel vom Typ ZEILENTYP erstreckt sich stets auf den gesamten Datensatz. Bei der Einstellung SCHLÜSSELKOMPONENTEN können die Bestandteile des Schlüssels manuell gewählt werden.

Im Feld SCHLÜSSELART ist schließlich noch festzulegen, ob der Schlüssel eindeutig sein soll oder nicht.

Nach Wahl dieser Einstellung wird die Schaltfläche KOMPONENTEN AUSWÄHLEN bedienbar. Diese Schaltfläche blendet ein Popup ein (Abbildung 5.26), in dem die Bestandteile des zuvor festgelegten Zeilentyps ausgewählt werden können.

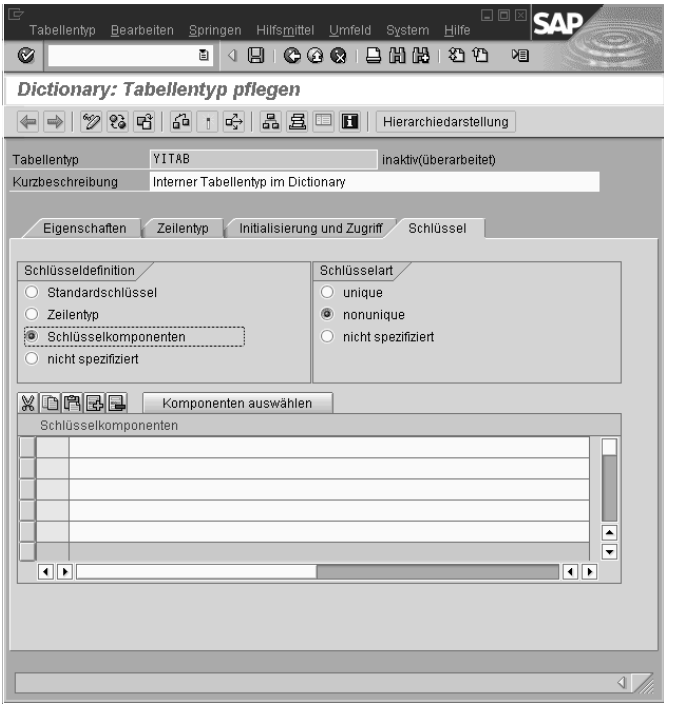

Abbildung 5.25 **© SAP AG**
Schlüsseleigenschaften spezifizieren

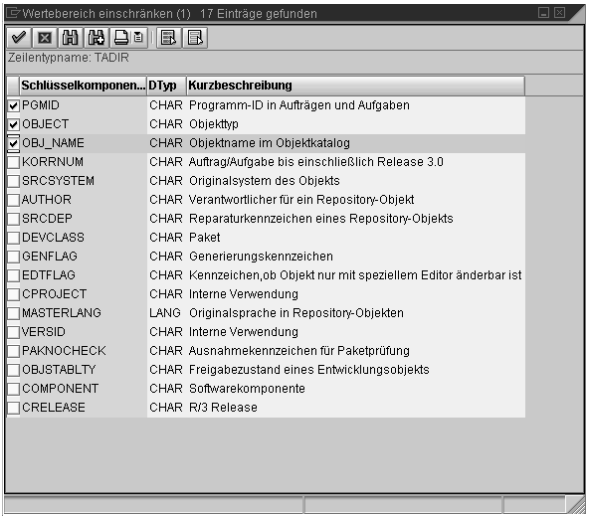

Abbildung 5.26 **© SAP AG**
Schlüsselkomponenten auswählen

5.1.7 Fremdschlüsselbeziehungen

In einer Datenbank bestehen Abhängigkeiten zwischen den verschiedenen Tabellen. Das relationale Datenbankkonzept sieht ja vor, Informationen möglichst redundanzfrei in Tabellen zu speichern und die Tabellen über gemeinsame Schlüssel miteinander zu verknüpfen. Diese Zuordnung erfolgt beim Entwurf einer Datenbank zunächst auf rein logischer Ebene. Das Data Dictionary des SAP-Systems ermöglicht darüber hinaus auch die Realisierung physischer Abhängigkeiten über so genannte Fremdschlüssel. Diese Fremdschlüssel können, abhängig von ihrer konkreten Definition, mehrere Aufgaben erfüllen:

- Dokumentation der Tabellenbeziehungen und damit des Datenmodells
- Bereitstellung von Informationen zur Tabellenverknüpfung für andere Werkzeuge
- Ausführung von Wertprüfungen in Dynpros

Fremdschlüssel werden in einer Tabelle für jeweils ein Tabellenfeld definiert. Sie geben an, wie der Inhalt dieses Feldes vom Inhalt eines anderen Feldes in einer übergeordneten Tabelle (auch Prüftabelle genannt) abhängt. Die Tabelle, die den Fremdschlüssel enthält, wird auch als Fremdschlüsseltabelle und das Feld, für das der Fremdschlüssel angelegt wurde, also das zu prüfende Feld, als Prüffeld bezeichnet. Ein Fremdschlüssel kann neben dem Prüffeld weitere Felder enthalten, die Fremdschlüsselfelder genannt werden. Die Fremdschlüsselfelder sind in der Prüftabelle Schlüsselfelder.

Ein einfaches Beispiel für Fremdschlüssel bieten die bereits definierten Tabellen `YZ4B` und `YZ4BT`. In beiden Tabellen ist ein Feld `BRANCH` enthalten. Da in der Tabelle `YZ4BT` nur ergänzende Texte zu den Datensätzen aus `YZ4B` stehen sollen, dürfen im Feld `YZ4BT-BRANCH` nur Werte eingetragen werden, die im Feld `YZ4B-BRANCH` enthalten sind. Aus diesem Grund wurde in der Tabelle `YZ4BT` ein Fremdschlüssel definiert. Die Tabelle `YZ4BT` ist damit die Fremdschlüsseltabelle, das Feld `YZ4BT-BRANCH` ist das Prüffeld. Die Tabelle `YZ4B` ist die Prüftabelle für den Fremdschlüssel.

Damit ein Fremdschlüssel seine Aufgaben erfüllen kann, muss er einige wichtige Informationen besitzen:

- Welche Tabelle ist die Prüftabelle?
- Gegen welches Feld in der Prüftabelle soll geprüft werden?
- Sollen zusätzliche Felder berücksichtigt werden, um die Wertemenge aus der Prüftabelle einzuschränken?
- Welche Abhängigkeit von der Prüftabelle soll bestehen?

Die erforderlichen Informationen werden beim Anlegen eines Fremdschlüssels zum Teil durch das System ermittelt, einige weitere Angaben müssen manuell festgelegt werden. Bearbeitet werden Fremdschlüssel in einem Popup, das bereits in Abbildung 5.11 zu sehen war. Zu den einzelnen Aspekten der Fremdschlüsseldefinition sind noch einige Erläuterungen erforderlich.

Prüftabelle und Prüftabellenfeld

Wenn ein Feld geprüft werden soll, so ist eindeutig festzulegen, gegen welches Feld aus welcher Tabelle geprüft werden soll. Diese beiden Eigenschaften eines Fremdschlüssels können nicht beliebig gewählt werden. Sie werden durch die Domäne, auf der das zu prüfende Feld beruht, bestimmt. In den Eigenschaften einer Domäne kann eine Wertetabelle eingetragen werden. Diese Wertetabelle wird bei der Definition eines Fremdschlüssels als Prüftabelle vorgeschlagen. In der Prüftabelle wird vom System ein Feld aus dieser Tabelle gesucht, das auf derselben Domäne beruht. Gegen dieses Feld wird dann geprüft.

Der Vorschlag für die Prüftabelle kann überschrieben werden. Dabei darf aber nur eine Tabelle gewählt werden, die direkt oder indirekt über Fremdschlüsselbeziehungen mit der eigentlichen Wertetabelle der Domäne verbunden ist. Der neue Tabellenname wird im Fremdschlüssel-Popup im entsprechenden Eingabefeld eingetragen. Nach Betätigen der Datenfreigabetaste werden die Fremdschlüsselfelder und die Zuordnungsvorschläge aktualisiert. In diesem Fall unterscheiden sich Werte- und Prüftabelle.

Schlüsselfelder

Bei der Definition eines Fremdschlüssels versucht das System, für alle Schlüsselfelder der Prüftabelle korrespondierende Felder in der Fremdschlüsseltabelle (der untergeordneten Tabelle) zu finden. Dazu werden wiederum die Domänen der Schlüsselfelder der Prüftabelle ausgewertet. Felder der Fremdschlüsseltabelle, die auf denselben Domänen beruhen, werden in den Fremdschlüssel aufgenommen. Falls in der Fremdschlüsseltabelle keine geeigneten Felder vorhanden sind, weist das System darauf hin, dass der Fremdschlüssel nicht vollständig angegeben werden kann.

Wenn später der Inhalt eines Feldes einer Fremdschlüsseltabelle geprüft wird, dann entnimmt das System den Fremdschlüsselfeldern dieser Tabelle die aktuellen Werte und bildet daraus einen Schlüssel für die Prüftabelle. Daraufhin werden nur die über den Schlüssel identifizierten Datensätze der Prüftabelle zur Prüfung herangezogen.

Diese weitgehende Einschränkung des Wertevorrats kann abgemildert werden, indem Feldzuordnungen zu Schlüsselfeldern der Prüftabelle durch Markieren des Flags GENERISCH im Fremdschlüssel-Popup aufgehoben werden. Der Inhalt dieses Feldes wird bei der Fremdschlüsselprüfung dann nicht mehr zur Selektion von Datensätzen in der Prüftabelle herangezogen.

Abhängigkeit

Für Fremdschlüssel können so genannte semantische Eigenschaften gepflegt werden. Diese geben Auskunft über die Art der Abhängigkeit der Fremdschlüsselfelder von der Prüftabelle. Für die eigentliche Datenprüfung in Dynpros sind

sie nicht von Bedeutung. Sie werden allerdings zur Bildung von so genannten Aggregaten (Views, Eingabehilfen etc.) herangezogen.

Über die Kardinalität wird festgelegt, wie viele abhängige Sätze zu einem Satz der Prüftabelle existieren können. Sie wird in der Form n:m angegeben. Der Wert 1 für n besagt, dass zu jedem Satz in der Fremdschlüsseltabelle ein Satz in der Prüftabelle existieren soll. Der Wert C hingegen lässt zu, dass auch Sätze in der Fremdschlüsseltabelle existieren, für die es keine Sätze in der Prüftabelle gibt. Mit dem Wert für m wird vorgegeben, wie viele Sätze in der abhängigen Tabelle zu jedem Prüftabellensatz existieren dürfen. Hier steht 1 für genau einen, C für maximal einen, N für mindestens einen und CN für beliebig viele.

Über den Abhängigkeitsgrad wird festgelegt, ob die Fremdschlüsselfelder, die in der Fremdschlüsseltabelle ja keine Schlüsselfelder sein müssen, die Datensätze in der Prüftabelle eindeutig identifizieren oder nicht. Von besonderer Bedeutung ist hier die Auswahl SCHLÜSSELFELDER EINER TEXTTABELLE. Sie besagt, dass die Fremdschlüsselfelder der Fremdschlüsseltabelle bis auf ein zusätzliches Sprachenfeld exakt den Schlüsselfeldern der Prüftabelle entsprechen und selbst Schlüsselfelder sind.

5.2 Views

Views stellen eine Sicht auf eine oder mehrere Tabellen dar. Sie werden sehr häufig benutzt, um ohne komplizierte Abfragen oder Programme die Informationen aus mehreren Tabellen miteinander zu verknüpfen oder speziellen Anwendern nur die Sicht auf ausgewählte Felder einer Tabelle zu ermöglichen. Views werden recht häufig im Customizing oder für Spezialaufgaben eingesetzt. Viele Views werden automatisch generiert. Da Views in der allgemeinen Programmierung keine so bedeutende Rolle spielen wie Tabellen oder Dynpros, soll an dieser Stelle lediglich das Erzeugen eines einfachen Views demonstriert werden. Einige Grundlageninformationen zu Views finden Sie in Kapitel 2.

Für die Bearbeitung von Views steht in der Dictionary-Pflege wie für Tabellen und Strukturen ein eigener Auswahlpunkt zur Verfügung. Da Views ähnlich wie Tabellen benutzt werden können, insbesondere auch mit der Anweisung `TABLES` in Anwendungen eingebunden werden können, muss ein Name benutzt werden, der nicht schon für Tabellen oder Strukturen vergeben wurde. Es bietet sich an, bereits im Namen einen Hinweis darauf zu geben, dass es sich um eine View handelt. Dies ist lediglich eine Empfehlung, keine Forderung. Die Namen vieler der von SAP ausgelieferten Views beginnen beispielsweise mit dem Buchstaben V. Oft geht aus dem Namen auch der Name der in der View benutzten Primärtabelle hervor. Für dieses Beispiel bietet es sich daher an, den Namen `YVZ3B` zu benutzen. Im Grundbild der Dictionary-Pflege wird dieser Name eingetragen, die Auswahl VIEW markiert und die Drucktaste ANLEGEN betätigt. Nach einem Popup, in dem der Viewtyp ausgewählt werden muss (für dieses Beispiel benutzen Sie bitte den Typ DATENBANKVIEW), erscheint das Grundbild der Viewpflege (Abbildung 5.27).

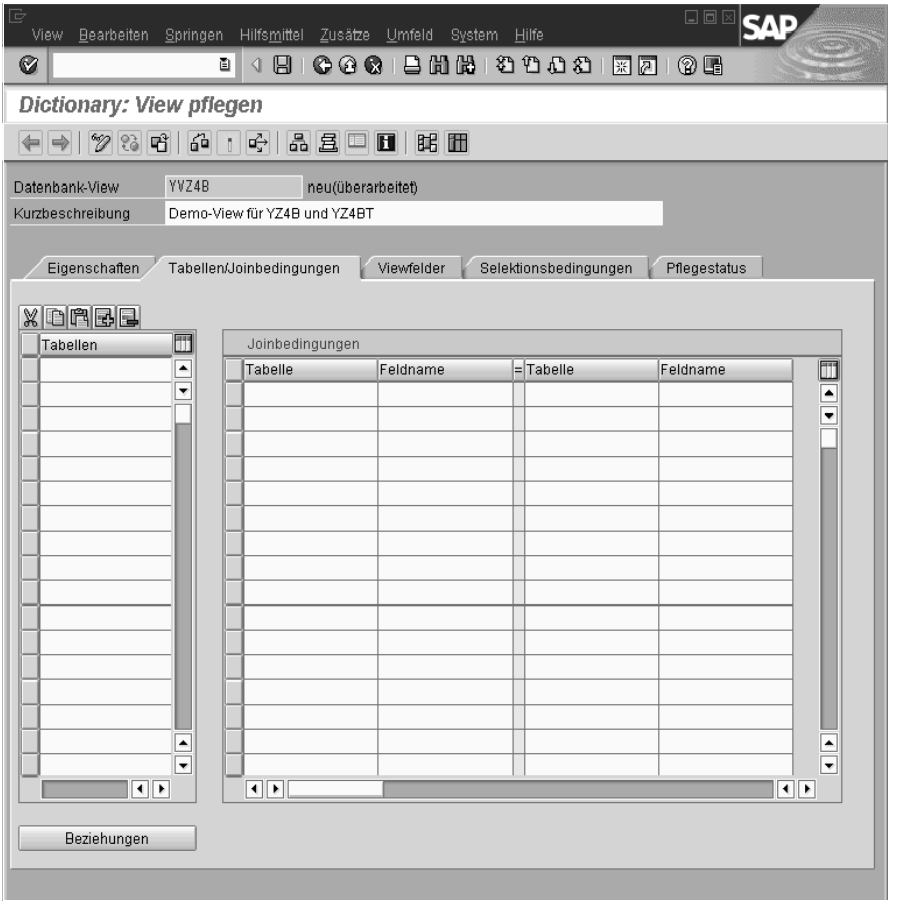

Abbildung 5.27 **© SAP AG**
Viewpflege

Wie für alle anderen Objekte auch ist zunächst eine Kurzbeschreibung zu erfassen. Die eigentliche Definition der View erfolgt in einigen Registerkarten der Pflegeoberfläche. Unmittelbar nach dem Start ist der Bereich TABELLEN/JOINBEDINGUNGEN aktiv, der zwei Table-View-Bereiche enthält. Im linken Table View TABELLEN müssen Sie alle Tabellen eintragen, aus denen Felder in die View aufgenommen werden sollen. Für dieses Beispiel sollen dies die Tabellen `YZ4B` und `YZ4BT` sein.

Die Bedingungen, mit denen die Datensätze der beteiligten Tabellen miteinander verknüpft werden, nimmt der Bereich JOINBEDINGUNGEN auf. Um die Joinbedingungen auf einfache Weise pflegen zu können, bietet das System wiederum eine Eingabehilfe, die ihre Informationen aus existierenden Fremd-

schlüsselbeziehungen und gemeinsam verwendeten Datenelementen bzw. Domänen bezieht. Setzen Sie den Kursor im linken Table View auf die Tabelle YZ4B und betätigen Sie die Drucktaste BEZIEHUNGEN unter diesem Table View. Das System blendet nun ein Popup auf, das alle Tabellen auflistet, die mit der vorher markierten Tabelle in einer Beziehung stehen. Durch einen Doppelklick auf eine der Zeilen blenden Sie zusätzlich die möglichen Verknüpfungen (Joins) der Felder der beiden Tabellen ein. Abbildung 5.28 zeigt das Popup.

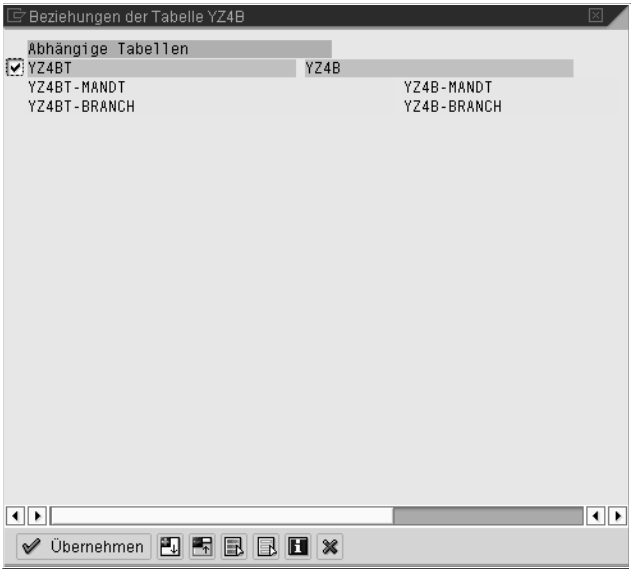

Abbildung 5.28 **© SAP AG**
Joinbedingungen auswählen

Durch die Drucktaste Übernehmen im Popup übernehmen Sie die ausgewählte Beziehung mit den dazugehörigen Feldverknüpfungen in den rechten Table View des Pflegedialogs. Sollte die Notwendigkeit bestehen, können Sie nun Joinbedingungen manuell bearbeiten. Sie können natürlich auch auf die Eingabehilfe verzichten und die Beziehungen der beteiligten Tabellen vollständig manuell erfassen (Abbildung 5.29).

An dieser Stelle können Sie die View-Definition zunächst abspeichern. Nun sind noch die Felder auszuwählen, die in die View aufgenommen werden sollen. Wechseln Sie dazu in die Registerkarte VIEWFELDER (ABBILDUNG 5.30).

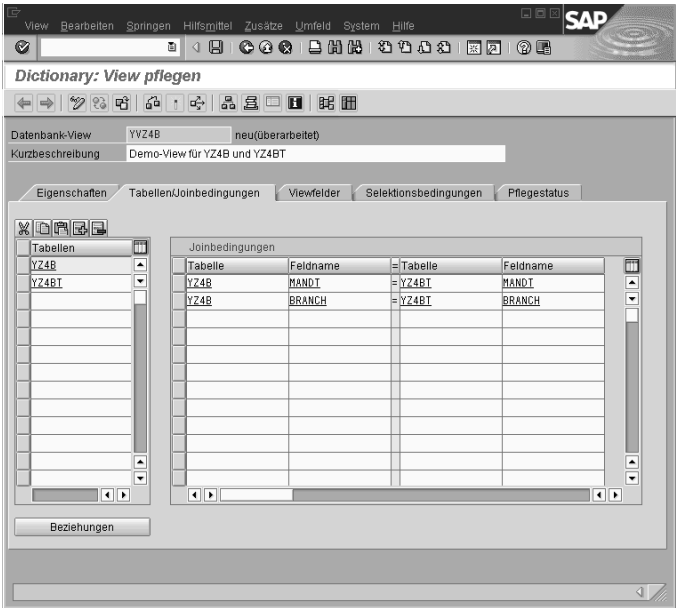

Abbildung 5.29 **© SAP AG**
Join-Bedingung

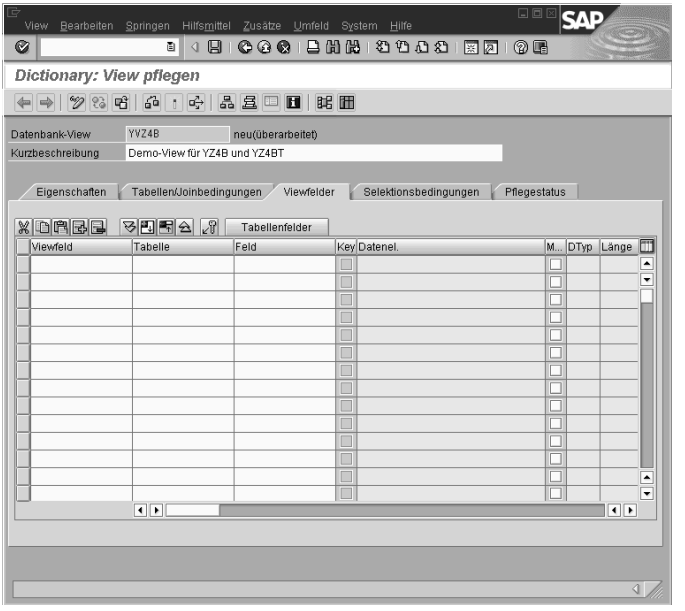

Abbildung 5.30 **© SAP AG**
Viewfelder bestimmen

Auf dieser Registerkarte existiert eine Drucktaste TABELLENFELDER. Ein Druck auf diese Taste blendet zunächst ein Popup auf, in dem durch Doppelklick eine der beteiligten Tabellen ausgewählt werden kann. Ein darauf folgendes Popup listet alle Felder dieser Tabelle auf (Abbildung 5.31). Die gewünschten Felder werden markiert und mit der Drucktaste ÜBERNEHMEN in den Table View übernommen. Da die in diesem Beispiel benutzten Tabellen aufeinander aufbauen, reicht es aus, alle Felder der Tabelle `YZ4BT` zu übernehmen. Daran anschließend speichern Sie die Angaben und aktivieren die View. Sie ist damit in Anwendungen benutzbar.

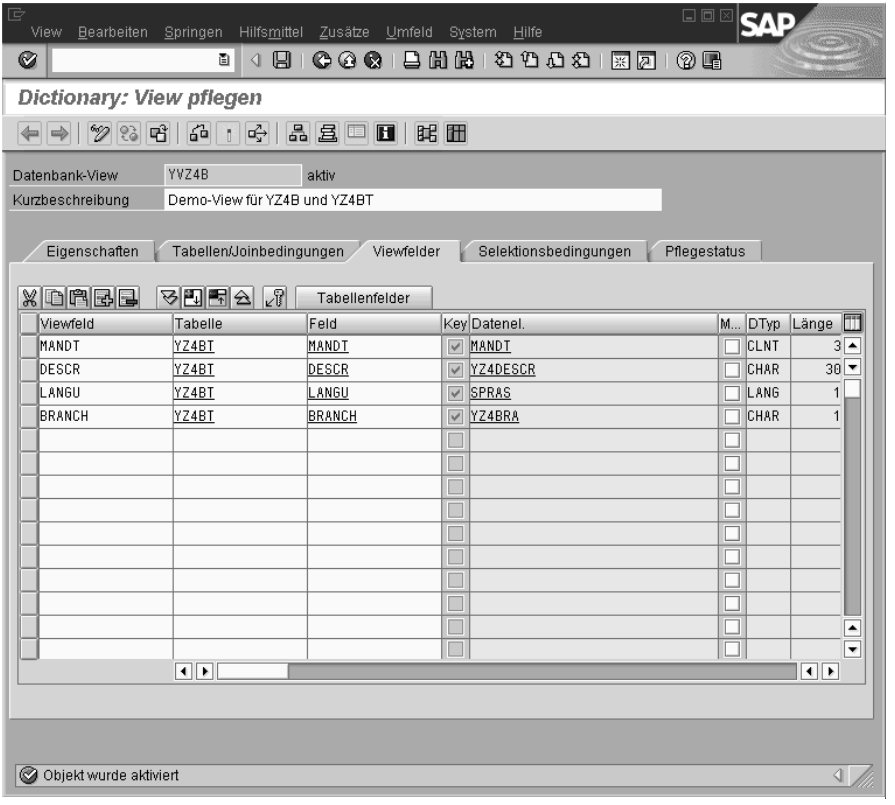

Abbildung 5.31 **© SAP AG**
View-Felder und deren Eigenschaften

Über Datenbank-Views können nur dann Änderungen der Tabellendaten ausgeführt werden, wenn nicht mehr als eine Tabelle in der View enthalten ist. In diesem Beispiel ist diese Bedingung nicht erfüllt, über die View ist damit nur das Lesen der Daten möglich. Der automatisch vorgeschlagene Pflegestatus, der auf der gleichnamigen Registerkarte zu finden ist, weist darauf hin.

Falls über selbst definierte Views mit mehreren Tabellen auch die Pflege der Daten möglich sein soll, muss der Typ C (Customizing View) benutzt werden.

Datenbank-Views (Typ D) werden als Views des Datenbanksystems realisiert, sie müssen also ähnlich wie Tabellen nach der Definition mit dem Datenbank-Utility auf der Datenbank angelegt werden. Während dies unter Release 3.0 automatisch beim Aktivieren erfolgt, muss in älteren Releaseständen das Datenbankwerkzeug manuell aufgerufen werden.

5.3 Suchhilfe

Mit der Version 4.0 wurden die zuvor existierenden Matchcodes durch eine allgemeine Suchhilfe abgelöst. Eine Suchhilfe ermöglicht die Suche nach einem Wert eines Tabellenfeldes über andere Felder derselben Tabelle oder aber einer mit dieser Tabelle verbundenen Texttabelle. Eine Suchhilfe kann beispielsweise dazu dienen, in der Tabelle `YZ4B` den einstelligen Bezeichner einer Branche über den Kurztext zu suchen. Suchhilfen werden mittels der Transaktion `SE11` gepflegt.

Jede Suchhilfe existiert zunächst als eigenständiges Element und kann separat benutzt werden. Man spricht in diesem Fall von einer elementaren Suchhilfe. Falls jedoch ein Wert durch unterschiedliche Verfahren und damit unterschiedliche elementare Suchhilfen gesucht werden kann, so können Sie mehrere elementare Suchhilfen zu einer Sammelsuchhilfe kombinieren. Die Entscheidung darüber, ob eine elementare oder eine Sammelsuchhilfe angelegt wird, fällt beim Anlegen einer neuen Suchhilfe. In einem Popup mit zwei Auswahlfeldern ist der gewünschte Typ zu markieren.

Bei der Zuweisung von Suchhilfen zu einem Feld können Sie wahlweise elementare oder Sammelsuchhilfen verwenden. Derartige Zuweisungen sind an verschiedenen Stellen möglich:

- Verbindung mit einem Dynprofeld: Innerhalb der Attributpflege für ein Dynprofeld existiert das Attribut Suchhilfe.
- Innerhalb der Transaktion SE11 können Sie Tabellen- oder Strukturfeldern ebenfalls eine Suchhilfe zuordnen. Dies erfolgt bei Strukturen auf der Registerkarte EINGABEHILFE/PRÜFUNGEN. Innerhalb der Tabellenpflege hingegen platzieren Sie den Cursor auf das gewünschte Feld und betätigen die Drucktaste SUCHHILFE, die sich innerhalb der Registerkarte befindet.
- Suchhilfen können mit Datenelementen verbunden werden. Die Suchhilfe wird an alle Felder vererbt, die auf diesem Datenelement beruhen. Sie pflegen die Zuordnung der Suchhilfe auf der Registerkarte Definition der Datenelementpflege.

Suchhilfen können mehr als ein Dynprofeld füllen. Außerdem können Sie die Inhalte mehrerer Dynprofelder als Vorgabe für die Suchkriterien übernehmen.

Zusätzlich können Suchhilfen einen frei programmierbaren Funktionsbaustein aufrufen. Durch diese Möglichkeit können Sie die Funktionalität der Suchhilfe beliebig erweitern. Abbildung 5.32 zeigt den Pflegedialog für eine elementare Suchhilfe.

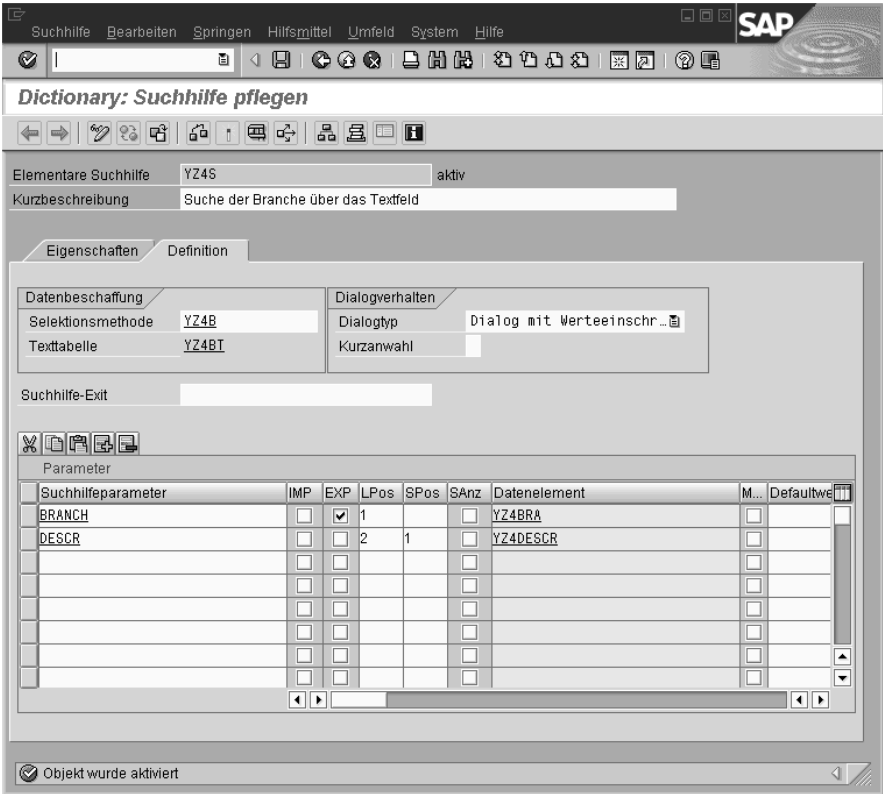

Abbildung 5.32 **© SAP AG**
Definition einer Suchhilfe

Ein sehr wichtiges Feld ist die SELEKTIONSMETHODE. In diesem Feld tragen Sie den Namen der Tabelle oder der View ein, in der Werte zu suchen sind. Falls zu dieser Tabelle eine Texttabelle existiert, wird deren Name automatisch im Feld TEXTTABELLE eingefügt. Die manuelle Belegung dieses Feldes ist nicht möglich.

Durch die Eingabe im Feld DIALOGTYP bestimmen Sie das Verhalten und die Erscheinungsform der Suchhilfe. Es stehen drei Auswahlmöglichkeiten zur Verfügung. Der Dialogtyp „Sofortige Werteanzeige“ sorgt dafür, dass alle ermittelten Werte sofort in einer Trefferliste aufgelistet werden, unabhängig davon, wie viele Datensätze gefunden wurden. Eine Suche im engeren Sinne ist damit nicht möglich, es handelt sich eher um eine Eingabehilfe.

Die Einstellung „Dialog mit Werteeinschränkung“ bewirkt zunächst die Abfrage von Suchbedingungen in einem Popup. Die Eingaben in diesem Dialog werden dann zur Selektion in der Tabelle herangezogen. Das Ergebnis der Abfrage steht dann als Trefferliste zur Verfügung. Über den Suchen-Popup pflegen Sie die aus Reports bekannten Selektionen. Sie können also Einzelwerte, Muster, Bereiche oder Wertegruppen benutzen, um die Treffermenge einzuschränken.

Die dritte mögliche Einstellung „Dialog abhängig von Wertemenge“ kombiniert beide zuvor genannten Varianten. Wenn die Trefferliste weniger als 100 Einträge enthält, wird sie sofort dargestellt. Sind es mehr, werden zunächst einschränkende Suchkriterien erfragt.

Die SUCHHILFEPARAMETER sind diejenigen Tabellen- oder Viewfelder, die innerhalb der Suchhilfe benutzt werden sollen. Dabei müssen alle beteiligten Felder angegeben werden. Sie können nur Felder aus der unter SELEKTIONSMETHODE benannten Tabelle oder der zugehörigen Texttabelle verwenden. Jeder der Suchhilfeparameter erhält zusätzliche Attribute, mit denen die Funktion des Parameters genauer beschrieben wird. Suchhilfeparameter können von der rufenden Anwendung mit Werten vorbelegt werden, falls für diesen Parameter das Flag IMPORT gesetzt ist. Die durch den Suchvorgang ermittelten Werte liefert die Suchhilfe nur für die Parameter mit aktiviertem EXPORT-Flag. Wenn die Suchhilfe mehrere mögliche Werte ermittelt, bietet sie eine Trefferliste zur Auswahl des gewünschten Eintrages an. Sie können auf den Aufbau der Trefferliste Einfluss nehmen, indem Sie in der Spalte LPOS die Reihenfolge der Felder in der Trefferliste festlegen. Wenn in der Spalte LPOS kein Eintrag erfolgt, erscheint das Feld nicht in der Trefferliste. Diese Spalte muss im Falle elementarer Suchhilfen mit Werten gefüllt werden, da eine Suchhilfe ohne Trefferliste sinnlos ist. Analog dazu bestimmen Sie durch den Wert in der Spalte SPOS die Position des Feldes im Popup zur Eingabe der Suchbegriffe. Auch hier gilt, dass Felder ohne Eintrag in dieser Spalte nicht im Suchen-Popup erscheinen. Mitunter ist es sinnvoll, das eigentlich zu suchende Feld nicht im Suchen-Dialog anzubieten.

Falls Sie für ein Feld das Flag SANZ markieren, wird es im eben erwähnten Popup nur angezeigt, besitzt aber keine Eingabefunktionalität. Das ist für Felder sinnvoll, die von der rufenden Anwendung vorbelegt werden.

Zur Demonstration der Suchhilfe soll eine solche angelegt werden, die in der Tabelle `YZ4B` den Kennbuchstaben für eine Branche über den beschreibenden Text sucht. Die Definition ist relativ einfach, Sie müssen nur die in Abbildung 5.28 ersichtlichen Werte eintragen und die Suchhilfe aktivieren. Zur Auswahl der Feldnamen in der Spalte Suchhilfeparameter können Sie die Eingabehilfe des Tabellenfeldes (F4-Hilfe) benutzen.

Der Test einer Suchhilfe ist direkt aus der Pflegetransaktion heraus mit der Menüfunktion SUCHHILFE | TESTEN möglich. Das System erzeugt zunächst ein Test-Dynpro, in welches das zu suchende Feld als Eingabefeld übernommen wird (Abbildung 5.33).

Abbildung 5.33 **© SAP AG**
Testumgebung für Suchhilfen

Wenn Sie in diesem Bild die Eingabehilfe für das Feld `BRANCH` aufrufen, wird die eigentliche Suchhilfe gestartet. Gemäß der Definition erscheint zunächst das Popup mit den Suchbegriffen (Abbildung 5.34).

Abbildung 5.34 **© SAP AG**
Eingabemaske für Suchwerte

Die gefundenen Werte werden anschließend in einer Trefferliste (Abbildung 5.35) angeboten. In dieser Trefferliste ist die Spalte mit dem zu übernehmenden Wert durch ein kleines Symbol neben der Spaltenüberschrift gekennzeichnet.

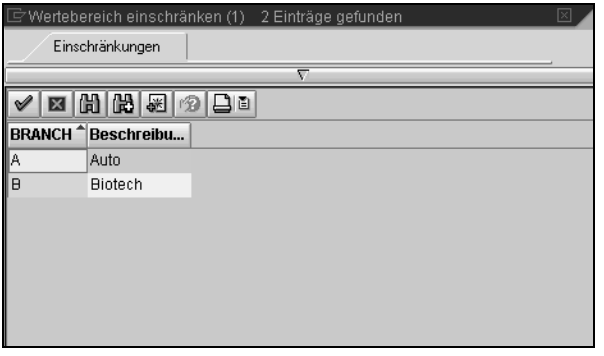

Abbildung 5.35 **© SAP AG**
Trefferliste

5.4 Sperrobjekte

Mit Sperrobjekten kann ein Programm den Zugriff auf ausgewählte Datensätze einer oder mehrerer Tabellen sperren. Die eigentliche Sperre wird dabei durch Aufruf eines vom System generierten Funktionsbausteines gesetzt oder freigegeben. Die Sperrfunktionalität greift dabei nicht auf Dienste des Datenbanksystems zurück, sondern wird im Dictionary mit SAP-Mitteln realisiert. Die erforderlichen Funktionsbausteine werden entsprechend der im Data Dictionary enthaltenen Definition eines Sperrobjekts aufgebaut. Diesen Funktionsbausteinen werden beim Aufruf als Parameter die Schlüsselwerte der zu sperrenden Datensätze übergeben. Bei der Definition eines Sperrobjekts sind daher vor allem zwei Arbeitsgänge notwendig. Zunächst sind die Tabellen anzugeben, die gemeinsam gesperrt werden sollen. Zu diesen Tabellen können dann noch die als Sperrargument notwendigen Felder ausgewählt werden. Dieser Vorgang ähnelt der Definition von Views und Matchcodes.

Das nachfolgende Beispiel beschreibt die Definition eines gemeinsamen Sperrobjekts für die beiden Tabellen `YZ4B` und `YZ4BT`.

Zu Beginn ist im Grundbild der Dictionary-Pflege der Name des Sperrobjekts einzutragen, das Auswahlfeld SPERROBJEKTE zu markieren und die Drucktaste ANLEGEN zu betätigen. Die Namen von Sperrobjekten sollen laut SAP-Empfehlung mit `E` beginnen. Der Rest des Namens ist wahlfrei, meist wird allerdings der Name der Primärtabelle benutzt. Der Name des Sperrobjekts für dieses Beispiel ist also `EYZ4B`.

Das Dynpro zur Pflege der Sperrobjekte ähnelt den bereits beschriebenen Werkzeugen (Abbildung 5.36).

Nach dem Start der Transaktion ist die Registerkarte TABELLEN aktiv. Innerhalb des Teilbereiches PRIMÄRTABELLE legen Sie die zu sperrende Tabelle sowie den Sperrmodus fest. Dieser Sperrmodus legt fest, ob und wie mehrere Anwender auf die gesperrten Datensätze zugreifen können. Der Modus *Lesesperre* (Kennbuchstabe S) erlaubt den gleichzeitigen Lesezugriff mehrerer Anwender. Erst wenn ein Anwender die Daten auch für Modifikationen beansprucht, ist kein weiterer Zugriff möglich. Der Modus *Schreibsperre* (STANDARDVORGABE, KENNBUCHSTABE E) sperrt die angegebenen Datensätze sofort und vollständig für alle anderen Anwender. Der sperrende Anwender kann aber nochmals sperren. Dieses so genannte kumulative Sperren wird durch den Modus *erweiterte Schreibsperre* (Kennbuchstabe X) ausgeschlossen, der Anwender kann den Datensatz also nur ein einziges Mal, also nur in einem Modus, sperren. Der Sperrmodus wird getrennt für die Primär- und die Sekundärtabellen eingestellt. Dazu steht jeweils ein Eingabefeld mit einer Listbox zur Verfügung.

Falls gleichzeitig mehrere Tabellen gesperrt werden sollen, so können im Teilbereich SEKUNDÄRTABELLEN zusätzliche Tabellen angegeben werden. Dabei ist keine manuelle Eintragung möglich, die zusätzlichen Tabellen können nur über eine Eingabehilfe ermittelt werden, die Sie durch die Schaltfläche HINZUFÜGEN aufrufen. Diese Eingabehilfe ermittelt unter Verwendung der Fremdschlüsselbeziehungen die mit der Primärtabelle verknüpften Tabellen und bietet sie in einem Popup zur Auswahl an.

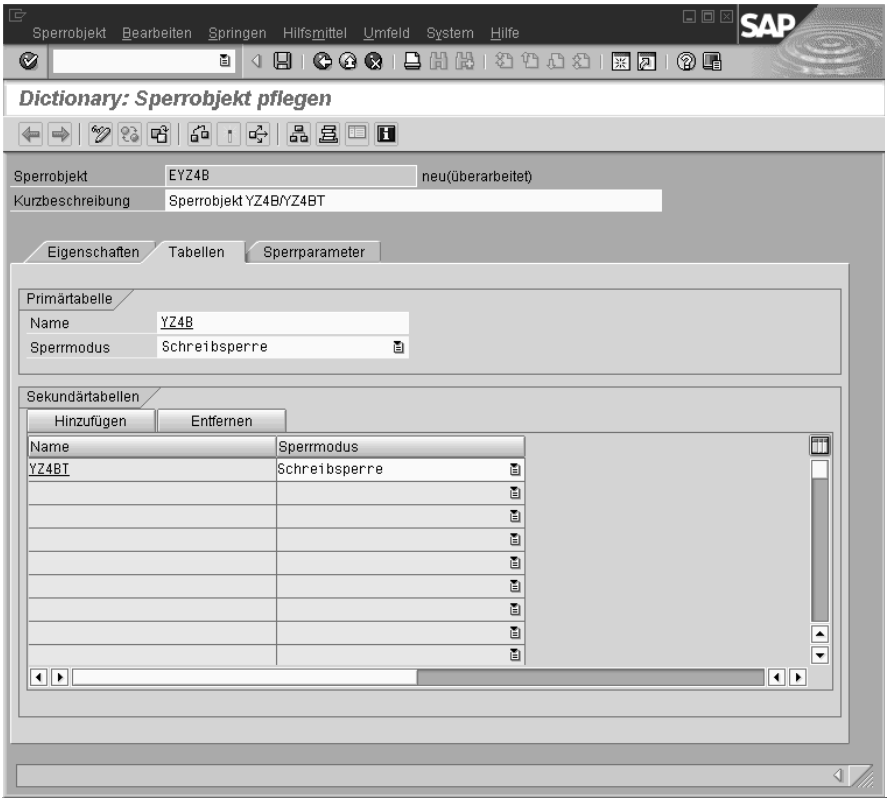

Abbildung 5.36 **© SAP AG**
Pflegebild für Sperrobjekte

Das Werkzeug übernimmt alle Schlüsselfelder der beteiligten Tabellen als potenzielle Sperrparameter. Nach Einfügen der Tabellen können Sie zur Karteikarte SPERRPARAMETER wechseln. In diesem Dynpro müssen Sie all die Felder markieren, die Sie später benutzen möchten, um den zu sperrenden Satz zu identifizieren. Im Normalfall werden dies alle Schlüsselfelder sein.

Nach Eingabe aller Attribute wird das Sperrobjekt aktiviert. Dabei werden zwei Funktionsbausteine erzeugt, deren Namen aus dem Präfix `ENQUEUE_` (Sperren) bzw. `DEQUEUE_` (Sperre freigeben, Entsperren) und dem Namen des Sperrobjekts bestehen. Die beiden Funktionsbausteine für dieses Beispiel heißen daher `ENQUEUE_EYZ4B` und `DEQUEUE_EYZ4B`. Diese Funktionsbausteine besitzen einige allgemeine Parameter und einige Parameter, die von den Sperrargumenten des Sperrobjekts abhängig sind. Das folgende Listing zeigt den Aufruf dieses Bausteins, wie er im Programmeditor mit der Funktion BEARBEITEN | ANWEISUNGSMUSTER eingefügt werden kann.

```
CALL FUNCTION 'ENQUEUE_EYZ4B'
*    EXPORTING
*         MODE_YZ3B      = 'E'
*         MODE_YZ3BT     = 'E'
*         MANDT          = SY-MANDT
*         BRANCH         =
*         LANGU          =
*         X_BRANCH       = ' '
*         X_LANGU        = ' '
*         _SCOPE         = '2'
*         _WAIT          = ' '
*         _COLLECT       = ' '
*    EXCEPTIONS
*         FOREIGN_LOCK   = 1
*         SYSTEM_FAILURE = 2
*         OTHERS         = 3
```

Alle Parameter des Funktionsbausteines – außer die eigentlichen Sperrparameter – verfügen über Vorgabewerte. Die Defaultwerte werden in das eingefügte Anweisungsmuster eingetragen, aber zunächst automatisch auskommentiert. Weichen die Vorgaben von den erforderlichen Werten ab, so ist der korrekte Wert einzutragen und das Kommentarzeichen zu entfernen.

Die ersten beiden Parameter sind die Sperrmodi für die Tabellen. Es folgen die eigentlichen Sperrargumente. Deren Namen entsprechen denen der Tabellenfelder. Für alle Sperrargumente außer dem Mandanten folgt je ein X-Parameter. Dieser Parameter legt fest, wie das eigentliche Sperrargument durch den Funktionsbaustein ausgewertet wird. Werden sowohl das Sperrargument als auch das X-Feld mit einem Leerzeichen belegt, erfolgt eine generische Sperre. Ein „X“ im X-Feld hingegen sorgt dafür, dass der Inhalt des Sperrarguments in der Form ausgewertet wird, wie er übergeben wurde.

Alle nun noch folgenden Parameter sind Parameter, die für jeden Sperrbaustein generiert werden. Der Inhalt des Parameters `_SCOPE` legt fest, wie eine gesetzte Sperre in Verbuchungsbausteinen behandelt wird. Die möglichen Werte und ihre Auswirkung zeigt Tabelle 5.6.

Wert	Wirkung
1	Sperren werden nicht an Verbucher weitergegeben.
2	Die Sperre wird an einen Verbucher weitergegeben. Der Dialog, der diese Sperre gesetzt hat, verliert damit seinen Einfluss auf die Sperre.
3	Die Sperre wird zusätzlich an den Verbucher übergeben. Sie muss später sowohl im Verbucher als auch im Dialog freigegeben werden.

Tabelle 5.6
Weitergabe von Sperren

Falls ein Sperrversuch fehlschlägt, beispielsweise weil der oder die zu sperrenden Datensätze bereits gesperrt sind, bricht der Funktionsbaustein normalerweise sofort mit einer Ausnahme ab. Wenn aber der Parameter _WAIT mit einem „X" belegt ist, unternimmt der Baustein nach einer kurzen Wartezeit weitere Sperrversuche. Die Zahl der Sperrversuche und die maximale Wartezeit werden durch Parameter im Systemprofil bestimmt. Diese Parameter können nur vom Systemadministrator auf Betriebssystemebene geändert werden. Über den Parameter _COLLECT können Sie bestimmen, ob sofort gesperrt werden soll oder Sperren zunächst gesammelt werden.

Ein Sperrfunktionsbaustein liefert über das Systemfeld SY-SUBRC eine Statusinformation zurück. Der Wert 0 steht für erfolgreiche Ausführung. Wenn die Sperre nicht ausgeführt werden kann, weil ein oder mehrere Sätze bereits durch andere Anwendungen oder Anwender gesperrt sind, so liefert der Baustein den Wert der Ausnahme FOREIGN_LOCK zurück. In diesem Fall kann dem Systemfeld SY-MSGV1 der Name des sperrenden Anwenders entnommen werden. Systemfehler hingegen werden durch die Ausnahme SYSTEM_FAILURE gemeldet. Funktionsbausteine, die zu Sperrobjekten mit Sperrmodus „X" gehören, besitzen noch eine dritte Ausnahme OWN_LOCK. Diese Ausnahme wird ausgelöst, wenn der gleiche Prozess bereits eine Sperre angefordert hat, die sich mit der aktuellen Sperranforderung überschneidet. Bei Verwendung der Standardvorgaben für die Parameter des Sperrbausteins würde die gesamte Tabelle gesperrt.

Wenn der Sperr-Funktionsbaustein eine Ausnahme erzeugt, erfolgt dies mit einem Kommando ähnlich diesem:

```
message e602(mc) with sy-uname raising foreign_lock.
```

Das bedeutet, dass alle nachrichtenbezogenen Systemfelder gefüllt werden. Sie können daher entweder eine eigene Fehlermeldung ausgeben oder aber die belegten Systemfelder benutzen, um die Standardnachricht auszusenden:

```
if sy-subrc <> 0.
  message id sy-msgid type sy-msgty number sy-msgno
          with sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
endif.
```

Die Funktionsbausteine zum Entsperren lösen keine Ausnahmen aus. Ihre Schnittstelle ist daher auch etwas einfacher gehalten, wie das folgende Listing zeigt.

```
CALL FUNCTION 'DEQUEUE_EYZ4B'
*    EXPORTING
*         MODE_YZ3B  = 'E'
*         MODE_YZ3BT = 'E'
*         MANDT      = SY-MANDT
*         BRANCH     =
*         LANGU      =
```

```
*          X_BRANCH   = ' '
*          X_LANGU    = ' '
*          _SCOPE     = '3'
*          _SYNCHRON  = ' '
*          _COLLECT   = ' '
```

Zusätzlich zu den bereits erläuterten Parametern existiert hier noch der Parameter `_SYNCHRON`. Ist er mit „*X*“ belegt, dann wartet der Baustein, bis die Sperre tatsächlich aufgehoben wurde. Ohne diesen Eintrag arbeitet dieser Baustein asynchron.

5.5 Typgruppen

Ab Version 3.0 existieren so genannte *Typgruppen*. In Typgruppen werden global verwendbare Typdefinitionen abgelegt. Typgruppen fügen sich eigentlich nicht in das Schema der bisher besprochenen Dictionary-Objekte ein, da sich hinter den Typgruppen keine datenbankspezifischen Objekte verbergen, sondern lediglich Dateien mit speziellen ABAP-Anweisungen. Da aber auch Dictionary-Objekte wie globale Typdefinitionen benutzt werden können, macht die Einordnung der Typgruppen an dieser Stelle Sinn.

In einer Typgruppe, auch als Type-Pool bezeichnet, werden Datentypen oder Konstanten deklariert. Dabei kommen die Anweisungen `TYPES` oder `CONSTANTS` zum Einsatz. In einer Anwendung können Typgruppen mit der Anweisung

```
TYPE-POOLS typgroup.
```

eingebunden werden. Danach sind alle Deklarationen der Typgruppe im Programm verfügbar. Typgruppen sollen einer Entwicklungsklasse zugeordnet werden. SAP empfiehlt daher, den Namen für eine Typgruppe aus dem Namen der Entwicklungsklasse und bei Bedarf ein oder zwei weiteren Buchstaben zu bilden. Die Namen der in einer Typgruppe deklarierten Elemente müssen mit dem Namen der Typgruppe und einem Unterstrich beginnen.

5.6 Übungen

❶ Falls in Ihrem System die Anmeldung mit verschiedenen Sprachen möglich ist, testen Sie die allgemeine Tabellenpflege für die beiden Tabellen unter verschiedenen Sprachen.

❷ Erzeugen Sie die Tabellen `YZ4STOCK` und `YZ4PRICE` sowie die dazu notwendigen Datenelemente und Domänen (Domänenname = Datenelementname). Die Struktur der Tabellen entnehmen Sie bitte der folgenden Übersicht (Tabellen 5.7 und 5.8). Für das Feld `WKZ` der Tabelle `YZ4PRICE` soll eine Fremdschlüsselbeziehung zum Feld `WKZ` der Tabelle `YZ4STOCK` bestehen, die Tabel-

le YZ4STOCK ist also Wertetabelle der Domäne YZ4WKZ. Für das Feld PRICE der Tabelle YZ4PRICE wird als Referenzfeld das Feld YZ4STOCK-CURRENCY eingetragen. Die erforderlichen Eingabefelder werden nach einem Doppelklick auf den Feldnamen verfügbar. Weiterhin wird für das Feld CURRENCY der Tabelle YZ4STOCK eine Fremdschlüsselbeziehung zur Tabelle TCURC hergestellt. Beide Tabellen erhalten die Auslieferungsklasse A. Die Bausteine zur allgemeinen Tabellenpflege müssen nicht generiert werden.

Feldname	Key	Datenelement	Typ	Länge	PrüfTab	Kurztext
MANDT	X	MANDT	CLNT	3	*	Mandant
WKZ	X	YZ3WKZ	NUMC	6		WKZ
NAME	X	YZ3DESCR	CHAR	30		Aktienname
BRANCH		YZ3BRA	NUMC	1	*	Bezeichnung für Branche
CURRENCY		WAERS_CURC	CUKY	5	TCURC	Währungsschlüssel

Tabelle 5.7
Struktur der Tabelle YZ4STOCK

Feldname	Key	Datenelement	Typ	Länge	PrüfTab	Kurztext
MANDT	X	MANDT	CLNT	3	*	Mandant
WKZ	X	YZ3WKZ	NUMC	6	YZ3STOCK	Aktienname
AKTDATE	X	YZ3DAT	DATS	8		Datum
PRICE		YZ3PRC	CURR	8		Kurs

Tabelle 5.8
Struktur der Tabelle YZ4PRICE

Hilfsmittel der Entwicklungsumgebung

6

Die Erstellung einer ABAP-Anwendung ist eine komplexe Aufgabe, für die eine Vielzahl von Werkzeugen benötigt wird. Einige Hilfsmittel erleichtern die Arbeit. Dieses Kapitel soll die wichtigsten Hilfsmittel ausführlich demonstrieren.

6.1 Korrektur- und Transportsystem

Das gesamte R/3-System ist eine sehr umfangreiche und komplexe Anwendung, an der bei SAP zeitweise mehrere hundert Entwickler gleichzeitig arbeiten. Die Entwicklungen müssen koordiniert und in einem funktionsfähigen Zustand zum Kunden gebracht werden. Diese Aufgabe erfüllen einige Werkzeuge, die unter dem Oberbegriff *Korrektur- und Transportwesen* zusammengefasst werden. Einige dieser Werkzeuge arbeiten im R/3-System, andere auf Betriebssystemebene.

Kenntnisse zum Korrektur- und Transportwesen und zur R/3-typischen Software-Logistik sind für jeden Software-Entwickler Voraussetzung für die Erstellung eigener Software. Nur durch Berücksichtigung diverser Anforderungen des SAP-Systems kann eine fehlerfreie Weitergabe der Software und deren korrekte Funktion nach einem SAP-Upgrade sichergestellt werden.

Die Einrichtung des Korrektur- und Transportwesens sowie einige systemnahe Tätigkeiten im Zusammenhang mit diesen Werkzeugen liegen nicht im Aufgabenbereich eines Programmierers. In diesem Kapitel werden zunächst die Bestandteile des Korrektur- und Transportwesens beschrieben, mit denen ein Entwickler tagtäglich in Berührung kommt. Es schließen sich Beispiele für typische Entwicklungstätigkeiten an. Den Abschluss bilden einige Hinweise zur Organisation eines R/3-Software-Projekts.

Ab Version 3.0 dient der *Workbench Organizer* als Oberfläche für alle Werkzeuge des Korrektur- und Transportwesens. Inzwischen wird er als *Transport Organi-*

zer bezeichnet. Es wird mit dem Transaktionscode SE09 aufgerufen. Das aus früheren Releaseständen bekannte Werkzeug (Transaktion SE01) ist weiterhin verfügbar, allerdings mit einer stark überarbeiteten Oberfläche. Es kommt vor allem dann zum Einsatz, wenn spezielle Probleme gelöst werden sollen, für die der Transport Organizer nicht flexibel genug ist.

6.1.1 Aufgaben

Ebenso wie eine Anwendung eine Tabelle gegen Fremdzugriff sperren kann, muss es einem Entwickler möglich sein, die von ihm momentan bearbeiteten Elemente vor der Bearbeitung durch Dritte zu schützen. Dabei müssen zwei unterschiedliche Arten von Sperren unterschieden werden. Möchte ein Programmierer ein Entwicklungsobjekt bearbeiten, so lesen die dazu notwendigen Werkzeuge die Beschreibung dieses Objekts aus der Datenbank. Dabei wird eine ganz normale Datenbanksperre gesetzt, die das gleichzeitige Bearbeiten des Objekts durch einen weiteren Entwickler verhindert. Diese Sperre ist nur so lange wirksam, bis der Entwickler das geänderte Objekt abspeichert und das Entwicklungswerkzeug verlässt.

Neben dieser Kurzzeitsperre existiert ein weiterer Mechanismus, der das Objekt während der gesamten Zeit der Entwicklung schützen kann. Diese Sperre unterscheidet sich von den Sperren auf Tabellen dadurch, dass sie über längere Zeit, mitunter einige Wochen, bestehen bleibt. Während dieser Zeit kann normalerweise nur der Entwickler das Objekt bearbeiten, der die Sperre auf dieses Objekt angelegt hat. Die erste Aufgabe des Korrektur- und Transportwesens besteht also darin, zu bearbeitende Objekte einem Entwickler zuzuordnen und den Zugriff anderer Entwickler zu verhindern.

Eine weitere Aufgabe besteht darin, die Bearbeitung von Objekten zu protokollieren. In speziellen Systemtabellen werden alle bearbeiteten Objekte und deren Bearbeiter erfasst. Auf diese Weise kann ermittelt werden, wann und durch wen ein Objekt bearbeitet wurde. Diese Informationen sind relevant, wenn eine Entwicklung in andere Systeme, z. B. die eines Kunden, übertragen werden soll.

Alle Entwicklungen müssen getestet und später zum Kunden oder in das eigene Produktivsystem transportiert werden. Entwicklung und Test in ein und demselben System sind, zumindest bei SAP, praktisch nicht durchführbar. Für einen Test ist ein definierter und vor allem ein über einige Zeit unveränderter Zustand der Anwendungen erforderlich. Dies würde bedeuten, dass während des Tests keine Entwicklungen stattfinden dürfen, was nur bei einem sehr kleinen Entwickler-Team möglich sein dürfte. Daher steht für die Entwicklung ein Systemverbund zur Verfügung. Die abgeschlossenen Entwicklungen werden aus dem Entwicklungs- in ein so genanntes Konsolidierungssystem transportiert (physisch kopiert) und dort getestet. Durch das Korrektur- und Transportwesen werden für die verschiedenen Entwickler und Projekte Listen mit allen bearbeiteten Objekten geführt. Nach Abschluss einer Entwicklung werden anhand die-

ser Listen die bearbeiteten Objekte ermittelt. Auf diese Weise ist der selektive Transport der bearbeiteten Objekte möglich. Auch bei Kunden existieren üblicherweise mindestens zwei Systeme, ein Test- und ein Produktivsystem. Im Testsystem werden beispielsweise Vorabkorrekturen oder überarbeitete Customizing-Einstellungen zunächst getestet und erst danach in das Produktivsystem kopiert.

Auf ähnliche Weise erfolgt der Transport von neuen Entwicklungen zum Kunden. Dabei werden, abgesehen von einer kompletten Neuinstallation, nicht alle, sondern nur die seit der letzten Auslieferung veränderten Objekte zum Kunden transportiert. Auch diese Funktionalität wird durch Auswertung der Protokolle des Transport- und Korrekturwesens möglich.

6.1.2 Prinzip

Wenn ein Anwender ein neues Objekt erstellen oder ein vorhandenes bearbeiten will, wird dieses Objekt in einen so genannten *Auftrag* aufgenommen. Dieser Auftrag wird neu angelegt oder vom Entwickler aus der Liste seiner aktuellen Aufträge gewählt. Durch Aufnahme eines Objekts in einen Auftrag ist dieses für die Bearbeitung durch andere Entwickler gesperrt. Hinter dem Begriff Auftrag verbirgt sich im Prinzip eine Stückliste, die mit einigen zusätzlichen Attributen versehen wird. Zu diesen Zusatzinformationen gehört etwa eine erläuternde Bezeichnung und der Eigentümername. Ein Entwicklungsobjekt, das in einem solchen Auftrag enthalten ist, kann in keinen anderen Auftrag aufgenommen werden. Auf diese Weise kann nur der Eigentümer des Auftrages das jeweilige Objekt bearbeiten. Damit innerhalb eines Entwicklungs-Teams mehrere Programmierer ein und dasselbe Objekt bearbeiten können, bietet das Korrektur- und Transportwesen die Möglichkeit, mehrere Entwickler und deren Aufträge zusammenzufassen.

Jeder Entwickler kann beliebig viele derartiger Aufträge anlegen. Auf diese Weise ist es möglich, zwischen unterschiedlichen Entwicklungen zu unterscheiden. Nach Abschluss einer Entwicklung wird der Auftrag freigegeben. Dies bedeutet zunächst, dass die enthaltenen Objekte nicht mehr gesperrt sind. Die Liste der Objekte, die innerhalb dieses Auftrages bearbeitet wurden, bleibt zwar erhalten, sie wird bei der Ermittlung eventueller Sperren aber nicht mehr ausgewertet. Abhängig von der Art des Systems, den bearbeiteten Objekten und der Systemkonfiguration werden die bearbeiteten Elemente gegebenenfalls auch exportiert. Dies bedeutet, dass die Beschreibung aller im Auftrag enthaltenen Entwicklungsobjekte in eine Datei auf Betriebssystemebene abgelegt wird. Der Aufbau dieser Datei ist unabhängig vom Betriebssystem. Sie kann zu einem anderen System gebracht und dort eingelesen werden. Auf diese Weise werden Entwicklungsobjekte in andere Systeme transportiert. Dies bedeutet auch, dass Entwicklungsobjekte, die nicht in einem Auftrag erfasst wurden, auch nicht transportiert werden können.

6.1.3 Begriffe

Im Zusammenhang mit dem Korrektur- und Transportwesen werden einige Begriffe verwendet, die nachfolgend beschrieben werden sollen. Dabei werden nur SAP-Versionen ab 3.0 berücksichtigt. Einige der Begriffe existierten schon in älteren Versionen, erhalten ab 3.0 aber teilweise eine etwas andere Bedeutung.

Paket

Eine komplexe R/3-Anwendung besteht aus vielen verschiedenen Entwicklungsobjekten. Früher wurden alle Elemente einer Anwendung (Programme, Dictionary-Objekte, Transaktionen usw.) zu einer so genannten Entwicklungsklasse zusammengefasst. Der Begriff der Entwicklungsklasse wurde inzwischen durch das Paket abgelöst. Prinzipiell erfüllt ein Paket zunächst dieselben Aufgaben wie eine Entwicklungsklasse. Sie bestimmt die Transporteigenschaften der in ihr enthaltenen Objekte, insbesondere das Zielsystem bei einem eventuellen Export. Eine neue Eigenschaft der Pakete besteht in der Möglichkeit, sie zu verschachteln. Außerdem kann für ein Paket festgelegt werden, auf welche der enthaltenen Objekte von außen zugegriffen werden darf.

Beim Erzeugen neuer Objekte müssen Sie im Popup des Korrektur- und Transportwesens den Namen eines Pakets eintragen. Im Grundbild der Workbench (Transaktion `SE80` bzw. `SEU` in älteren Versionen) steht ein Eingabefeld für den Paketnamen zur Verfügung. Bei entsprechender Eingabe zeigt der Object Browser alle Entwicklungsobjekte des gewählten Paketes an.

Vor Beginn einer Entwicklung sollte ein passendes Paket angelegt werden. Die Einrichtung kann innerhalb des Customizing, in der Transaktion `SE80` oder aber mittels der allgemeinen Tabellenpflege `SM30` erfolgen. Die zu pflegende Tabelle ist die `V_TDEVC`. Unabhängig von der Art des Aufrufs erfolgt die Bearbeitung der Daten einer Entwicklungsklasse in einem Popup gemäß Abbildung 6.1.

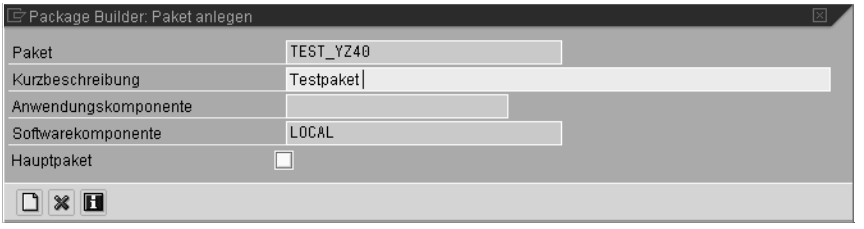

Abbildung 6.1 **© SAP AG**
Anlegen eines neuen Pakets

Die beiden Felder SOFTWAREKOMPONENTE und ANWENDUNGSKOMPONENTE erlauben es, Pakete zu größeren logischen Einheiten zusammenzufassen. Außerdem werden über diese Einstellungen indirekt die Transporteigenschaften festgelegt. Die zur Verfügung stehenden Komponenten und deren Eigenschaften

entziehen sich der direkten Bearbeitung durch den Entwickler, sie werden vom Systemadministrator gepflegt.

Ein Paket ist das erste Element in der Objektliste des Object Navigator. Es kann daher nach dem Anlegen innerhalb des Object Navigator weiter bearbeitet werden. Bild 6.2 zeigt das Pflegewerkzeug für Pakete. Die wichtigsten Eigenschaften und Zusammenhänge werden in den nachfolgenden Abschnitten beschrieben.

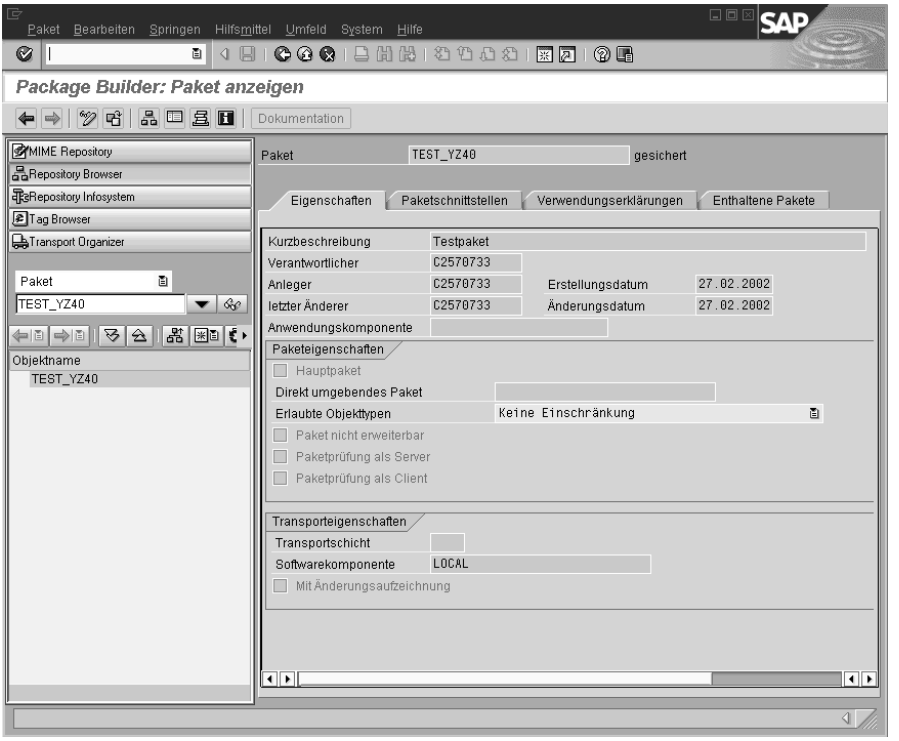

Abbildung 6.2 **© SAP AG**
Pflege der Eigenschaften eines Pakets

Auch für Pakete existieren Namensräume. Die Namen der Entwicklungsklasse von Kunden müssen mit Y oder Z oder einem für den Kunden vorgesehenen Namensraumpräfix beginnen. Eine Sonderrolle spielen Klassen, deren Name mit T beginnt. Dabei handelt es sich um lokale Entwicklungsklassen, deren Objekte nicht transportiert werden können. Sie werden benutzt, um die Vorteile des Korrektur- und Transportwesens (Protokollierung, Schutz gegen Bearbeitung durch Dritte) auch für lokale Anwendungen nutzen zu können.

Lokale private Objekte

Der Status *Lokales privates Objekt* kann an neu anzulegende Objekte vergeben werden. Derartige Objekte werden außerhalb des Korrekturwesens bearbeitet. Sie können nicht mit den Mitteln des Korrekturwesens gegen die Bearbeitung durch Dritte geschützt und nicht transportiert werden. Der einzige mögliche Schutz besteht in der Sperre des Objekts während der direkten Bearbeitung mit dem jeweiligen Entwicklungswerkzeug. Diese Sperre wird nicht durch das Korrekturwesen veranlasst. Es handelt sich dabei um eine ganz normale Datenbanksperre. Lokale private Objekte erhalten die Entwicklungsklasse `$TMP`.

Original und Kopie

Die Entwicklung von Anwendungen erfolgt in einem Entwicklungssystem, von dem aus die Entwicklungsobjekte weiter verteilt werden. Dieser Vorgang erfolgt periodisch, um Fehler zu beheben oder neue Funktionalität bereitzustellen. Die permanente Änderung an einem Entwicklungsobjekt ist also nur in dem System möglich, von dem aus die Auslieferung erfolgt. Für jedes Entwicklungsobjekt wird daher das System vermerkt, in dem es erzeugt wurde. Dieses System wird auch als Originalsystem des Objekts bezeichnet, das Objekt in diesem System ist das Original. In allen anderen Systemen, auch den Kundensystemen, die über SAP-Upgrades beliefert werden, befinden sich nur Kopien des Objekts. Bei Transporten von Objekten zwischen verschiedenen SAP-Systemen verhindert das Korrektur- und Transportwesen, dass Originalobjekte überschrieben werden. Das Überschreiben von Kopien ist aber jederzeit möglich.

Aufgabe

Eine Aufgabe ist die kleinste physische Organisationseinheit im Korrekturwesen. Sie dient zur eigentlichen Registrierung der bearbeiteten Objekte und zur Realisierung der Bearbeitungssperre. Zur Identifikation einer Aufgabe dient eine eindeutige Nummer. Eine oder mehrere Aufgaben werden einem Auftrag zugeordnet. Immer dann, wenn das Korrekturwesen ein neu zu bearbeitendes Objekt registrieren will, muss der Programmierer zwar die Nummer eines Auftrags eingeben, die Speicherung der Information erfolgt aber zunächst in einer Aufgabe innerhalb dieses Auftrags. Sie entspricht damit im Prinzip der bereits erwähnten Liste. Eine Aufgabe kann immer nur einen Eigentümer haben.

Das Korrekturwesen unterscheidet Aufgaben mit unterschiedlichem Typ. Da innerhalb eines Systems Entwicklungsobjekte mit unterschiedlichem Status (Original oder Kopie) existieren, die individuelle Behandlung erfordern, ist eine derartige Unterscheidung notwendig. Eine Aufgabe kann immer nur Objekte mit identischem Status enthalten. Die verschiedenen Typen von Aufgaben (auch als Eigenschaften von Aufgaben bezeichnet) sind in Tabelle 6.1 aufgeführt. In der Regel werden Aufgaben durch das Korrekturwesen automatisch

angelegt, wobei auch der Typ zugeordnet wird. Dieser Typ ist für den Benutzer des Korrekturwesens in der Regel ohne Bedeutung.

Aufgabentyp	Beschreibung
Nicht zugeordnet	Neu angelegte Aufgaben ohne Inhalt.
Entwicklung/Korrektur	Enthält nur Objekte, die im aktuellen SAP-System neu angelegt wurden (Originale).
Reparatur	Enthält nur Objekte, die in anderen SAP-Systemen angelegt und in das aktuelle SAP-System transportiert wurden (Kopien).

Tabelle 6.1
Eigenschaften von Aufgaben

In früheren Versionen existierte der Begriff Aufgabe nicht. Daher wurden Aufgaben des Typs Entwicklung/Korrektur auch als *Korrektur* und Aufgaben des Typs Reparatur als *Reparatur* bezeichnet.

Auftrag

Der Auftrag ist eine logische Verwaltungseinheit, die alle Aufgaben für eine in sich abgeschlossene Entwicklungsaufgabe zusammenfasst. Er kann eine oder mehrere Aufgaben unterschiedlichen Typs und verschiedener Eigentümer enthalten. Der Auftrag bestimmt für alle Objekte der in ihm enthaltenen Aufgaben die Art und Weise der Weiterleitung in andere Systeme. Aufträge werden in der Regel automatisch angelegt, wobei auf jeden Fall eine untergeordnete Aufgabe erzeugt und mit dem neuen Auftrag verbunden wird.

Wenn ein Objekt durch das Korrekturwesen registriert werden soll, gibt der Programmierer den Auftrag an, dem das Objekt zugeordnet werden soll. Das Korrekturwesen ermittelt dann die konkrete Aufgabe, in der das Objekt registriert wird, anhand des Benutzers und des Status des Objekts automatisch. Falls keine Aufgabe mit einem geeigneten Typ existiert, wird sie vom System angelegt.

Auch Aufträge besitzen einen Typ, der durch die Transporteigenschaften der in ihm enthaltenen Objekte, genauer gesagt die des ersten aufzunehmenden Objekts, bestimmt wird (siehe Tabelle 6.2). Alle im weiteren Verlauf der Entwicklung zu registrierenden Objekte müssen dieselben Transporteigenschaften haben. Ist dies nicht der Fall, kann das Objekt nicht in diesem Auftrag registriert werden. Das Korrekturwesen legt dann einen neuen Auftrag an.

Typ eines Auftrags	Beschreibung
Nicht zugeordnet	Leerer Auftrag.
Transportierbar	Auftrag mit Objekten, die in ein anderes SAP-System exportiert werden können.
Lokal	Auftrag mit Objekten, die nicht aus dem aktuellen SAP-System exportiert werden können.

Tabelle 6.2
Eigenschaften eines Auftrags

Der Eigentümer eines Auftrags kann durch spezielle Funktionen des Workbench Organizers in seinen Aufträgen weitere Aufgaben für andere Entwickler anlegen. Alle unter einem Auftrag zusammengefassten Entwickler können alle Objekte, die in Aufgaben innerhalb dieses Auftrages erfasst sind, bearbeiten.

Für jede Entwicklungsklasse wird in verschiedenen Systemtabellen festgelegt, ob Elemente dieser Klasse aus dem aktuellen System heraus transportiert werden dürfen oder nicht. Falls nicht transportierbare Objekte bearbeitet werden, erzeugt das System automatisch einen lokalen Auftrag, ansonsten einen transportierbaren Auftrag.

Freigabe

Nach Abschluss einer Entwicklung müssen die Aufgaben und Aufträge freigegeben werden. Dies erfolgt mit speziellen Menüfunktionen des jeweiligen Transportwerkzeugs und kann nur vom Eigentümer der Aufgabe oder des Auftrags ausgeführt werden.

Die Freigabe von Aufgaben bewirkt zunächst nur, dass die Stückliste der Aufgabe in die Stückliste des übergeordneten Auftrages kopiert wird. Die Sperre der Objekte bleibt erhalten, wird jetzt aber durch den Auftrag realisiert. Wenn alle Aufgaben eines Auftrages freigegeben wurden, so kann auch er freigegeben werden. Dabei wird die Sperre für die Objekte aufgehoben. Falls es sich um einen transportierbaren Auftrag handelt, werden die Objekte exportiert. Freigegebene Aufgaben oder Aufträge können nicht mehr bearbeitet werden.

6.1.4 Beispiel für Neuentwicklung im Kundensystem

Dieser Abschnitt demonstriert die Vorgänge beim Erstellen eines neuen Objekts in einem Kundensystem. Die Namen der beteiligten Objekte entsprechen daher den Konventionen für Kundenobjekte. Aus Gründen der Verfügbarkeit wurde dieses Beispiel an einem Kundensystem mit der Version 4.5 erstellt. An der prinzipiellen Verfahrensweise hat sich allerdings nichts Wesentliches geändert, so

dass die vermittelten Kenntnisse problemlos auch an neueren Systemen verwendbar sind. Vorausgesetzt, Sie arbeiten an einem Kundensystem und verfügen über die erforderlichen Berechtigungen, können Sie dieses Beispiel im Prinzip nachvollziehen. Das Korrektur- und Transportwesen ist allerdings sehr stark von systemspezifischen Einstellungen abhängig, so dass der Arbeitsablauf in Ihrem System von dem hier beschriebenen abweichen kann. Prinzipiell sollten Sie in Ihrem System aber nur dann transportierbare Objekte anlegen und tatsächlich transportieren, wenn Sie sich über die Auswirkungen auf Ihr System im Klaren sind.

Es soll der Report `YZ361010` bearbeitet werden. Dazu wird über die Transaktion `SE38` oder die Workbench ein Report mit diesem Namen angelegt. Im Attributebildschirm des Programmeditors werden Kurzbeschreibung und Programmtyp erfasst. Diese Arbeitsschritte sind identisch mit denen beim Anlegen des „Hello-World"-Programms in Kapitel 3. Wenn die Angaben im Attribute-Bildschirm abgespeichert werden sollen, erscheint das erste Popup des Korrekturwesens (siehe Abbildung 6.2). Bei allen anderen Beispielen dieses Buches wurde dieses Popup generell mit der Betätigung der Drucktaste LOKALES PRIVATES OBJEKT beendet, wodurch Objekte entstanden, die nicht der Kontrolle des Korrektur- und Transportwesens unterlagen.

Abbildung 6.3 **© SAP AG**
Zuordnen einer Entwicklungsklasse

Falls das Objekt aber mit dem Korrekturwesen angelegt werden soll, muss in diesem Popup eine Entwicklungsklasse (hier `YZ30`) eingetragen werden. Damit werden die Transporteigenschaften dieser Klasse (Zielsystem) für das neue Objekt übernommen. Anschließend wird das Popup mit der SICHERN-Taste beendet. Das Korrektursystem erkennt, dass das neue Objekt in einen Auftrag aufgenommen werden muss. Es erfragt daher in einem zweiten Popup (Abbildung 6.3) die Auftragsnummer.

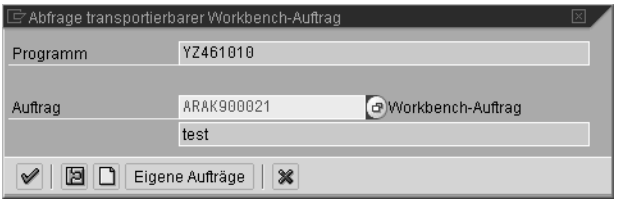

Abbildung 6.4 **© SAP AG**
Festlegen des Auftrags

Mittels der Drucktaste EIGENE AUFTRÄGE können die vorhandenen eigenen Aufträge, deren Eigenschaften die Aufnahme des aktuellen Objekts zulassen, als Eingabehilfe angezeigt werden. Eventuell wird in das Eingabefeld für die Auftragsnummer die letzte bearbeitete Nummer als Vorschlag automatisch eingefügt. Mittels der Drucktaste AUFTRAG ANLEGEN wird ein neuer Auftrag erzeugt. In diesem Fall ist in einem dritten Popup eine Kurzbeschreibung zu erfassen. Nach Abspeichern des dritten Popups gelangt man zurück zum zweiten, in dem man mit der Enter-Taste oder alternativ dazu mit der entsprechenden Drucktaste (grünes Häkchen) den Auftrag endgültig anlegt. In diesen Auftrag können nun bei Bedarf weitere Objekte aufgenommen werden.

Nach Beendigung der Entwicklung muss der Auftrag freigegeben werden. Dazu wird der *Workbench Organizer* benutzt, der nicht mit der eigentlichen Workbench verwechselt werden darf. Er ist über den Transaktionscode SE09 oder vom Hauptmenü der Programmentwicklung aus mit der Menüfunktion ÜBERSICHT | WORKBENCH ORGANIZER erreichbar. Abbildung 6.4 zeigt die Oberfläche dieses Werkzeugs.

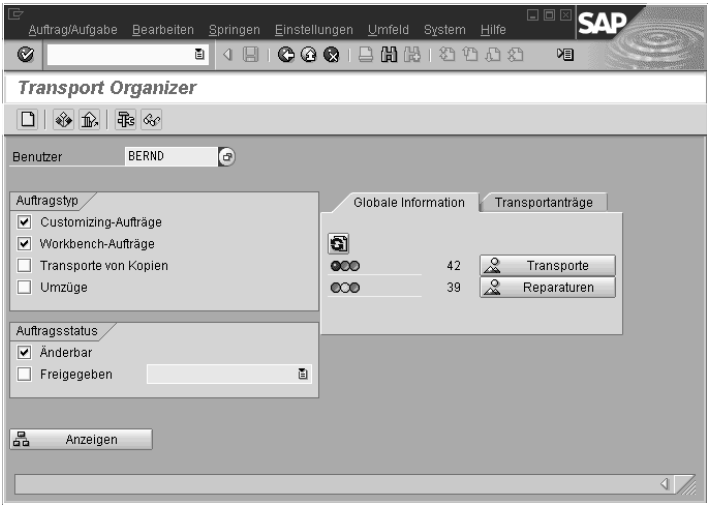

Abbildung 6.5 **© SAP AG**
Der Workbench-Organizer

Vom Grundbild des Workbench Organizers aus können alle Aufträge und Aufgaben gepflegt werden. Die rechte Seite des Dynpros erlaubt einen Überblick über alle Transporte des aktuellen Benutzers. Falls das System in einen so genannten Transportverbund eingebunden wurde, gehen in diese Übersicht auch Transporte aus anderen Systemen ein.

Auf der linken Seite des Dynpros werden die zu bearbeitenden Aufgaben und Aufträge zunächst über ihre Eigenschaften selektiert. Nach Betätigung der Drucktaste ANZEIGEN erscheinen in einem zweiten Dynpro (Abbildung 6.4) alle Aufgaben und Aufträge, die den markierten Eigenschaften entsprechen.

Die beiden Felder ÄNDERBAR und FREIGEGEBEN erlauben die Differenzierung zwischen freigegebenen, also erledigten Aufträgen und solchen, die noch bearbeitet werden können.

Um den eben erzeugten Auftrag freigeben zu können, sind zumindest die Felder ÄNDERBAR und TRANSPORTIERBAR zu markieren. Nach Druck auf die Taste ANZEIGEN baut die Transaktion ein neues Dynpro auf, das einen interaktiven Report enthält (siehe Abbildung 6.5).

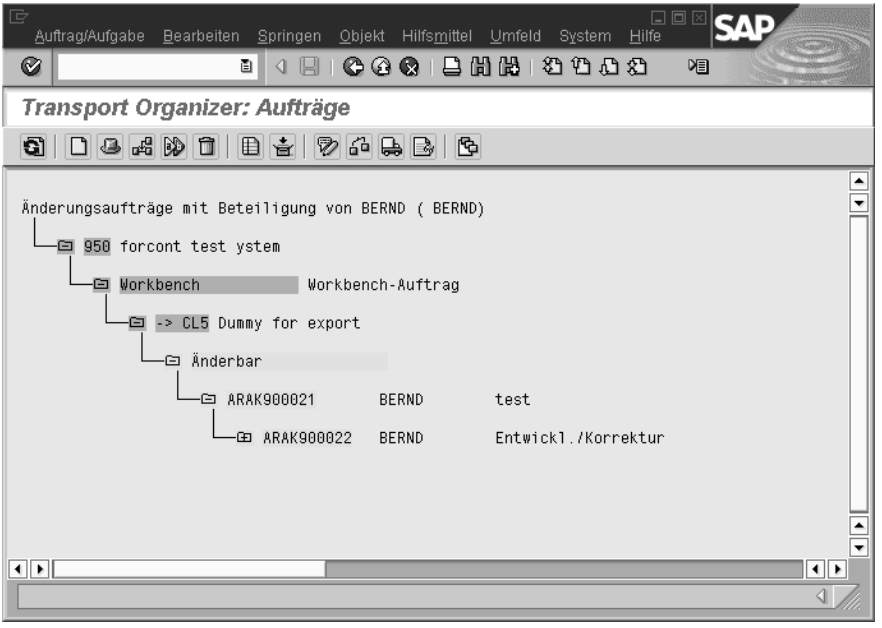

Abbildung 6.6 **© SAP AG**
Übersicht über Aufgaben und Aufträge

Ein Einzelklick auf das Ordnersymbol vor der Auftragsnummer blendet zunächst alle zu diesem Auftrag gehörenden Aufgaben ein. Weitere Mausklicks auf die Ordnersymbole der Aufgaben zeigen deren Inhalt an. Um einen Auftrag frei-

zugeben, müssen zunächst alle Aufgaben freigegeben werden. Dazu wird der Cursor auf die Nummer der freizugebenden Aufgabe platziert und die Drucktaste FREIGEBEN betätigt. Im Verlauf der Freigabe können Sie für jede Aufgabe eine Dokumentation erfassen. Dies ist nicht unbedingt erforderlich, Sie können das vom System vorgeschlagene Muster einfach abspeichern und den Doku-Editor mit der Funktionstaste F3 verlassen. Freigegebene Aufgaben werden im Workbench Organizer durch eine andere Farbe hervorgehoben. Anschließend kann der Vorgang für den Auftrag wiederholt werden. Nach Freigabe einer Aufgabe wird die Stückliste des Auftrages (die Liste der bearbeiteten Objekte) in eine Liste kopiert, die dem Auftrag zugeordnet ist. Dadurch wird die sperrende Wirkung aufrechterhalten, bis auch der Auftrag freigegeben wurde. Nach der Freigabe des Auftrages kann das Objekt durch einen anderen Anwender bearbeitet werden.

Beim Export aus dem System heraus und dem Import in ein anderes System kann es zu verschiedenen Fehlern kommen. Das System erstellt daher ein Protokoll, das mit der Menüfunktion SPRINGEN | PROTOKOLLE | TRANSPORTPROTOKOLL eingesehen werden kann.

Das Detailbild für die Aufgaben und Aufträge bietet einige weitere wichtige Bearbeitungsmöglichkeiten. Mit der Drucktaste INHABER ÄNDERN erhält das markierte Objekt einen anderen Eigentümer. Nur der Eigentümer kann einen Auftrag oder eine Aufgabe bearbeiten, also neue Objekte aufnehmen oder den Auftrag freigeben. Falls bei Abwesenheit des Eigentümers eines Auftrags oder einer Aufgabe Änderungen erforderlich sind, kann auch ein anderer Mitarbeiter das Objekt weiter bearbeiten. Diese Funktion kann auch mit fremden Objekten ausgeführt werden, man kann sich also die Objekte anderer Mitarbeiter holen.

Wenn die Änderung des Inhabers nicht sinnvoll erscheint, können einem Auftrag auch weitere Mitarbeiter zugeordnet werden. Dazu wird mit dem Cursor ein Auftrag markiert und die Drucktaste MITARB.HINZUF. gedrückt. In einem Popup kann der Name eines zusätzlichen Mitarbeiters eingetragen werden. Für diesen Mitarbeiter wird dann eine eigene Aufgabe angelegt. Diese Funktion kann nur durch den Eigentümer des Auftrags ausgeführt werden.

6.2 Berechtigungskonzept

Berechtigungen sind außerordentlich wichtig, um Programme und Daten vor Fremdzugriffen zu schützen und sollten daher auch in eigenen Entwicklungen benutzt werden. Den Berechtigungen liegt allerdings ein recht kompliziertes System zu Grunde, außerdem erweist sich die Pflege der Berechtigungen für die verschiedenen Anwender als sehr zeitaufwändig. Jede weitere Berechtigung erhöht den Aufwand für die Pflege und die Wahrscheinlichkeit von Problemen. Aus diesen Gründen sollten in eigenen Anwendungen nach Möglichkeit bereits vorhandene Berechtigungen benutzt werden. Da sich Kundenentwicklungen in den meisten Fällen auf Ergänzungen zu vorhandenen SAP-Anwendungen beschränken, ist dies in der Regel möglich. Auch wenn eigene Berechtigungen ge-

schaffen werden müssen, kann sehr oft auf vorgefertigte Elemente, so genannte Berechtigungsobjekte, zurückgegriffen werden. Beim Einbau von Berechtigungsprüfungen in eigenen Anwendungen unterscheidet man zwischen drei unterschiedlich aufwändigen Methoden:

- Vorhandene Berechtigungen nutzen
- Neue Berechtigungen auf Grundlage vorhandener Berechtigungsobjekte anlegen und benutzen
- Neue Berechtigungsobjekte erzeugen und von diesen eigene Berechtigungen ableiten

Für die Pflege der diversen Elemente des Berechtigungssystems sind spezielle Berechtigungen erforderlich, über die aus Gründen der Systemsicherheit natürlich nicht jeder Anwender verfügen darf. Die nachfolgend beschriebenen Arbeiten fallen daher nicht unbedingt in das Arbeitsgebiet jedes Entwicklers, sondern sind durch einen Systemadministrator auszuführen. Die Prüfungen innerhalb der Berechtigungspflege sind außerdem so organisiert, dass die Teilaufgaben Erstellen und Freigeben von Berechtigungen zwei verschiedenen Administratoren übertragen werden können. Damit ist bei sicherheitsrelevanten Änderungen ein Vier-Augen-Prinzip gewährleistet. Abhängig von Ihren konkreten Rechten innerhalb des Systems können Sie das folgende Beispiel möglicherweise nicht ausführen. In diesem Fall bieten Ihnen die Erläuterungen in Verbindung mit den Bildschirmkopien zumindest einen Überblick über die Elemente des Berechtigungssystems.

Das Beispiel erläutert den gesamten Vorgang, beginnend mit dem Anlegen eines Berechtigungsfelds bis hin zum Einfügen einer Berechtigung in einen Benutzerstammsatz, anhand einer Berechtigung für das Demo-Programm in Kapitel 7. Diese Beschreibung entspricht den eben genannten drei Punkten, allerdings in umgekehrter Reihenfolge. Für die Demonstration ist ein als Kundensystem eingerichtetes SAP-System erforderlich. Daher wurde auch hier ein 4.5A-System eingesetzt. Abgesehen davon, dass in neueren Versionen weitere Transaktionen zur Berechtigungspflege zur Verfügung stehen (Profilgenerator `PFCG`), existieren keine wesentlichen Unterschiede.

6.2.1 Berechtigungsklassen, -felder und -objekte

Ein Berechtigungsobjekt ist eine Vorlage für eine Berechtigung. Das Berechtigungsobjekt enthält Felder, die später in der Berechtigung mit Werten gefüllt werden. Ein Berechtigungsobjekt ist daher entfernt mit einer Typdeklaration vergleichbar. Mehrere Berechtigungsobjekte einer Anwendungsgruppe (z.B. Rechnungswesen oder Materialwirtschaft) bilden eine Berechtigungsklasse. Diese dient lediglich zur logischen Gruppierung der Berechtigungsobjekte und damit zur Verbesserung der Übersichtlichkeit.

Alle Felder, die Sie in Berechtigungsobjekte aufnehmen möchten, müssen in der Tabelle AUTHX gepflegt werden. Im Gegensatz zu früheren Versionen, in denen die Berechtigungsfelder als echtes Feld in einer von drei Dictionary-Strukturen angelegt wurden, werden sie nun durch einen Datensatz in der Tabelle AUTHX beschrieben. Der Inhalt dieser Tabelle wird über die Transaktion SU20 gepflegt. Vom SAP-Menü aus erfolgt der Aufruf über den Menüzweig ENTWICKLUNG | WEITERE WERKZEUGE | BERECHTIGUNGSOBJEKTE | SU20 – FELDER.

6.2.2 Berechtigungsgruppen

Die explizite Berechtigungspflege kann bei unsachgemäßem Einsatz Sicherheitslücken zur Folge haben. Aus diesem Grund führt das R/3-System einige Prüfungen automatisch aus, ohne dass diese innerhalb einer Anwendung programmiert werden müssen. Überwacht wird beispielsweise der Aufruf von Transaktionen zur Tabellenpflege oder die Modifikation von Anwendungen.

Grundlage für diese Prüfungen sind so genannte Berechtigungsgruppen. Eine Berechtigungsgruppe definieren Sie durch einen Eintrag in der Tabelle TPGP. Die Tabelle TPGPT enthält die Kurztexte zu den Berechtigungsgruppen. Sie legen in der Tabelle TPGP einen achtstelligen Bezeichner an. Dies erfolgt getrennt für jedes Anwendungsgebiet. Dieser Bezeichner ist der Name einer Berechtigungsgruppe. Diesen Namen können Sie bei der Pflege diverser Objekte als zusätzliches Attribut angeben. Geprüft wird vor dem Starten oder Editieren eines Programms gegen das Berechtigungsobjekt S_DEVELOP. Die Pflege der Programmattribute bzw. der Zugriff anderer Werkzeuge der Entwicklungsumgebung ist durch das S_PROGRAM geschützt.

Ein Anwender hat das Recht zum Zugriff auf das Objekt dann, wenn er über eine Berechtigung verfügt, die vom Berechtigungsobjekt S_DEVELOP bzw. S_PROGRAM der Berechtigungsklasse BC_C (Basis Entwicklungsumgebung) abgeleitet wurde, und die im Feld P_GROUP den Namen der Berechtigungsgruppe enthält.

6.2.3 Schutz von Transaktionen

Ab Release 3.0E müssen Sie alle Transaktionen, auf die ein Anwender Zugriff haben soll, explizit freigeben. Sie erreichen dies durch Eintragen des Transaktionscodes (oder eines passenden Musters) im Feld TCD des Berechtigungsobjekts S_TCODE.

6.2.4 Ein Beispiel

Um eigene Berechtigungsobjekte zu erzeugen, sind bis zu 3 verschiedene Arbeitsschritte erforderlich:

- Definieren der Berechtigungsfelder
- Anlegen einer Berechtigungsklasse
- Anlegen von Berechtigungsobjekten

Die für diese Aufgaben erforderlichen Werkzeuge können vom Hauptmenü der Entwicklungsumgebung mit den Menüfunktionen ENTWICKLUNG | WEITERE WERKZEUGE | BERECHTIGUNGSOBJEKTE | FELDER (Transaktionscode SU20) bzw. ENTWICKLUNG | WEITERE WERKZEUGE | BERECHTIGUNGSOBJEKTE | OBJEKTE (Transaktionscode SU21) erreicht werden.

Zunächst soll ein Berechtigungsfeld angelegt werden. Dazu dient die Transaktion SU20 bzw. die bereits genannte Menüfunktion. Diese Transaktion listet im Grundbild zunächst alle Berechtigungsfelder auf (Abbildung 6.6).

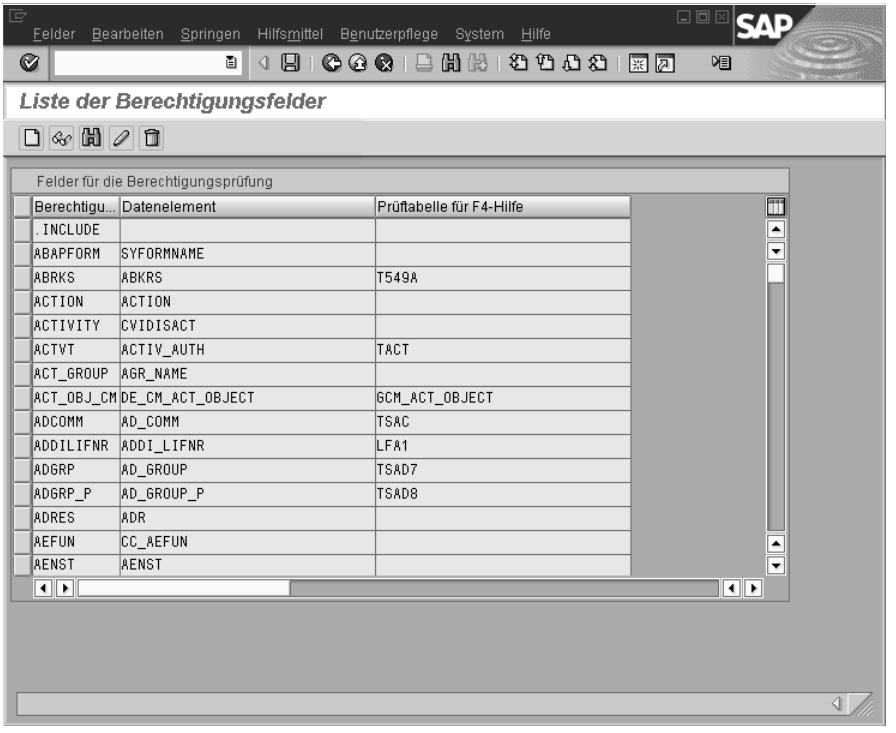

Abbildung 6.7 **© SAP AG**
Neues Berechtigungsfeld definieren

Innerhalb des Startbilds kann über die Menüfunktion FELDER | ANLEGEN ein neues Feld erzeugt werden. Beim Anlegen werden in einem Popup (Abbildung 6.7) lediglich zwei Angaben erfragt.

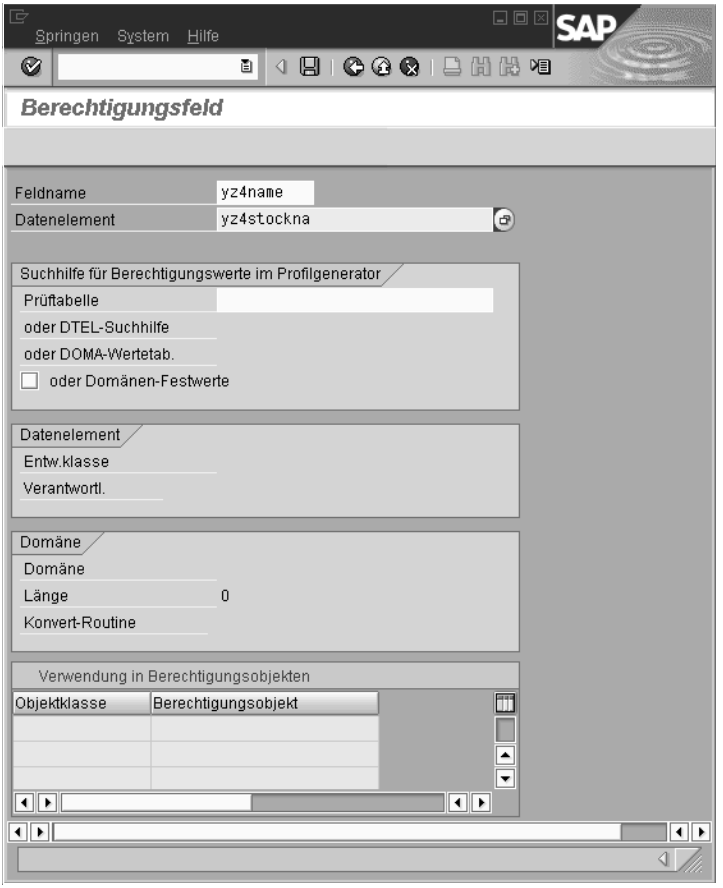

Abbildung 6.8 **© SAP AG**
Neues Berechtigungsfeld anlegen

Es handelt sich um den gemäß den Namenskonventionen frei wählbaren Namen des Berechtigungsfelds und den Namen des Datenelements, das die Eigenschaften des Berechtigungsfelds bestimmt. In diesem Beispiel dient `YZ4NAME` als Name und `YZ4STOCKNA` als Datenelement. Die drei Anzeigefelder im Popup werden erst beim Sichern der Angaben entsprechend den Eigenschaften des Datenelements aktualisiert. Diese Angaben erscheinen auch in der Liste. Auf die beschriebene Weise können natürlich auch weitere Felder angelegt werden. Für das Beispiel soll dieses eine Feld genügen.

Im zweiten Schritt wird eine neue Berechtigungsklasse angelegt. Die Verfahrensweise entspricht im Wesentlichen der bei der Erzeugung eines neuen Feldes. Zunächst wird die Transaktion `SU21` entweder durch direkte Eingabe des Transaktionscodes oder durch die zugehörige Menüfunktion gestartet. Eine

neue Klasse erzeugen Sie mit einer Drucktaste oder einer Menüfunktion. Die Angaben für die neue Klasse (Bezeichnung und Beschreibung) werden in einem Popup (Abbildung 6.8) erfasst.

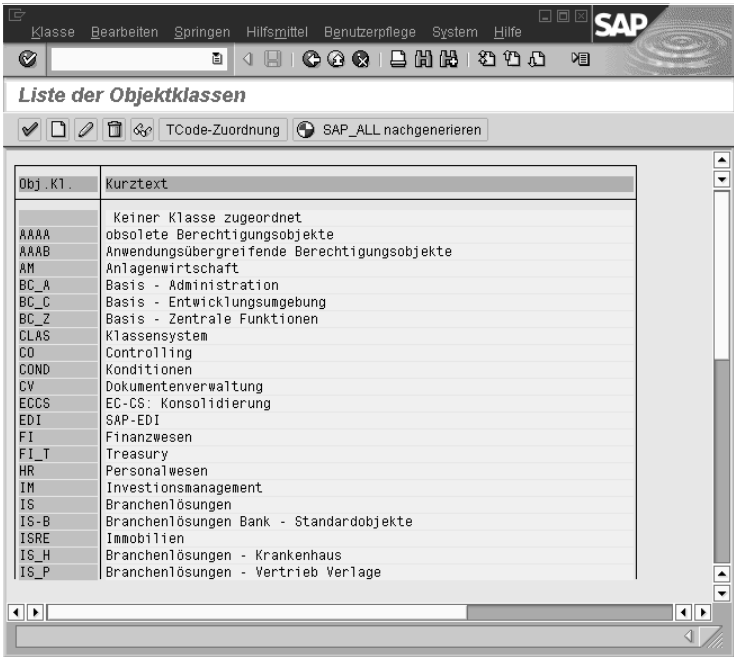

Abbildung 6.9 **© SAP AG**
Berechtigungsklasse anlegen

Nach dem Anlegen der Klasse erscheint diese in der Liste der verfügbaren Klassen. Von dort aus können durch einen Doppelklick auf den Klassennamen oder die Menüfunktion KLASSE | LISTE OBJEKTE die Objekte in einer Klasse bearbeitet werden. Diese Objekte werden ebenfalls in Form einer Liste angezeigt, in die Sie neue Objekte per Menüfunktion oder Drucktaste aufnehmen können. Das Popup zur Definition eines Berechtigungsobjekts erfordert die Angabe des Objektnamens und einer Kurzbeschreibung (Abbildung 6.10).

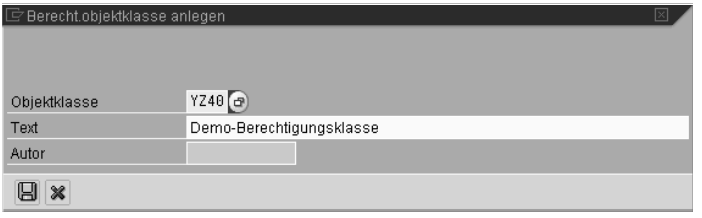

Abbildung 6.10 **© SAP AG**
Berechtigungsobjekt anlegen

Neben diesen Angaben können Sie im unteren Bereich des Popups bis zu zehn Berechtigungsfelder eintragen (Abbildung 6.9). Diese Felder müssen zu diesem Zeitpunkt bereits existieren.

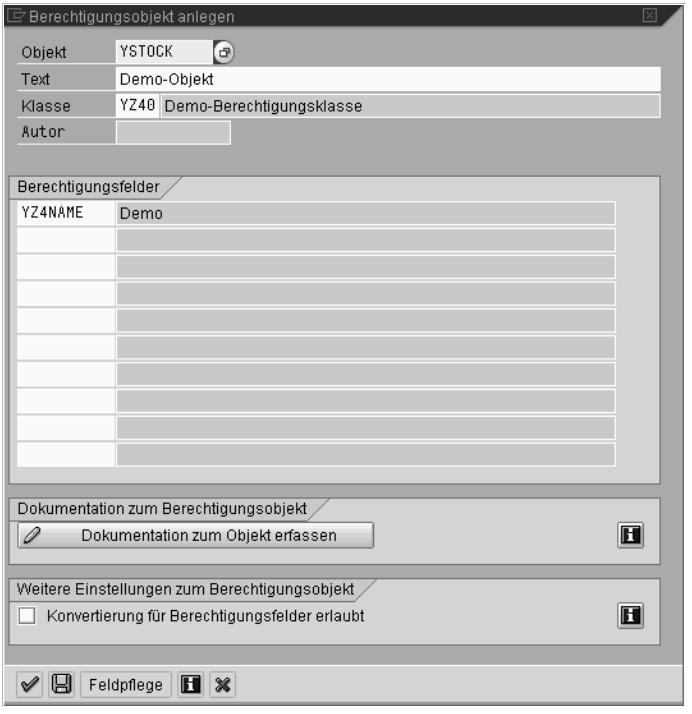

Abbildung 6.11 **© SAP AG**
Berechtigungsfelder einfügen

Mit der Definition des Berechtigungsobjekts ist der erste Teil der zur Berechtigungspflege erforderlichen Arbeiten abgeschlossen.

Das Berechtigungsobjekt kann nun benutzt werden, um eine Berechtigung zu definieren. Eine Berechtigung ist eine Ausprägung eines Berechtigungsobjekts. Sie erhält einen eigenen Namen und enthält alle Felder des Berechtigungsobjekts, wobei den Feldern aber Werte zugewiesen werden können. Gegen diese Werte erfolgt dann später die Berechtigungsprüfung.

Es existieren verschiedene Möglichkeiten, die Berechtigungspflege aufzurufen. In den verschiedenen Dynpros der Transaktionen SU20 und SU21 steht das Menü BENUTZERPFLEGE mit den drei Funktionen BERECHTIGUNGEN, PROFILE und BENUTZER zur Verfügung. Eben dieses Menü ist auch im Bereichsmenü der Systemadministration enthalten, das vom Grundmenü des SAP-Systems aus (Bereichsmenü S000) mittels ADMINISTRATION | SYSTEMVERWALTUNG | BENUTZERPFLEGE aktiviert werden kann. Die Transaktionscodes für die drei Menüfunktionen sind SU01 (Benutzer), SU02 (Profile) und SU03 (Berechtigungen).

Um eine Berechtigung anzulegen, wird die entsprechende Transaktion (SU03) aufgerufen. Es erscheint zunächst eine Liste mit allen verfügbaren Berechtigungsklassen. Dabei wird allerdings auch die verbale Beschreibung angezeigt und nach dieser sortiert. Dadurch ändert sich im Vergleich zur Liste der Transaktion SU21 die Reihenfolge der Einträge. Ein Doppelklick auf diesen Eintrag verzweigt in eine Liste, in der alle Berechtigungsobjekte dieser Klasse aufgelistet werden.

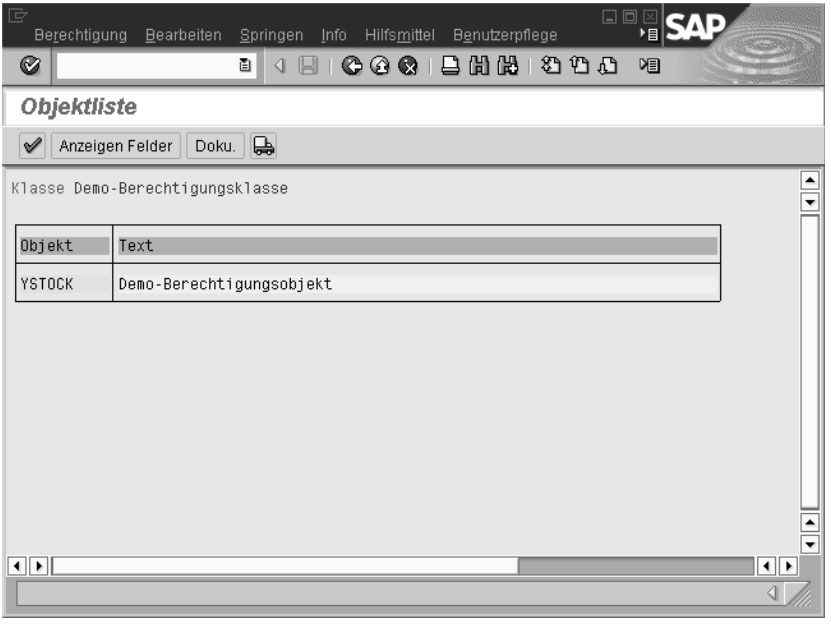

Abbildung 6.12 **© SAP AG**
Objektliste der neuen Berechtigungsklasse

Alle von einem Berechtigungsobjekt abgeleiteten Berechtigungen werden in einer dritten Liste sichtbar, die wiederum nach einem Doppelklick auf den Namen des Berechtigungsobjekts erscheint. In unserem Fall ist diese Liste noch leer. Mittels der Drucktaste ANLEGEN oder der Menüfunktion BERECHTIGUNG | ANLEGEN wird eine Berechtigung erzeugt. Es erscheint ein Popup, in dem die Bezeichnung der Berechtigung und eine Kurzbeschreibung eingetragen werden müssen. Da eine Berechtigung konkrete Werte enthält und damit genau definierte Aktionen erlaubt oder verbietet, kann sie einen aussagekräftigen Namen erhalten. Die erste Berechtigung (siehe Abbildung 6.10) soll den Namen Y:AKT_ALL erhalten. Beachten Sie bitte, dass gemäß den SAP-Namenskonventionen in den Namen von Kundenberechtigungen an der zweiten Position kein Unterstrich stehen darf. Die Kurzbeschreibung können Sie nach Belieben wählen.

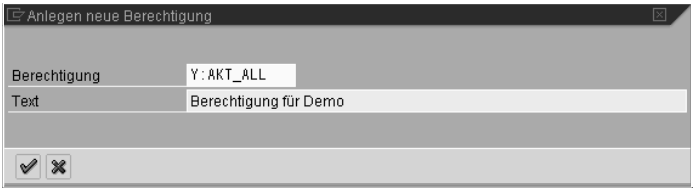

Abbildung 6.13 **© SAP AG**
Pflegen einer Berechtigung

Die so erzeugte Berechtigung erscheint dann in der Liste der Berechtigungen, ist aber noch leer. Die Zuweisung von Werten erfolgt mittels der Drucktaste WERTE EINPFLEGEN, der Menüfunktion SPRINGEN | WERTE EINPFLEGEN oder einem Doppelklick auf den Namen der Berechtigung. In dem nun aktiven Popup (Abbildung 6.11) können ein oder mehrere Einzelwerte oder Wertpaare erfasst werden.

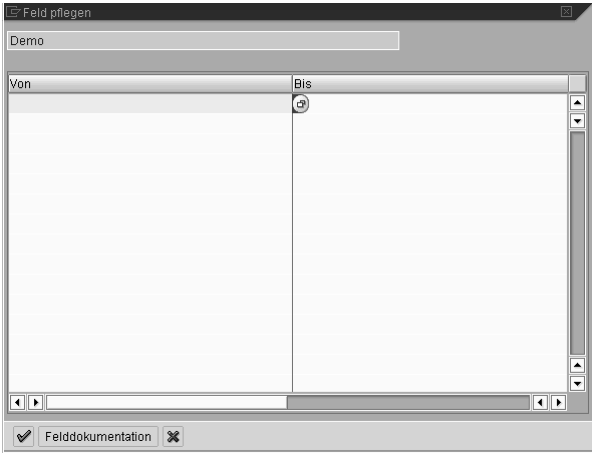

Abbildung 6.14 **© SAP AG**
Zuweisen von Werten zu einer Berechtigung

Nach dem Festlegen der Feldwerte muss die Berechtigung gesichert und aktiviert werden. Erst nach einer Aktivierung werden die eingepflegten Werte auch wirksam. Der Status der Berechtigung (überarbeitet, aktiv) wird durch eine Ausschrift in der Liste mit den Berechtigungen gekennzeichnet. Mitunter werden die Berechtigungen für Pflege und Aktivierung auf zwei verschiedene Administratoren aufgeteilt, so dass Änderungen des einen durch eine zweite Person bestätigt werden müssen, bevor sie wirksam werden. Während der Überarbeitung einer existierenden Berechtigung werden daher zwei Versionen gespeichert: die aktuell für Anwender gültige Version und eine Pflegeversion. Erst durch die Aktivierung wird die Pflegeversion zur aktuellen Version. Damit dem aktivierenden Administrator die Ausübung seiner Kontrollfunktion erleichtert wird, stellt das System bei einer Aktivierung die aktuelle und die Pflege-

version gegenüber. Erst nach nochmaliger Ausführung der Funktion AKTIVIEREN erfolgt die endgültige Aktivierung.

Zu einem Berechtigungsobjekt können beliebig viele Berechtigungen mit unterschiedlichen Feldwerten angelegt werden. Um die konkreten Rechte eines Benutzers festzulegen, muss in dessen Benutzerstammsatz die Berechtigung hinterlegt werden. Dies ist nicht auf direktem Wege möglich, sondern nur unter Zwischenschaltung so genannter Profile. Ein Profil fasst eine oder mehrere logisch zusammengehörende Berechtigungen zusammen und erleichtert so die Pflege der Benutzerdaten. Die direkte Zuweisung einer Berechtigung zum Benutzerstammsatz ist nicht möglich.

Beginnend mit den letzen 4.6-Versionen wurde neben der Pflege der Profile ein Rollenkonzept eingeführt. Dieses löst schrittweise die manuelle Pflege der Profile ab. Grundprinzip des Rollenkonzeptes ist es, zunächst Berechtigungen zu erstellen und neben anderen Angaben in einer Rolle zusammenzufassen. Dieser Rolle werden dann die Benutzer zugeordnet. Eine Rolle enthält mehr Angaben als die eigentlichen Berechtigungen. Allerdings sollen die zusätzlichen Eigenschaften hier nicht erwähnt werden.

Aufgerufen wird die Pflege der Rollen über den Transaktionscode PFCG. Abbildung 6.15 zeigt das Startbild dieser Transaktion.

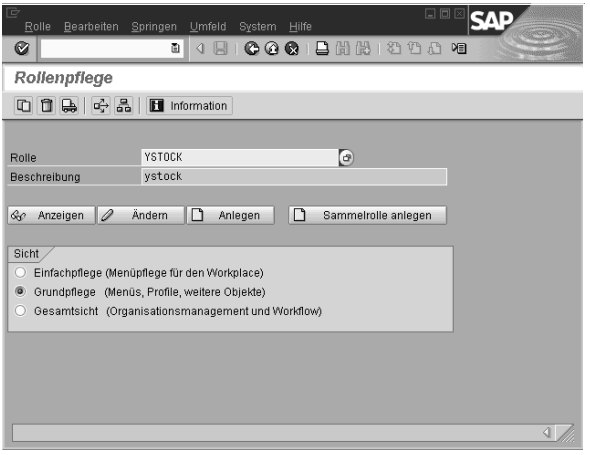

Abbildung 6.15 **© SAP AG**
Einstiegsbild der Rollenpflege

Im Startbild werden der Name und die Beschreibung der anzulegenden Rolle eingetragen und anschließend die Drucktaste ANLEGEN betätigt. Daraufhin erscheint das Pflegebild der Transaktion PFCG (Abbildung 6.16).

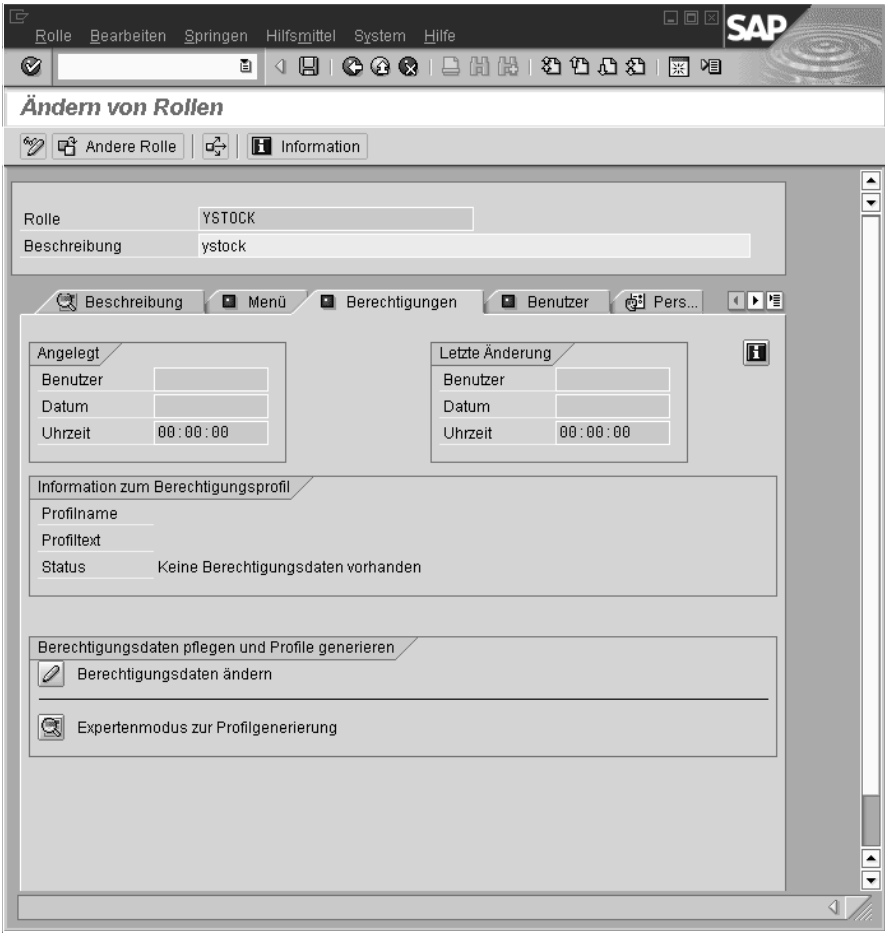

Abbildung 6.16 **© SAP AG**
Pflegebild der Transaktion PFCG

Innerhalb der Pflegeoberfläche wählen Sie nun die Registerkarte BERECHTIGUNGEN. Auf diesem Dynpro stehen im unteren Drittel zwei Drucktasten zur Verfügung, die beide zur Pflege der Berechtigungen verzweigen. Benutzen Sie für dieses Beispiel die Schaltfläche BERECHTIGUNGSDATEN ÄNDERN. Ein Popup (Abbildung 6.17) ermöglicht es Ihnen, vorgefertigte Templates zu übernehmen. In diesem Beispiel sind diese Vorlagen nicht erforderlich, sie können das Popup mit der Schaltfläche KEINE VORLAGE AUSWÄHLEN beenden.

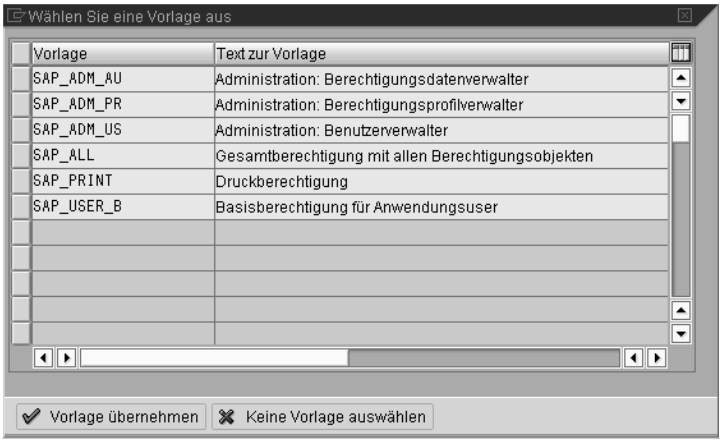

Abbildung 6.17 **© SAP AG**
Auswahl von Vorlagen für Berechtigungen

Im darauf folgenden Dynpro (Abbildung 6.18) finden Sie einen Eintrag mit dem Namen der Rolle vor. Um dieser Rolle Berechtigungen zuzuordnen, betätigen Sie die Drucktaste MANUELL oder benutzen die Tastenkombination Ctrl + ⇧ + F9.

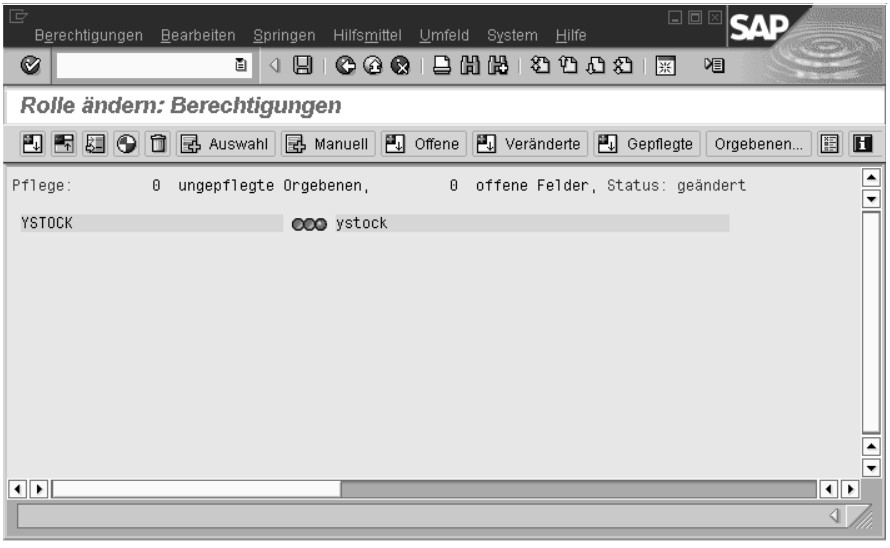

Abbildung 6.18 **© SAP AG**
Pflegebild für die Berechtigungen einer Rolle

In einem Popup (Abbildung 6.19) können Sie nun die gewünschten Berechtigungsobjekte erfassen. Sie können die Bezeichner direkt eintragen oder aber die Eingabehilfe benutzen.

Abbildung 6.19 **© SAP AG**
Berechtigungsobjekte erfassen.

Die so erfassten Berechtigungsobjekte werden vom System unter der Rolle eingefügt. Abbildung 6.20 zeigt das Berechtigungs-Pflegebild mit der eingefügten Berechtigung. Es ist zu sehen, dass nicht nur das eigentliche Berechtigungsobjekt, sondern auch die übergeordneten Elemente bis hin zur Berechtigungsklasse aufgeführt werden. Bei der Pflege komplexerer Berechtigungsstrukturen ist so ein wesentlich besserer Überblick über die verwendeten Einstellungen möglich als zuvor.

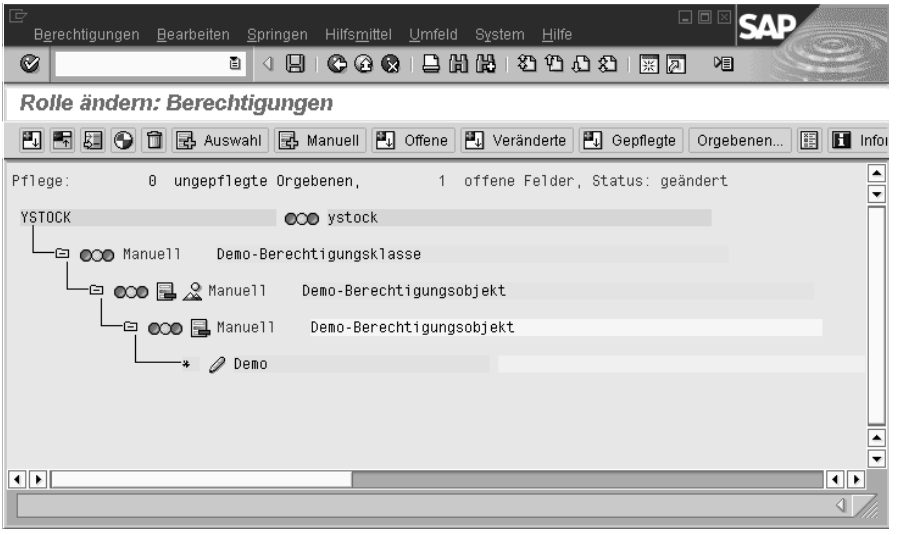

Abbildung 6.20 **© SAP AG**
Pflege von Berechtigungsstrukturen

Beim Abspeichern der Änderungen fasst das System die eingefügten Berechtigungen zu einem Profil zusammen. Für dieses Profil wird ein Namensvorschlag generiert, der an dieser Stelle nochmals abgeändert werden könnte (Abbildung 6.21).

Abbildung 6.21 **© SAP AG**
Erstellen eines Berechtigungsprofils

Nach dem Speichern der Einstellungen kann die Rolle generiert werden. Dies erfolgt am einfachsten über das Generieren-Symbol in der Drucktastenzeile. Abbildung 6.22 zeigt die Anwendung nach dem Generieren der Rolle.

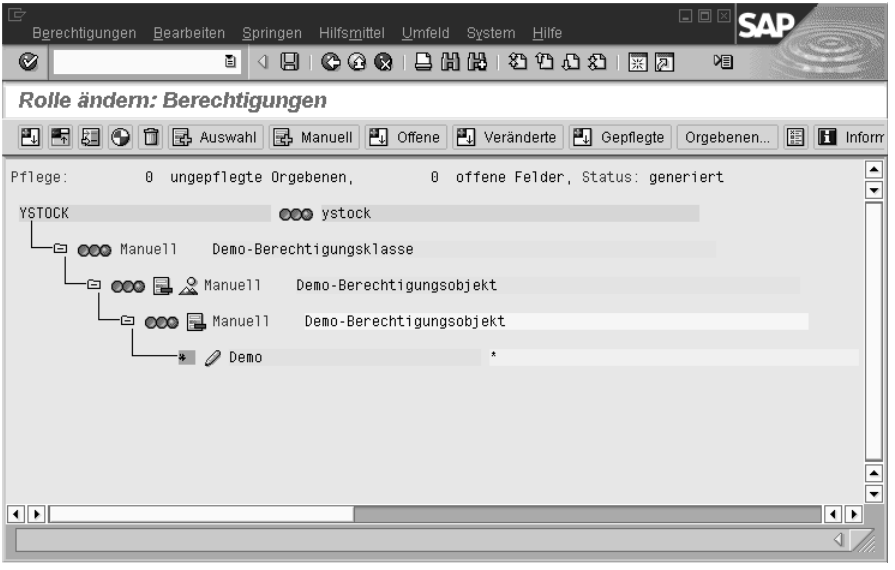

Abbildung 6.22 **© SAP AG**
Pflegeoberfläche nach dem Generieren

Nach der Pflege der Berechtigungen können Sie in das Grundbild der Rollen-Pflege zurückkehren. Dort könnten Sie nun auf der Registerkarte BENUTZER die Namen der Anwender eintragen, die zur eben erstellten Rolle gehören sollen.

Ein Blick in den Benutzerstammsatz zeigt, dass dort sowohl die Rolle als auch das generierte Profil enthalten sind (Abbildung 6.23 und 6.24). Diese Einstellungen sind bezüglich der Berechtigungen redundant.

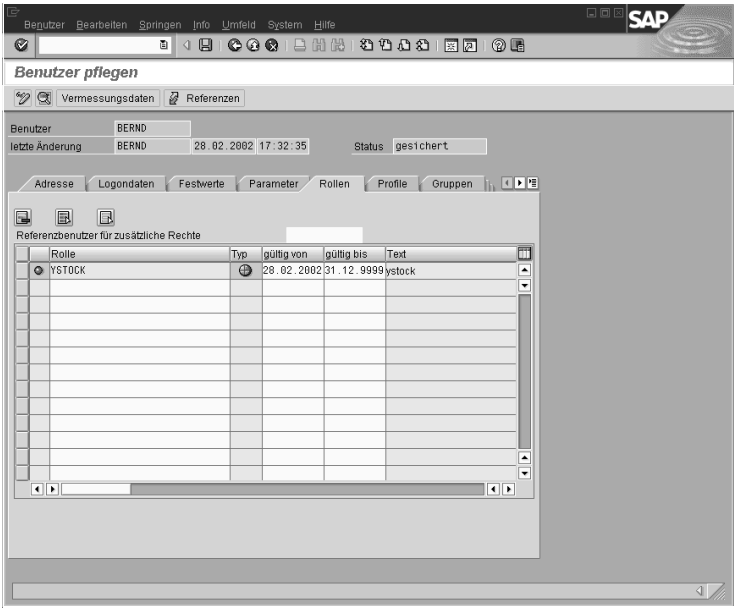

Abbildung 6.23 **© SAP AG**
Benutzerstammsatz: Rollen eines Benutzers

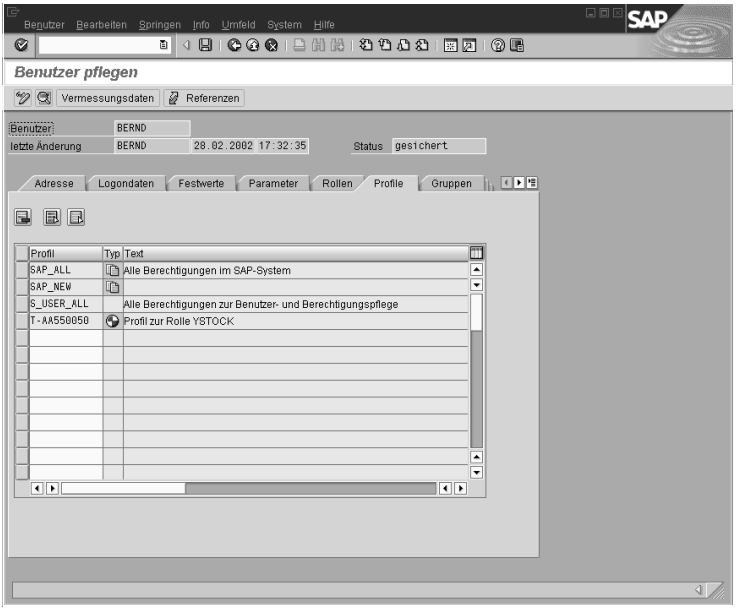

Abbildung 6.24 **© SAP AG**
Benutzerstammsatz: Profile

7 Ein Beispiel

Dieser Abschnitt veranschaulicht mit Hilfe eines kleinen Beispiels die grundlegende Programmiertechnik beim Erstellen einer Dialoganwendung. Dabei liegt der Schwerpunkt nicht auf der Bedienung der einzelnen Werkzeuge oder der Tätigkeiten bei der Erstellung der Anwendung, sondern auf der Programmstruktur und der Beschreibung der Aufgaben der einzelnen Programmteile. Für viele Anwendungen im SAP-System wird eine relativ einheitliche Programmstruktur benutzt. Diese Struktur hat sich im Laufe der Zeit herausgebildet. Sie ermöglicht eine klare Programmierung, eine relativ einheitliche Bedienung sowie die einfache Verknüpfung von Transaktionen. Dieses Beispiel ist relativ unabhängig von einer konkreten Version der R/3-Software. Das Beispiel ermöglicht die Erfassung von Aktienkursen. Dabei werden in der Tabelle `YZ3STOCK` zunächst die Aktien gepflegt, zu denen Kurse erfasst werden sollen. Die Kurse stehen in einer eigenen Tabelle `YZ3PRICE`. Abbildung 7.1 zeigt die Beziehungen der Tabellen. Dabei wird eine bei SAP übliche Entity-Relationship-Darstellung benutzt.

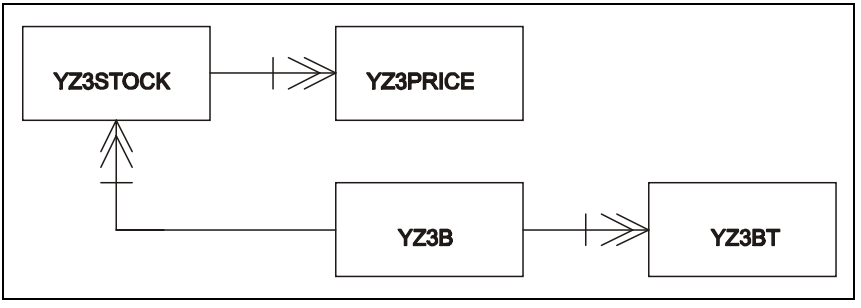

Abbildung 7.1
Beziehungen der Tabellen der Demoanwendung

Da in diesem Abschnitt nicht auf die in den vorangegangenen Abschnitten erzeugten Objekte zurückgegriffen werden soll, müssen Sie zunächst die benötigten Tabellen anlegen. Tabelle 7.1 zeigt Ihnen den Aufbau. Der besseren Übersicht wegen wurden in der Tabelle für die Felder, für die im R/3-System kein vordefiniertes Datenelement existiert, nur die Feldnamen und Datentypen aufgeführt. Allerdings sollten Sie beim Erstellen der Tabellen die Typisierung der Felder per Datenelement berücksichtigen und diese selbstständig anlegen. Die Feldbezeichner können Sie bei der Datenelementpflege nach freiem Ermessen vergeben. Die Beschreibung der Aufgabe des Feldes kann dabei als Anhaltspunkt dienen.

Tabelle	Feldname	Schlüssel	vordefiniertes Datenelement	Typ	Länge	Prüftabelle	Beschreibung
YZ3B	MANDT	X	CLNT				
	BRANCH	X		CHAR	1		Schlüssel für Branche
YZ3BT	MANDT	X	CLNT				
	BRANCH	X		CHAR	1	YZ3B	Schlüssel für Branche
	LANGU	X	LANGU				
	DESCR			CHAR	30		Bezeichnung der Branche
YZ4STOCK	MANDT	X	CLNT				
	WKZ	X		NUMC	6		Kennzahl der Aktie
	NAME			CHAR	30		Bezeichnung
	BRANCH			CHAR	1		Branche
	CURRENCY		WAERS_CURC			TCURC	Währungsschlüssel

Tabelle 7.1
Aufbau der benötigten Tabellen

Tabelle	Feldname	Schlüssel	vordefiniertes Datenelement	Typ	Länge	Prüftabelle	Beschreibung
YZ4PRICE	MANDT	X	CLNT				
	WKZ	X		NUMC	6	YZ3STOCK	Kennzahl der Aktie
	AKTDATE	X		DATS			Kursdatum
	PRICE			CURR	8		Kurs

Tabelle 7.1
Aufbau der benötigten Tabellen (Fortsetzung)

Für die Tabellen `YZ3B` und `YZ3BT` sollte eine Tabellenpflege generiert werden. Dabei können Sie sich an den Ausführungen im Kapitel 4 orientieren. Über das im Folgenden beschriebene Programm werden lediglich die Tabellen `YZ3STOCK` und `YZ3PRICE` bearbeitet. Die Anwendung für `YZ3STOCK` soll das Anlegen, Ändern, Anzeigen und Löschen von Datensätzen ermöglichen. Dabei ist das Löschen als Unterfunktion des Programmzweiges Ändern realisiert. Die zweite Anwendung gestattet das Erfassen oder Editieren von Kursdaten zu allen in `YZ3PRICE` gepflegten Datensätzen. Funktionen zum einfachen Anzeigen oder Löschen sind hier nicht vorgesehen. Da nur Kurse für Aktien gepflegt werden können, für die in `YZ3STOCK` ein Datensatz existiert, muss mit der Anwendung für `YZ3STOCK` begonnen werden.

7.1 Programmstruktur

Eine Anwendung besteht aus einem oder mehreren Dynpros, die nacheinander aufgerufen werden. Jedes Dynpro erfüllt zusammen mit den Anweisungen der Ablauflogik eine genau definierte Teilaufgabe. Der Übergang von einem Dynpro zum anderen bedeutet, dass eine Teilaufgabe abgeschlossen wurde, und dafür eine andere ausgeführt werden soll. Jedes Dynpro stellt somit einen Zustand innerhalb der Anwendung dar. Die erste Aufgabe beim Entwurf einer Anwendung ist es daher, die auszuführende Aktion in Teilaufgaben zu zerlegen, diese den verschiedenen Dynpros zuzuordnen und festzulegen, unter welchen Bedingungen und anlässlich welcher Aktionen die unterschiedlichen Dynpros bearbeitet werden sollen. Für die Pflege von `YZ3STOCK` ist die Festlegung der Teilaufgaben relativ einfach. Es existieren nur zwei:

- Datensatz selektieren
- Datensatz bearbeiten

Jede dieser Aufgaben wird einem Dynpro übertragen. Bei komplexeren Anwendungen kann es durchaus notwendig sein, die oben genannten Aufgaben weiter zu untergliedern und damit die Zahl der Dynpros zu erhöhen. Für das sehr einfach gehaltene Beispiel reichen aber zwei Dynpros aus.

Nachdem Zahl und Aufgabe der Dynpros festgelegt wurden, müssen die möglichen Übergänge von einem Dynpro zum anderen definiert werden. Derartige Übergänge sind nur im PAI-Teil eines Dynpros sinnvoll. Dieser Abschnitt wird aber erst abgearbeitet, wenn der Anwender im Dynpro eine Funktion ausgelöst hat. Es ist also zu entscheiden, welche Funktionen und Funktionscodes in einem Dynpro zur Verfügung stehen, und was sie bewirken sollen. In diesem Zusammenhang spielen sowohl grundlegende Forderungen des SAP-Style-Guide als auch anwendungsspezifische Anforderungen eine Rolle. Das SAP-Style-Guide fordert beispielsweise systemweit einheitliche Reaktionen auf die Kommandos ZURÜCK (Funktionstaste F3), ABBRECHEN (Funktionstaste F12) und BEENDEN (Funktionstaste F15). Nicht jedes dieser Kommandos ist in jedem Dynpro notwendig, wenn es aber genutzt wird, sollte die Reaktion der Anwendung für den Benutzer vorhersehbar sein.

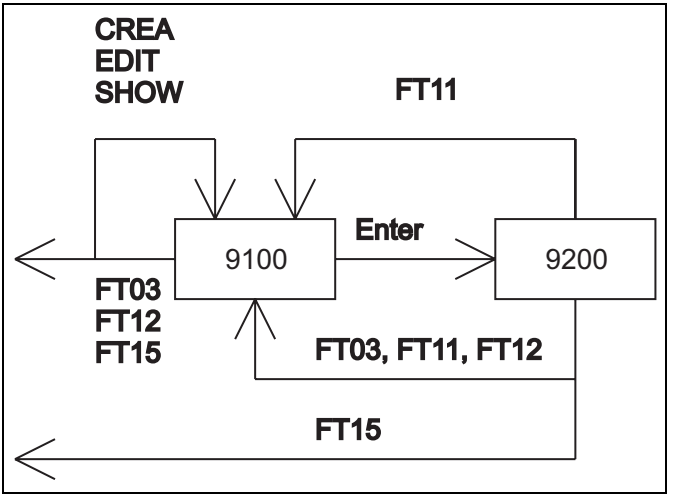

Abbildung 7.2
Navigation zwischen den Dynpros

In Abbildung 7.2 ist die Navigationsstruktur der Demoanwendung dargestellt. Diese Grafik erfordert einige Erläuterungen. Es ist unüblich, für jede der Teilaufgaben Anlegen, Ändern, Anzeigen und Löschen eine eigenständige Anwendung zu schreiben. Vielmehr wird innerhalb einer einzelnen Anwendung anhand des Transaktionscodes in verschiedene Programmteile verzweigt. Das Grundbild einer Anwendung stellt daher oft einige Menüeinträge bereit, mit denen von einer Grundfunktionalität der Anwendung zu einer anderen gewech-

selt werden kann. Diese Funktionscodes führen zum Neuaufruf der Anwendung über einen entsprechenden Transaktionscode. Des Weiteren muss im Grundbild einer Anwendung mindestens eine Möglichkeit zum Beenden zur Verfügung stehen.

Um vom Grundbild aus nach der Eingabe der Selektionskriterien in das Bearbeitungsbild zu gelangen, wird die Datenfreigabetaste (Enter-Taste) benutzt. Auch diese Eigenschaft wird systemweit einheitlich realisiert, damit beim Aufruf einer Transaktion mit `CALL TRANSACTION` der Zusatz `AND SUBMIT FIRST SCREEN` korrekt funktioniert.

Im eigentlichen Bearbeitungsbild führt die Datenfreigabetaste üblicherweise zur Prüfung der eingegebenen Daten und eventuell zum Aufruf eines weiteren Dynpros zur Datenerfassung. Da das Beispiel nicht über ein weiteres Dynpro verfügt, kehrt im Bearbeitungsbild die Datenfreigabetaste also immer wieder zum Bearbeitungsbild zurück. Das Sichern von Daten beendet einen Bearbeitungsvorgang. Es ist nicht notwendig, nach dem Sichern im Bearbeitungsbild zu bleiben, es kann also zum Grundbild der Anwendung zurückgekehrt werden.

Auch das Bearbeitungsbild muss ohne Datenprüfung und Sicherung verlassen werden können. Dazu stehen auch hier die bereits erwähnten Funktionen ZURÜCK, ABBRECHEN und BEENDEN zur Verfügung. Während im Grundbild diese drei Funktionen aber einheitlich zur Beendigung der Anwendung führten, ist laut Style-Guide in einem Bearbeitungsbild eine Differenzierung der Funktionalität erforderlich. Die Funktion ABBRECHEN soll nach Möglichkeit zum vorhergehenden Dynpro zurückkehren. Falls im aktuellen Dynpro Daten eingegeben wurden, ist der Anwender auf den möglichen Datenverlust hinzuweisen. Er sollte die Gelegenheit bekommen, die ABBRUCH-Funktion zu unterbinden. Die Funktion ZURÜCK soll ebenfalls zum vorangegangenen Dynpro wechseln, wobei der Anwender allerdings geänderte Daten sichern kann. Dieses Sichern soll auch bei der Funktion BEENDEN möglich sein, wobei diese aber nicht nur das aktuelle Dynpro, sondern die gesamte Anwendung beendet.

Wenn ein Datensatz gelöscht werden soll, ist es ebenfalls nicht sinnvoll, die aktuellen Daten im Dynpro zu prüfen. Das Löschen kann daher im Exit-Modul erfolgen, wobei aber auch eine Sicherheitsabfrage erforderlich wird. Nach dem Löschen eines Satzes muss natürlich zum Grundbild zurückgekehrt werden.

7.2 Programm 1: SAPMYZ3S

Die Beschreibung der Programmstruktur ist Voraussetzung für die Analyse des Quelltextes. Da eine Anwendung aber nicht nur aus dem eigentlichen Quelltext, sondern aus weiteren Elementen besteht, folgt hier eine Beschreibung dieser Elemente:

Die Anwendung kann mit den drei Dialogtransaktionen `YZ3I` zum Anlegen (Insert), `YZ3U` zum Ändern (Update) und `YZ3S` zum Anzeigen (Show) aufgerufen werden. Alle drei führen das Dynpro 9100 des Modul-Pools `SAPMYZ3S` aus.

Das Dynpro 9100 enthält das Feld YZ3STOCK-WKZ als Muss-Eingabefeld. Da ein derartiges Feld keinen Wert akzeptiert, der nur aus Leerzeichen besteht, ist sichergestellt, dass keine namenlosen Datensätze erzeugt werden können.

Die Tabelle TPARA enthält alle Get/Set-Parameter. In dieser Tabelle wurde ein Parameter YZS definiert. Zur Pflege dient die Transaktion SM31. Bei der Pflege des Dynpro 9100 muss der Get/Set-Mechanismus für das einzige Eingabefeld aktiviert werden. Außerdem muss der Parameter YZS im Datenelement YZ3WKZ als Attribut PARAMETER-ID des Datenelementes eingetragen werden.

Dynpro 9200 ist in Abbildung 7.3 dargestellt. Diese Abbildung zeigt auch die Feldnamen der Dynprofelder. In diesem Dynpro sind nur die Felder YZ3STOCK-NAME, YZ3STOCK-BRANCH und YZ3STOCK-WAEHR Eingabefelder. Alle anderen sind Anzeigefelder, bei denen zum Teil auch das Attribut ZWEIDIMENSIONAL gesetzt wurde.

Beide Dynpros legen den Funktionscode im Feld FCODE ab.

Es existieren zwei Status, STAT_G für das Grundbild (9100) und STAT_A für das Arbeitsdynpro (9200). Die in diesen Status erforderlichen Funktionscodes und deren Typ ergeben sich im Prinzip bereits aus Abbildung 7.2. Alle Funktionscodes, deren Grafen nach links bzw. unten aus einem Dynpro herausführen (FT03, FT12, FT14, FT15, CREA, EDIT, SHOW), sind Exit-Funktionscodes, alle anderen sind einfache Funktionscodes. Die Zuordnung der Funktionscodes zu den Symbolen der Symbolleiste ist durch deren Aufgabe bereits vorgegeben. Sie muss allerdings manuell vorgenommen werden. Nur für den Funktionscode FT14 zum Löschen wird eine Drucktaste definiert.

Es existiert jeweils ein Sperrbaustein für die Tabelle YZ3STOCK und YZSPRICE.

Im Deklarationsteil des Modul-Pools wurden einige globale Datenfelder und Konstanten deklariert (z.B. C_TRUE, C_FALSE etc.). Den Quelltext des Deklarationsteils finden Sie am Ende des Abschnitts.

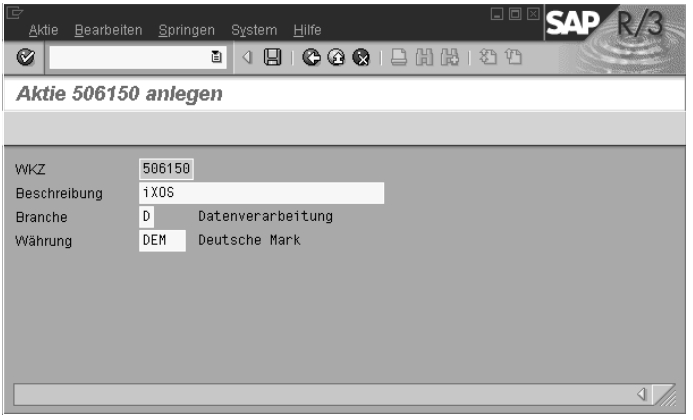

Abbildung 7.3 **© SAP AG**
Dynpro 9200 der Demoanwendung

Ausgangspunkt für die Analyse einer Anwendung sind die Dynpros bzw. deren Ablauflogik. Das nachfolgende Listing zeigt zunächst die Ablauflogik des Dynpro 9100.

```
PROCESS BEFORE OUTPUT.
*...Dynpro initialisieren
  MODULE status_9100.

PROCESS AFTER INPUT.
*...Exit-Codes auswerten
  MODULE exit_9100 AT EXIT-COMMAND.

*...Prüfen, ob Eingabewert zur aktuellen Transaktion passt
  FIELD yz3stock-wkz
    MODULE read_record_9100.

*...Verzweigen zum Folgedynpro
  MODULE user_command_9100.
```

Im Modul `status_9100` sollen alle Initialisierungen stattfinden. Ein derartiges Modul ist in nahezu allen Dynpros zu finden. Weitere Aktivitäten im PBO-Abschnitt sind nicht erforderlich. Auch der PAI-Teil enthält keine Besonderheiten. Die Auswertung der Exit-Funktionscodes erfordert ein Modul, die der übrigen Funktionscodes ein zweites. Da im Dynpro 9100 der zu bearbeitende Datensatz gelesen werden soll, ist noch ein dementsprechendes drittes Modul erforderlich.

Das Modul `status_9100` ist sehr einfach strukturiert. Zunächst wird, abhängig vom Transaktionscode, eine Überschrift gesetzt. Diese besteht aus einem feststehenden Text „Aktie & &", in den ein oder zwei variable Parameter eingefügt werden können. Im Dynpro 9100 wird nur ein Parameter benutzt, das Dynpro 9200 hingegen nutzt beide Parameter. Der Wortlaut des variablen Teils, der hier als nummeriertes Textelement programmiert wurde, ist als Kommentar hinter der Anweisung eingefügt.

```
MODULE status_9100 OUTPUT.
*...Titelzeile des Fensters setzen
  CASE sy-tcode.
    WHEN 'YZ3I'.
      SET TITLEBAR '001' WITH text-001.  " anlegen
    WHEN 'YZ3U'.
      SET TITLEBAR '001' WITH text-002.  " ändern
    WHEN 'YZ3S'.
      SET TITLEBAR '001' WITH text-003.  " anzeigen
  ENDCASE.

*...Status für Grundbild setzen
  SET PF-STATUS 'STAT_G'.

*...Kopfzeile der Tabelle löschen
  CLEAR yz3stock.
ENDMODULE.                 " STATUS_9100  OUTPUT
```

Nach dem Setzen der Überschrift wird der Status gesetzt, um die Verfügbarkeit der Funktionscodes zu gewährleisten. Zur Sicherheit wird auch die Kopfzeile YZ3STOCK initialisiert. Dadurch ist gewährleistet, dass beim Neuanlegen von Datensätzen im Dynpro 9200 immer ein leerer Datensatz zur Verfügung steht. Würde auf die Initialisierung verzichtet, könnte in einigen Fällen beim Neuanlegen eines Datensatzes im Dynpro 9200 der Inhalt des zuletzt bearbeiteten Dynpros erscheinen. Das Feld YZ3STOCK-WKZ wird durch die Wirkung des Get/Set-Parameters allerdings immer auf den aktuellsten Wert gesetzt.

Das PAI-Modul EXIT_9100 ist ebenfalls sehr einfach. Die einzige Besonderheit besteht in der Behandlung des Funktionscodes des Dynpros. Das für den Funktionscode vorgesehene Feld FCODE ist ein globales Datenfeld. Falls der Datenfreigabetaste kein Funktionscode zugeordnet wurde (dies ist erst ab Release 3.0 möglich, wobei aber auch dort nur selten von dieser Möglichkeit Gebrauch gemacht wird), führt das Auslösen dieser Taste nicht zu einem Überschreiben des Feldinhaltes von FCODE. Dieses Feld würde dann also noch den zuvor gültigen Funktionscode enthalten. In vielen Anwendungen, die nicht für Release 3.0 neu entwickelt oder komplett überarbeitet wurden, wird daher der Funktionscode in einem zweiten Feld gesichert und das eigentliche Feld für den Funktionscode gelöscht. Obwohl das Beispiel unter Release 4.6 entwickelt wurde, soll hier diese althergebrachte Programmiertechnik verwendet werden. Dies ermöglicht die problemlose Ausführung der Anwendung auch unter älteren Releaseständen.

```
MODULE exit_9100 INPUT.
*...Funktionscodefeld sichern und löschen, siehe Text
  g_fcode = fcode.
  CLEAR fcode.

*.....drei Funktionscodes verzweigen in andere Transaktionen
  CASE g_fcode.
    WHEN 'CREA'.
      LEAVE TO TRANSACTION 'YZ3I'.
    WHEN 'EDIT'.
      LEAVE TO TRANSACTION 'YZ3U'.
    WHEN 'SHOW'.
      LEAVE TO TRANSACTION 'YZ3S'.
  ENDCASE.

*...Falls nicht verzweigen, dann Programm beenden
  LEAVE TO SCREEN 0.
ENDMODULE.                      " EXIT_9100  INPUT
```

Im Exit-Modul müssen vor allem die drei Funktionscodes CREA, EDIT und SHOW ausgewertet werden, da sie eine differenzierte Behandlung erfordern. Alle anderen Exit-Funktionscodes sollen die Anwendung beenden. Daher wird nach der CASE-Anweisung lediglich noch ein Sprung zum Dynpro 0 erforderlich.

Noch einfacher als das Exit-Modul ist das Modul zur Auswertung der normalen Funktionscodes. Das Dynpro 9100 verfügt lediglich über die Datenfreigabetaste. Deren Betätigung soll die Bearbeitung des selektierten Datensatzes ermöglichen. Da dies im Dynpro 9200 erfolgt, ist im entsprechenden Modul nur der Sprung zum Dynpro 9200 zu programmieren.

```
MODULE user_command_9100 INPUT.
  g_fcode = fcode.
  CLEAR fcode.

*...wenn ENTER-Taste gedrückt, dann Datensatz bearbeiten
  IF g_fcode = ' '.
    LEAVE TO SCREEN 9200.
  ENDIF.
 ENDMODULE.                    " USER_COMMAND_9100  INPUT
```

Die eigentliche Funktionalität des Dynpros ist im Modul `READ_RECORD_9100` enthalten. Hier wird der zu bearbeitende Satz bereitgestellt. Dabei ist zu unterscheiden, ob ein neuer Satz angelegt oder ein bereits vorhandener bearbeitet werden soll. Diese Unterscheidung ist notwendig, da die Funktionalitäten der Anlegen- und der Ändern-Transaktion nicht miteinander vermischt werden dürfen. Im Modul wird daher zunächst versucht, den Datensatz anhand des im Dynpro eingegebenen Schlüsselbegriffs zu lesen. Dann wird der Rückgabecode der `SELECT`-Anweisung in Abhängigkeit vom Transaktionscode ausgewertet. Falls der Datensatz erfolgreich gelesen wurde, ist er durch Aufruf des Sperrbausteins gegen Fremdzugriff zu schützen.

Im Modul werden Nachrichten ausgelöst, falls das Ergebnis der `SELECT`-Anweisung nicht zum aktuellen Transaktionscode passt. Als Folge davon wird das Dynpro neu prozessiert, das Modul `USER_COMMAND_9100` also nicht ausgeführt.

```
MODULE read_record_9100 INPUT.
*....Versuchen, den gewünschten Satz zu lesen
  SELECT SINGLE * FROM yz3stock
    WHERE wkz = yz3stock-wkz.

*...SY-SUBRC gibt Auskunft, ob Satz gefunden
  IF sy-subrc = 0.

*.....Bei Anlegen-Transaktion darf noch kein Satz existieren

    IF sy-tcode = 'YZ3I'.
*.......Text: Aktie &/& existiert bereits!
      MESSAGE e001 WITH yz3stock-wkz yz3stock-name.
    ENDIF.

*.....Satz sperren
    CALL FUNCTION 'ENQUEUE_EYZ3STOCK'
```

```
      EXPORTING
        mode_yz3stock  = 'E'
        mandt          = sy-mandt
        wkz            = yz3stock-wkz
*       X_WKZ          = ' '
*       _SCOPE         = '2'
*       _WAIT          = ' '
*       _COLLECT       = ' '
      EXCEPTIONS
        foreign_lock   = 1
        system_failure = 2
        OTHERS         = 3.

    IF sy-subrc <> 0.
*...... Text: Satz kann nicht gesperrt werden!
      MESSAGE e005.
    ENDIF.

*...Satz existiert nicht
  ELSE.

*.....Bei Ändern und Anzeigen muss Satz vorhanden sein
    IF sy-tcode = 'YZ3U' OR sy-tcode = 'YZ3S'.

*...... Text: Aktie &/& existiert noch nicht!
      MESSAGE e002 WITH yz3stock-wkz yz3stock-name.
    ENDIF.
  ENDIF.
ENDMODULE.                     " READ_RECORD_9100  INPUT
```

Die Funktionalität dieses Moduls könnte auch im User-Command-Modul programmiert werden. Die Trennung macht aber deutlich, wie die Modularisierung der Anwendung erfolgen kann.

Das Dynpro 9200 übernimmt alle Pflegeaufgaben für den im Dynpro 9100 selektierten Datensatz. Dazu gehört die Prüfung der eingegebenen Werte und das Sichern der Daten. Bedingt durch die erweiterten Aufgaben und die differenziertere Behandlung der Exit-Funktionscodes ist der Quelltext für dieses Modul etwas umfangreicher. Dies zeigt sich schon in der Ablauflogik.

```
PROCESS BEFORE OUTPUT.
*...Dynpro initialisieren
  MODULE status_9200.

*...Zusatzinformationen lesen
  MODULE read_info_9200.

PROCESS AFTER INPUT.
```

```
  MODULE exit_9200 AT EXIT-COMMAND.
*...Datenveränderungen erkennen
  CHAIN.
    FIELD: yz3stock-branch,
           yz3stock-name,
           yz3stock-currency.
    MODULE data_changed_9200 ON CHAIN-REQUEST.
  ENDCHAIN.

*...Branche prüfen
  FIELD yz3stock-branch MODULE branche_check_9200.

*...Funktionscode auswerten
  MODULE user_command_9200.
```

Zunächst ist wiederum ein `STATUS`-Modul erforderlich, um den Status zu setzen und gegebenenfalls zu modifizieren. In der Tabelle `YZ3STOCK` enthalten die Felder `CURRENCY` und `BRANCH` lediglich Kürzel. Es ist für den Anwender oft nützlich, eventuelle Langtexte zu diesen Kennzeichen zu sehen. Die dazu notwendigen Felder wurden im Dynpro bereits vorgesehen. Sie müssen vor Darstellung des Dynpros natürlich mit Werten gefüllt werden. Diese Aufgabe wurde in ein zweites PBO-Modul `read_info_9200` verlagert.

Im PAI-Teil ist zunächst die Auswertung der Exit-Funktionscodes erforderlich. Innerhalb der nachfolgenden `CHAIN`-Kette werden alle Eingabefelder des Dynpros auf eventuelle Eingaben geprüft. Falls Datenveränderungen stattfanden, muss ein entsprechendes Flag gesetzt werden, das Einfluss auf Sicherheitsabfragen beim Beenden hat. Da mehr als ein Feld mit einem Modul verbunden werden soll, ist die `CHAIN`-Anweisung zum Verketten erforderlich. Es wäre allerdings auch denkbar, drei getrennte Modulaufrufe zu programmieren, wie nachfolgend dargestellt:

```
FIELD yz3stock-branch
  MODULE data_changed_9200 ON CHAIN-REQUEST.

FIELD yz3stock- name
  MODULE data_changed_9200 ON CHAIN-REQUEST.

FIELD yz3stock- currency
  MODULE data_changed_9200 ON CHAIN-REQUEST.
```

In der dem Datenfeld `YZ3STOCK-BRANCH` zu Grunde liegenden Domäne wurde ein zulässiger Datenbereich definiert. Die automatischen Feldprüfungen prüfen den Inhalt des Dynprofelds nur gegen diesen Wertebereich. Falls eine zusätzliche Einschränkung der zulässigen Werte auf die in der Tabelle `YZ3B` gepflegten Einträge stattfinden soll, ist ein weiteres Prüfmodul (`BRANCHE_CHECK_9200`) erforderlich. Letztlich müssen die normalen Funktionscodes, von denen es in diesem Dynpro zahlreiche gibt, verarbeitet werden.

Wegen der unterschiedlichen Reaktionen der Anwendung auf die Funktionstasten F3, F12 und F15 ist ein Zusammenspiel der beiden Module zur Funktionscodeverarbeitung erforderlich. Zunächst sollen aber die anderen vier Module beschrieben werden.

Im STATUS-Modul sind zusätzlich zu den beim Dynpro 9100 geschilderten Aufgaben zwei neue Aktivitäten erforderlich. Zunächst ist das Dynpro dynamisch zu modifizieren. In der Anzeige-Transaktion müssen beispielsweise alle Felder in reine Ausgabefelder umgewandelt werden, in den anderen Transaktionen ist nur das Feld YZ3STOCK-WKZ gegen Eingaben zu sperren. Es wird im Dynpro 9200 nur zur Information angezeigt, nachträgliche Änderungen des Schlüssels dürfen nach dem Selektieren und Sperren eines Datensatzes natürlich nicht möglich sein.

```
MODULE status_9200 OUTPUT.
*...Dynprofelder inaktiv setzen
*...beim Anzeigen alle, sonst nur Name
  LOOP AT SCREEN.
    IF screen-name = 'YZ3STOCK-WKZ'
      OR sy-tcode = 'YZ3S'.
      screen-input = c_off.
      MODIFY SCREEN.
    ENDIF.
  ENDLOOP.

*...Bei Anzeigetransaktion Sichern-Funktion inaktiv setzen
  CLEAR i_fcode.
  REFRESH i_fcode.
  IF sy-tcode = 'YZ3S'.
    i_fcode-fcode = 'FT11'.
    APPEND i_fcode.
  ENDIF.

*...bei allen Transaktionen außer Ändern
*...den Löschen-Code deaktivieren
  IF sy-tcode <> 'YZ3U'.
    i_fcode-fcode = 'FT14'.
    APPEND i_fcode.
  ENDIF.

*...Status für Ändern-Dynpro setzen
  SET PF-STATUS 'STAT_A' EXCLUDING i_fcode.

*...Globale Flags initialisieren
  g_exit   = c_false.
  g_delete = c_false.
  g_exit_module = c_false.
```

```
*...Titel mit WKZ ergänzen
  CASE sy-tcode.
    WHEN 'YZ3I'.
      SET TITLEBAR '001' WITH yz3stock-wkz text-001.  " anlegen
    WHEN 'YZ3U'.
      SET TITLEBAR '001' WITH yz3stock-wkz text-002.  " ändern
    WHEN 'YZ3S'.
      SET TITLEBAR '001' WITH yz3stock-wkz text-003. " anzeigen
  ENDCASE.
ENDMODULE.                           " STATUS_9200  OUTPUT
```

Die zweite neue Aufgabe besteht im Deaktivieren nicht benötigter Funktionscodes. Dazu wird eine interne Tabelle mit den Funktionscodes gefüllt. Diese Tabelle wird beim Setzen des Status durch den Zusatz `EXCLUDING` an das System übergeben. Im Dynpro 9200 regeln einige Flags das Verhalten der Anwendung. Manche dieser Flags werden bei jedem Durchlauf des PBO-Teils zurückgesetzt. Die Notwendigkeit für diesen Vorgang wird später beim Betrachten der beiden Funktionscode-Module ersichtlich.

Das Modul zum Lesen der beiden Bezeichnungen für Branche und Währung ist so einfach, dass es nicht näher erläutert werden muss.

```
MODULE read_info_9200 OUTPUT.
*...Branchenbezeichnung bereitstellen
  SELECT SINGLE * FROM yz3bt
  WHERE branch = yz3stock-branch AND
        langu  = sy-langu.

*...alten Inhalt löschen, wenn Suche erfolglos
  IF sy-subrc <> 0.
    CLEAR yz3bt.
  ENDIF.

*...Währungsbezeichnung bereitstellen
  SELECT SINGLE * FROM tcurt
  WHERE waers = yz3stock-currency AND
        spras = sy-langu.

*...alten Inhalt löschen, wenn Suche erfolglos
  IF sy-subrc <> 0.
    CLEAR tcurt.
  ENDIF.
ENDMODULE.                           " READ_INFO_9200  OUTPUT
```

Ähnliches gilt für die beiden PAI-Module.

```
MODULE data_changed_9200 INPUT.
*...SY-DATAR wird immer neu initialisiert,
*...muss daher gesichert werden
  IF sy-datar = c_true.
    g_changed = c_true.
  ENDIF.
ENDMODULE.                          " DATA_CHANGED_9200  INPUT
```

Das DATA_CHANGED-Modul setzt lediglich ein globales Flag. Dieses Flag muss natürlich auch dann erhalten bleiben, wenn nach erkannter Datenänderung das Dynpro nochmals prozessiert wird, beispielsweise weil ein Fehler auftrat oder weil die Datenfreigabetaste gedrückt wurde. Dieses Flag darf daher nicht im PBO-Teil initialisiert werden! Das Modul BRANCHE_CHECK_9200 ist wiederum so einfach, dass sich weitere Beschreibungen erübrigen.

```
MODULE branche_check_9200 INPUT.
  SELECT SINGLE * FROM yz3b
  WHERE branch = yz3stock-branch.

  IF sy-subrc <> 0.
*.....Text: Branche & nicht gepflegt!
    MESSAGE e006 WITH yz3stock-branch.
  ENDIF.
ENDMODULE.                          " BRANCHE_CHECK_9200
INPUT
```

Recht komplex hingegen sind die beiden Funktionscode-Module des Dynpros. Zunächst folgt der Quelltext des Exit-Moduls. Dieses Modul macht intensiv von verschiedenen Unterprogrammen Gebrauch, da einige Funktionen sowohl im Exit- als auch im User-Command-Modul benötigt werden. Es bietet sich daher an, diese Funktionen in Unterprogramme auszulagern.

```
MODULE exit_9200 INPUT.
  g_fcode = fcode.
  CLEAR fcode.
  g_exit_module = c_true.

*...Löschen extra behandeln, da unabhängig von Datenänderung
  IF g_fcode = 'FT14'.
    PERFORM delete_question_9200.
  ELSE.

*.....wenn Daten geändert, dann Sicherheitsabfrage
    IF g_changed = c_true OR
       sy-datar  = c_true.
```

```
*.......G_CHANGED setzen, damit Inhalt auch im ersten
*.......Durchlauf korrekte Auswertung ermöglicht
      g_changed = c_true.

*.......ggf. Sicherheitsabfragen durchführen
*.......und EXIT-Flag setzen
      PERFORM check_save_9200.

*.......wenn Beenden ohne Sichern gewünscht,
*.......dann Rücksprung direkt im EXIT-Modul
      IF g_exit = c_true AND
         g_save = c_false.
        g_changed = c_false.

*.........Satz entsperren, dann Rücksprung
        PERFORM dequeue_9200.
        PERFORM navigate_9200.
      ENDIF.

    ELSE.

*.......wenn keine Daten geändert
*.......G_EXIT setzen wegen Auswertung in NAVIGATE_9200
      g_exit = c_true.

*.......Satz entsperren, dann Rücksprung
      PERFORM dequeue_9200.
      PERFORM navigate_9200.
    ENDIF.
  ENDIF.
ENDMODULE.                          " EXIT_9200  INPUT
```

Zunächst wird wiederum der Funktionscode in ein zweites Feld gerettet und das mit dem Dynpro verbundene Feld initialisiert. In einem Flag wird vermerkt, dass das Exit-Modul aufgerufen wurde. Nicht alle Exit-Funktionscodes führen zwangsläufig zum Verlassen der Anwendung. Falls die Verarbeitung des Dynpros fortgesetzt wird, muss die Ausführung des Exit-Moduls im User-Command-Modul auf möglichst einfache Weise erkannt werden können.

Im Anschluss an das Setzen der Flags wird geprüft, ob der Datensatz gelöscht werden soll. Ist dies der Fall, so muss der Anwender seinen Wunsch in einem Popup bestätigen. Für alle anderen Exit-Codes erfolgt die Überprüfung von Datenänderungen. Falls bereits im ersten Durchlauf der Dynproverarbeitung ein Exit-Funktionscode ausgelöst wurde, kann das Flag `G_CHANGED` noch nicht gesetzt sein. Daher wird im Exit-Modul der Test sowohl von `SY-DATAR` als auch von `G_CHANGED` erforderlich. Damit alle weiteren Prüfungen einheitlich auf `G_CHANGED` zurückgreifen können, wird dieses Flag bei Bedarf gesetzt.

Je nach Exit-Code soll der Anwender die Möglichkeit erhalten, den Vorgang abzubrechen oder zumindest die eingegebenen Daten vor Beenden des Dynpros zu sichern, falls Daten geändert wurden. Dieses Verhalten wird durch die beiden Flags G_EXIT und G_SAVE gesteuert. Die Ausführung der Sicherheitsabfrage und die Auswertung des Ergebnisses werden durch das Unterprogramm CHECK_SAVE_9200 übernommen. Es wird ohne Parameter aufgerufen und setzt die beiden erwähnten Flags. Falls das Dynpro ohne Sichern verlassen werden soll, wird direkt im Exit-Modul der bearbeitete Datensatz entsperrt und das Dynpro verlassen. Auch diese beiden Funktionen werden durch Unterprogramme ausgeführt. Sie werden ohne weitere Abfrage ausgeführt, wenn ein Exit-Code (außer FT14) ausgelöst wurde, aber im Dynpro keine Änderung erfolgte.

Die beiden Funktionscodes F3 und F15 sollen das Speichern der geänderten Daten erlauben. Innerhalb des Exit-Moduls stehen die aktuellen Inhalte der Dynprofelder allerdings noch nicht zur Verfügung. Feldprüfungen haben ebenfalls noch nicht stattgefunden. Aus diesem Grund dürfen die mit diesen beiden Funktionstasten verbundenen Funktionscodes das Dynpro nicht direkt im Exit-Modul beenden, falls Daten gesichert werden müssen. Vielmehr werden nur die verschiedenen Flags gesetzt und das Programm fortgesetzt. Nach Ausführung des Exit-Moduls werden daher gegebenenfalls noch die beiden Prüfmodule und das User-Command-Modul ausgeführt. Dieses Modul ist nicht besonders umfangreich, da die eigentliche Funktionalität in die Unterprogramme verlagert wurde. Zunächst wird wieder der Funktionscode in einem zweiten Feld gesichert. Dies ist nur dann erforderlich, wenn das Exit-Modul nicht durchlaufen wurde. In diesem Fall ist es nicht nötig, das Unterprogramm CHECK_SAVE_9200 nochmals aufzurufen.

```
MODULE user_command_9200 INPUT.
*...Falls FCODE schon im EXIT-Modul zurückgesetzt
  IF g_exit_module = c_false.
    g_fcode = fcode.
    CLEAR fcode.

*.....feststellen, ob Daten gesichert werden müssen
*.....und Dynpro verlassen werden darf, Abfrage nur
*.....erforderlich, wenn EXIT-Modul nicht wirksam war
    PERFORM check_save_9200.
  ENDIF.

*...Daten sichern
  PERFORM save_9200.

*...Rücksprung, Ziel vom Funktionscode abhängig
  PERFORM navigate_9200.
ENDMODULE.                          " USER_COMMAND_9200  INPUT
```

Die beiden Unterprogramme SAVE_9200 und NAVIGATE_9200 werden auf jeden Fall durchlaufen, auch bei Betätigen der Datenfreigabetaste. Da in diesen Routinen aber die beiden Flags G_EXIT und G_SAVE ausgewertet werden, die in Abhängigkeit vom Funktionscode in CHECK_SAVE_9200 gesetzt werden, ist das Durchlaufen dieser Routinen nicht gleichbedeutend mit der Beendigung der Anwendung. Das folgende Listing zeigt die letztgenannte Routine:

```
FORM check_save_9200.
  CASE g_fcode.

*.....ENTER-Taste
*.....-> nur Datenprüfung
    WHEN ' '.
      g_save = c_false.
      g_exit = c_false.


*.....F3 = ZURÜCK
*.....-> Möglichkeit zum Sichern bei Datenänderung
    WHEN 'FT03'.
      PERFORM save_question_9200.

*.....F11 = SICHERN
*.....-> unbedingtes Sichern und Beenden des Dynpros
    WHEN 'FT11'.
      g_save = c_true.
      g_exit = c_true.

*.....F12 = ABBRECHEN
*.....-> nur Hinweis auf möglichen Datenverlust
    WHEN 'FT12'.
      PERFORM cancel_question_9200.

*.....F15 = BEENDEN
*.....-> Möglichkeit zum Sichern bei Datenänderung
    WHEN 'FT15'.
      PERFORM save_question_9200.

  ENDCASE.
ENDFORM.                    " CHECK_SAVE_9200
```

Je nach Funktionscode werden die globalen Flags gesetzt bzw. Sicherheitsabfragen ausgeführt. Diese wurden der Übersichtlichkeit wegen ebenfalls in eigene Unterprogramme verlagert. Die eigentliche Abfrage wird durch Aufruf eines von SAP gelieferten Funktionsbausteines ausgeführt. Je nach Abfrageergebnis werden die globalen Felder gesetzt.

```
FORM save_question_9200.
DATA: l_answer.

*...Abfrage nur erforderlich, wenn Daten geändert wurden
  IF g_changed = c_true.

    CALL FUNCTION 'POPUP_TO_CONFIRM_STEP'
      EXPORTING
*       DEFAULTOPTION = 'Y'
        textline1     = text-012  " Daten werden verlorengehen!
        textline2     = text-013  " Vorher sichern?
        titel         = text-010  " Achtung
*       START_COLUMN  = 25
*       START_ROW     = 6
      IMPORTING
        answer        = l_answer
      EXCEPTIONS
        OTHERS        = 1.
    CASE l_answer.

*.......Sichern und Beenden
      WHEN c_yes.
        g_save = c_true.
        g_exit = c_true.

*.......Beenden, ohne zu sichern
      WHEN c_no.
        g_save = c_false.
        g_exit = c_true.
*.......Abbrechen der Funktion -> Dynpro wird nicht beendet
      WHEN c_cancel.
        g_save = c_false.
        g_exit = c_false.
    ENDCASE.

*...wenn Daten nicht geändert, dann Beenden
*...immer möglich
  ELSE.
    g_save = c_false.
    g_exit = c_true.
  ENDIF.
ENDFORM.                    " SAVE_QUESTION_9200
```

Der Baustein POPUP_TO_CONFIRM_STEP stellt dem Anwender mittels dreier Drucktasten drei mögliche Entscheidungen frei: JA, NEIN und ABBRUCH. Übertragen auf den Grund der Abfrage bedeuten diese Antworten „Beenden mit Sichern", „Beenden ohne Sichern" oder „Dynpro nicht verlassen". Hingegen

verfügt der Baustein POPUP_TO_CONFIRM_LOSS_OF_DATA nur über die Auswahlmöglichkeiten JA und NEIN, was in der Anwendung „Beenden ohne Sichern" bzw. „Dynpro nicht verlassen" bedeutet.

```
FORM cancel_question_9200.
DATA: l_answer.

  CALL FUNCTION 'POPUP_TO_CONFIRM_LOSS_OF_DATA'
    EXPORTING
      textline1    = text-011   " Wirklich abbrechen?
*     TEXTLINE2    = ' '
      titel        = text-010   " Achtung
*     START_COLUMN = 25
*     START_ROW    = 6
    IMPORTING
      answer       = l_answer
    EXCEPTIONS
      OTHERS       = 1.

*...wenn wirklich abbrechen, dann EXIT-Flag setzen
  IF l_answer = c_yes.
    g_exit = c_true.
    g_save = c_false.

*...sonst EXIT-Flag zurücksetzen
  ELSE.
    g_exit = c_false.
  ENDIF.
ENDFORM.                    " CANCEL_QUESTION_9200
```

Die Sicherheitsabfrage, die beim Löschen eines Datensatzes erscheint, ähnelt bis auf die Texte der eben vorgestellten. Allerdings wird hier das Flag G_DELETE gesetzt, um der Routine SAVE_9200 zu signalisieren, dass der Datensatz aus YZ3STOCK zusammen mit allen Kursdaten aus YZ3PRICE nicht gesichert, sondern gelöscht werden soll.

```
FORM delete_question_9200.
DATA: l_answer.

  CALL FUNCTION 'POPUP_TO_CONFIRM_STEP'
    EXPORTING
      defaultoption = 'N'
      textline1     = text-021  " Datensatz mit allen
Folgesätzen
      textline2     = text-022  " wirklich löschen?
      titel         = text-023  " LÖSCHEN
*     START_COLUMN  = 25
*     START_ROW     = 6
    IMPORTING
```

```
      answer        = l_answer
    EXCEPTIONS
      OTHERS        = 1.

  IF l_answer = c_yes.
    g_delete = c_true.
    g_exit   = c_true.

  ELSE.
*.....nicht unbedingt notwendig, nur zur Sicherheit
    g_delete = c_false.
    g_exit   = c_false.
  ENDIF.
ENDFORM.                    " DELETE_QUESTION_9200
```

Wegen dieser beiden grundsätzlich verschiedenen Aufgaben (Sichern und Löschen) ist SAVE_9200 wieder etwas umfangreicher.

```
FORM save_9200.
  IF g_save = c_true.

    MODIFY yz3stock.

*.....Status der Datenbankoperation testen
    IF sy-subrc <> 0.

*.......Text: Fehler beim Speichern des Datensatzes!
      MESSAGE e003.
    ELSE.

*.......Text: Daten für & gesichert.
      MESSAGE s004 WITH yz3stock-wkz.
    ENDIF.

*.....Datensatzsperre aufheben
    PERFORM dequeue_9200.

*.....Änderungen festschreiben
    COMMIT WORK.

*.....Datenänderungsflag zurücksetzen
    g_changed = c_false.

  ENDIF.

*...Falls gelöscht werden soll
  IF g_delete = c_true.
    DELETE FROM yz3price
      WHERE wkz = yz3stock-wkz.
```

```
    DELETE FROM yz3stock
      WHERE wkz = yz3stock-wkz.

    COMMIT WORK.
    g_exit = c_true.
  ENDIF.
ENDFORM.                    " SAVE_9200
```

Sehr leicht überschaubar hingegen ist die Funktion von `NAVIGATE_9200`, die jede Beschreibung überflüssig macht.

```
FORM navigate_9200.
  IF g_exit = c_true.
    CASE g_fcode.

*.......F3 = ZURÜCK
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT03'.
        LEAVE TO SCREEN 9100.

*.......F11 = SICHERN
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT11'.
        LEAVE TO SCREEN 9100.

*.......F12 = ABBRECHEN
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT12'.
        LEAVE TO SCREEN 9100.

*.......F14 = LÖSCHEN
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT14'.
        LEAVE TO SCREEN 9100.

*.......F15 = BEENDEN
*.......-> Beenden der gesamten Anwendung
      WHEN 'FT15'.
        LEAVE PROGRAM.
    ENDCASE.
  ENDIF.
ENDFORM.                    " NAVIGATE_9200
```

Der Vollständigkeit halber sollen hier auch noch die globalen Datendeklarationen und das Unterprogramm zur Freigabe der Datenbanksperre aufgeführt werden:

```
PROGRAM  sapmyz3s MESSAGE-ID y3.
TABLES:
  yz3stock,
  yz3price,
  yz3b,
  yz3bt,
  tcurt
.  " TABLES

CONSTANTS:
  c_true          VALUE 'X',
  c_false         VALUE ' ',
  c_yes           VALUE 'J',
  c_no            VALUE 'N',
  c_cancel        VALUE 'A',
  c_on     TYPE i VALUE 1,
  c_off    TYPE i VALUE 0
.  " CONSTANTS
DATA:
  fcode         LIKE sy-ucomm,
  g_fcode       LIKE fcode,
  g_changed     LIKE c_false VALUE c_false,
  g_save        LIKE c_false VALUE c_false,
  g_exit        LIKE c_false VALUE c_false,
  g_delete      LIKE c_false VALUE c_false,
  g_exit_module LIKE c_false VALUE c_false,

  BEGIN OF i_fcode OCCURS 5,
    fcode LIKE sy-ucomm,
  END OF i_fcode
.  " DATA

FORM dequeue_9200.
  CALL FUNCTION 'DEQUEUE_EYZ3STOCK'
    EXPORTING
      mode_yz3stock = 'E'
      mandt         = sy-mandt
      wkz           = yz3stock-wkz
*     X_WKZ         = ' '
*     _SCOPE        = '3'
*     _SYNCHRON     = ' '
*     _COLLECT      = ' '
  .
ENDFORM.                    " DEQUEUE_9200
```

7.3 Programm 2: SAPMYZ3P

Nach der ausführlichen Beschreibung der ersten Anwendung sollte es für Sie kein Problem mehr sein, die zweite Anwendung selbst zu erstellen. Ihre Struktur weist große Ähnlichkeit mit der eben beschriebenen Anwendung auf. Dadurch ergibt sich auch eine weitgehende Übereinstimmung des Quelltextes.

In einem ersten Dynpro wird ein Datum eingetragen. Im zweiten Dynpro wird dann für jede in der Tabelle `YZ3STOCK` gepflegte Aktie der Kurs für diesen Tag erfasst oder, falls schon vorhanden, bearbeitet. Das Löschen von Datensätzen oder das reine Anzeigen ist nicht vorgesehen. Es existiert daher auch nur eine einzige Dialogtransaktion `YZ3P`.

Zum Bearbeiten der Kursdaten wird eine interne Tabelle erzeugt, deren Struktur der Tabelle `YZ3PRICE` entspricht. An diese wird für jede Aktie aus `YZ3STOCK` ein Datensatz angefügt und mit einem eventuell vorhandenen Kurs ergänzt. Im Bearbeitungsdynpro wird diese Tabelle mit einem Table View bearbeitet. Abbildung 7.4 zeigt die Gestaltung dieses Dynpros.

Sie können den existierenden Modul-Pool `SAPMYZ3S` mit allen seinen Bestandteilen zunächst kopieren. Das Ziel kann beispielsweise `SAPMYZ3P` heißen. Beim Kopieren muss darauf geachtet werden, dass alle Bestandteile (Status, Includes) wirklich kopiert werden. Während des Kopiervorgangs erscheint ein Popup, in dem alle zu kopierenden Elemente ausgewählt werden müssen. In der Kopie können Sie dann alle anwendungsspezifischen Quelltextelemente entfernen und nach der Bearbeitung der Dynpros neu programmieren. Sie sollten versuchen, diese zweite Anwendung zunächst selbst zu programmieren. Dabei können Sie auf den Quelltext der ersten Anwendung und auf einige Beispiele aus den vorangegangenen Kapiteln zurückgreifen. Zwecks Orientierung bei eventuellen Problemen finden Sie den kompletten Quelltext am Ende des Kapitels 9.

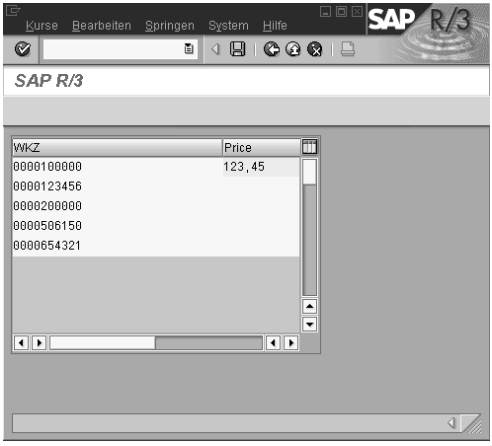

Abbildung 7.4 **© SAP AG**
Dynpro zur Pflege der Kursdaten

8 Tipps und Fallen

8.1 TIPPS

8.1.1 Erzeugen von Dynpro-Feldern

Dynpro-Felder sollten stets mit Dictionary-Bezug erzeugt werden. Damit ist eine einfache Übersetzung gewährleistet. Im Popup *Feldattribute* sind einige Angaben zum aktuellen Dictionary-Bezug enthalten. Schlüsselfeld und zugehörige Eingabemaske können durchaus zu verschiedenen Tabellen gehören bzw. es kann sein, dass nur das Schlüsselfeld Dictionary-Bezug hat, um einen übersetzungsfähigen Text zur Verfügung zu stellen, und die Eingabemaske sich auf irgendein programminternes Feld bezieht. Es ist auch möglich, für Dynpros Dictionary-Strukturen zu erzeugen, in denen alle im Dynpro benötigten Felder enthalten sind.

8.1.2 Nebenwirkungen bei Funktionscodes

Der Funktionscode in einem Dynpro wird oft in einem globalen Feld des Modul-Pools abgelegt, welches von mehreren Dynpros für diesen Zweck verwendet wird. Es kann zu Problemen führen, wenn im PAI-Teil eines Dynpros mit CALL ein weiteres Dynpro, z.B. als Popup, aufgerufen wird. Nach Rückkehr aus dem mit CALL gerufenen Objekt hat das globale Feld für den Funktionscode mit hoher Wahrscheinlichkeit einen anderen Inhalt als vorher. Das kann zu Problemen führen, wenn dieses Feld im PAI-Teil nach dem CALL-Aufruf nochmals ausgewertet werden muss. Derartige Probleme vermeiden Sie, wenn alle Auswertungen des Funktionscodes für ein Dynpro in ein Unterprogramm verlagert und in diesem Dynpro der Funktionscode am Anfang des Unterprogramms in ein lokales Feld kopiert wird. Zur Auswertung wird dann nur noch das lokale Feld benutzt.

8.1.3 Feldprüfungen

Feldprüfungen sind nur notwendig, wenn Daten gesichert oder in Folgedynpros weiter ausgewertet werden sollen. Es empfiehlt sich daher, in den Modulen, die für Feldprüfungen zuständig sind, den OK-Code zu testen und die Prüfungen nur dann auszuführen, wenn diese OK-Codes wirklich sichern. Das vermeidet z.B. auch unnötige Fehlerprüfungen beim Aufruf eines ergänzenden Popups, das ohnehin zum Hauptbild zurückkehrt.

Falls ein Modul-Pool für mehrere Transaktionen benutzt wird, sollte außer dem Funktionscode auch der Transaktionscode getestet werden, damit Feldprüfungen bei reinen Anzeigetransaktionen ebenfalls unterdrückt werden können.

8.1.4 Feldangabe bei Select-Options

Die korrekte Syntax für das Kommando `SELECT-OPTIONS` lautet

```
SELECT-OPTIONS selection FOR field.
```

Das bedeutet, dass eine Selektion für das angegebene Feld angelegt wird. Der Name des Feldes ist aus folgenden Gründen wichtig:

❶ Das angegebene Feld bestimmt das Format des Feldes auf dem Selektionsbildschirm, insbesondere die Feldlänge.

❷ Wird die Selektion in einem CHECK-Kommando in der Form

```
CHECK selection.
```

benutzt, so prüft das Kommando automatisch die zu Reportbeginn gefüllte Selektion gegen das Feld aus dem `SELECT-OPTIONS`-Kommando. Dies kann z.B. ein Tabellenfeld einer Tabelle sein, die in einer Schleife durchlaufen wird.

Wird hingegen die Selektion mittels des `IN-Operators in einer SELECT`-Anweisung oder einer `IF`-Anweisung benutzt, so interessiert der Feldname nicht, wenn das Vergleichsfeld explizit angegeben wird. Wird hingegen die abkürzende Schreibweise, z.B.

```
IF selection.
```

benutzt, so wird wieder das in der `SELECT-OPTIONS`-Anweisung angegebene Feld gewählt.

8.1.5 Strukturen

Strukturen speichern zwar keine Daten in einer Datenbanktabelle, besitzen aber einen bzw. entsprechen automatisch einem Kopfsatz, den Sie zur Datenspeiche-

rung benutzen können. Sie können eine Struktur auch mit Dynprofeldern verbinden, ohne dass Sie von der Struktur eine Feldleiste ableiten müssen.

8.1.6 Entsperren

Mitunter kommt es z. B. durch Netzprobleme oder Client-Absturz vor, dass sich einige Elemente nicht mehr bearbeiten lassen. Sie sind für das SAP-System scheinbar noch in Bearbeitung, da ihre Bearbeitung nicht ordnungsgemäß beendet wurde. Da sie zur Bearbeitung gesperrt werden, ist der nochmalige Zugriff zum Ändern nicht möglich, es erscheint dann eine Meldung der Form „Der Benutzer xyz bearbeitet abc bereits".

Um weiterarbeiten zu können, muss die Sperre aufgehoben werden. Dies erfolgt mit der Menüfunktion WERKZEUGE | ADMINISTRATION | MONITOR | SPERREINTRÄGE. Im darauf folgenden Dynpro können Sie einschränkende Suchbegriffe vorgeben. Anschließend erscheint eine Liste mit den momentanen Sperreinträgen, die markiert und mittels der Drucktaste LÖSCHEN gelöscht werden können.

8.1.7 Formale Parameter und Offset-Angaben

Auf einzelne Zeichen oder Zeichengruppen eines Feldes kann per Offset- und Längenangabe zugegriffen werden. Für formale Parameter eines Unterprogramms ist dies allerdings nicht möglich. Möchten Sie derartige Parameter per Offset bearbeiten, ist der Inhalt entweder in eine lokale Variable zu kopieren, dort zu bearbeiten und anschließend wieder in den formalen Parameter zu übertragen oder es wird ein Feldsymbol benutzt, das auf den Parameter zeigt. Auf das Feldsymbol kann per Offset- und Längenangabe zugegriffen werden.

8.1.8 Geltungsbereich von Feldern

Module stellen keinen eigenständigen Geltungsbereich für Felder dar. Das bedeutet, dass in Modulen keine Felder definiert werden dürfen, die bereits im Rahmenprogramm existieren. Die in Modulen angelegten Felder gelten auch außerhalb der Module weiter.

Wird innerhalb einer `FORM`-Routine ein Feld mit `DATA` definiert, ist es nur so lange verfügbar, bis die Anweisung `ENDFORM` der jeweiligen Routine erreicht wurde. Wird aus der Routine eine andere aufgerufen, so ist das Feld dort nicht verwendbar.

Wird in einer Form-Routine ein Feld mit `LOCAL` definiert, so wird der aktuelle Inhalt des Feldes, das bereits existieren muss, intern gesichert und erst beim Verlassen der Routine wieder zurückgeholt. Sie arbeiten also immer direkt mit dem global definierten Feld. Dieses hat also unabhängig vom Programmblock,

in dem LOCAL steht, zeitweise einen anderen Inhalt, der auch gültig ist, wenn eine weitere Routine aufgerufen wird!

8.1.9 Funktionstasten

In Subscreens, z. B. Popups zur Listenauswahl, werden am unteren Rand Drucktasten eingeblendet. Diese Drucktasten werden innerhalb der Oberfläche für dieses Subscreen wie die üblichen Drucktasten normaler Dynpros definiert.

Die Größe der Popups wird sehr oft dynamisch bestimmt, also abhängig von der Breite der anzuzeigenden Datenzeilen. Dabei kann es vorkommen, dass die Breite des Popups nicht für alle Drucktasten ausreicht. Diese sind zwar aktiviert, aber nicht sichtbar. Es ist daher hilfreich, die Funktionstastenbelegung mit der rechten Maustaste einzublenden, falls wichtige Funktionen, die im Popup zur Verfügung stehen sollten, nicht verfügbar sind. Dazu muss sich der Mauszeiger innerhalb des Popups befinden. Bei der Gestaltung der Oberfläche sollten die wichtigsten Drucktasten daher zuerst definiert werden, damit sie so weit wie möglich links liegen.

8.1.10 Drucktasten

Die Funktionscodes der Drucktasten im Arbeitsbereich von Dynpros und die des Menu-Painters werden getrennt verwaltet, auch wenn es sich um denselben Funktionscode handelt. Das bedeutet, dass sowohl eine Drucktaste als auch eine Funktionstaste denselben Funktionscode auslösen können, jedoch mit unterschiedlichem Funktionstyp. Diese Funktionscodes werden in der Ablauflogik u. U. unterschiedlich behandelt, obwohl es sich scheinbar um denselben Funktionscode handelt! Dieser Fehler kann auch bei zwei Drucktasten innerhalb eines Dynpros erzeugt werden, also mehrere Tasten mit demselben Funktionscode und unterschiedlichen Funktionstypen! Die Funktionstypen sind nur in der Oberflächen- oder Dynpropflege zu sehen, beim Debuggen eines Programms bleiben sie unbemerkt!

8.1.11 Syntaxprüfung

Die Syntax des Function-Pools einer Funktionsgruppe wird komplett geprüft. Auch beim Aufruf einer Funktion, an deren Code nicht gearbeitet wird, können durch Syntaxfehler in anderen Programmteilen oder Includes alle Funktionen der Funktionsgruppe gesperrt werden. Abhilfe ist nur möglich, wenn der jeweils aktive Programmcode beim Abspeichern frei von Syntaxfehlern ist. Das bedeutet nicht, dass er auch fehlerfrei arbeiten muss. Es sollten daher keine Programmteile aktiviert werden, die Syntaxfehler enthalten.

8.1.12 Rückkehr zum Startbild einer Transaktion

Viele Programme besitzen ein Grundbild, in dem zunächst Schlüsselwerte eingetragen werden, worauf zur eigentlichen Bearbeitung verzweigt wird. Dieses Grundbild wird immer wieder angezeigt, nachdem das eigentliche Bearbeitungsdynpro verlassen wird. Dies kann im PAI-Teil des Bearbeitungsdynpros mit zwei unterschiedlichen Methoden erfolgen. Zunächst ist es möglich, das Folgedynpro explizit anzugeben, z.B. mit:

```
LEAVE TO SCREEN 100.
```

In diesem Fall wird die Transaktion nicht verlassen, globale Daten usw. bleiben erhalten. Die zweite Möglichkeit besteht darin, die Transaktion neu zu starten:

```
LEAVE TO TRANSACTION SY-TCODE.
```

Dabei wird alles neu initialisiert, Datenbanksperren werden freigegeben usw. Dies vermeidet oft Probleme mit Verbuchungsbausteinen o.Ä.

8.1.13 Dynpro-Modifikation

Mit einer Loop über die automatisch generierte Tabelle `SCREEN` kann das Aussehen eines Dynpros modifiziert werden (Input, Ausblenden...). Mit dem Attribut `ACTIVE` eines Feldes kann dieses ganz aus dem Dynpro herausgenommen werden. Alle unter diesem Feld sitzenden Felder rutschen nach oben, Rahmen werden automatisch aktualisiert.

8.1.14 Drucktasten in Selektionsbildschirmen

In Selektionsbildschirmen können beliebig viele Drucktasten im Bildschirmbereich und eine oder zwei Drucktasten in der Symbolleiste angelegt werden. Drucktasten erscheinen trotz Deklaration nur dann auf dem Bildschirm, wenn mindestens eine Selektion oder ein Parameter existiert.

Die von den Drucktasten gelieferten Funktionscodes müssen ausgewertet werden, z.B. zum Zeitpunkt `AT-SELECTION-SCREEN`. Das Problem besteht darin, dass natürlich nur ausgewählte Funktionscodes zur List-Erstellung verzweigen (`START-OF-SELECTION`). Alle anderen Funktionscodes bewirken lediglich die Rückkehr zum Selektionsbild. Soll eine Drucktaste also den Report starten, meist nach Setzen eines speziellen Flags, so wird in `AT-SELECTION-SCREEN` zunächst über `SSCRFIELDS-UCOMM` der Funktionscode ausgewertet und dann in `SSCRFIELDS-UCOMM` gestellt, der den Sprung zur List-Verarbeitung auslöst, üblicherweise `ONLI`.

```
CASE SSCRFIELsDS-UCOMM.
  WHEN 'TEST'.
```

```
      G_CHANGE = ' '.
      SSCRFIELDS-UCOMM = 'ONLI'.
    WHEN 'COPY'.
      G_CHANGE = 'X'.
      SSCRFIELDS-UCOMM = 'ONLI'.
  ENDCASE.
```

8.1.15 Eingabebereitschaft in Grundbildern

Viele Dynpros werden so aufgebaut, dass in einem Anforderungsbild die Schlüsselwerte eingetragen werden, zu denen im Folgebild Daten editiert werden. Falls die eingetragenen Schlüsselwerte auf keinen Satz passen, müssen alle Eingabefelder für den Schlüssel wieder aktiv werden. Dazu müssen die Lese- bzw. Prüfroutine und eine `FIELD`-Anweisung für alle Schlüsselfelder in einer `CHAIN`-Kette stehen. Falls das Lesen direkt in dem Modul erfolgt, in dem auch die Funktionscodes ausgewertet werden, ist dieses Modul in der `CHAIN`-Kette auszuführen.

```
PROCESS AFTER INPUT.
CHAIN.
  FIELD: TAB-A, TAB-B, TAB-C.
  MODULE OKCODE.
ENDCHAIN.
```

8.1.16 Fehlende Berechtigungen

Falls Sie beim Ausführen einer Anwendung wegen unzureichender Berechtigungen eine Fehlermeldung erhalten, können Sie die Transaktion `SU53` aufrufen. Diese Transaktion zeigt Ihnen die zuletzt geprüfte Berechtigung mit den geprüften Werten sowie die in Ihren Berechtigungen enthaltenen Werte an.

8.2 TRICKS

8.2.1 Funktionsparameter (Release < 3.0)

Bis zum Release 2.2 besitzen Funktionsbausteine nur Import- und Exportparameter. Import-Parameter werden innerhalb des Bausteins gelesen, bei Beendigung aber erfolgt keine Datenrückgabe. Exportparameter hingegen werden nur zur Datenrückgabe benutzt, nicht aber, um dem Funktionsbaustein Daten zu übergeben. Soll also ein Funktionsbaustein den Inhalt eines Parameters einlesen, diesen verändern und dann wieder zurückgeben, müssen dafür ein Import- und ein Exportparameter geschaffen werden, denen beim Aufruf des Bausteins

jeweils dasselbe Feld zugewiesen wird, also etwa wie im folgenden Beispiel dargestellt:

```
CALL FUNCTION 'UPPER_CASE'
  IMPORTING
    IP_WORD = L_WORD
  EXPORTING
    OP_WORD = L_WORD.
```

Ab Release 3.0 können für derartige Zwecke `CHANGING`-Parameter benutzt werden.

8.2.2 Dynpro

Um zu einem Dynpro weitere Werte zu erfassen, die keinen Platz auf einem Bildschirm haben, werden Folgedynpros benutzt. Damit die Übersichtlichkeit einer solchen Dynprofolge nicht verloren geht, werden die Folgedynpros oft als (mehrfach verschachtelte) Popups dargestellt. Damit der Anwender beim Beenden des letzten Dynpros nicht schrittweise sämtliche Dynpros durchlaufen muss, kann der beendende Funktionscode durch alle Popups von hinten nach vorn durchgereicht werden.

```
Bsp:
DYNPRO 400:
...
CALL SCREEN 500 STARTING AT 10 10.
* Der Subscreen 500 wird aufgerufen und abgearbeitet.
* Die Beendigung erfolgt z.B. mit den Funktionscodes DATU
* (Daten übernehmen) und EABB (Abbruch)
* Der Funktionscode wird im Globalen Feld FCODE aufbewahrt,
* das im Dynpro 500 nicht gelöscht wird

IF FCODE = 'DATU'.
  CLEAR FCODE.  "nur im ersten Popup
  SET SCREEN 0.
  LEAVE SCREEN.
ENDIF.
...

DYNPRO 500:
...
IF FCODE = 'DATU'.
  SET SCREEN 0.
  LEAVE SCREEN.
ENDIF.
...
```

Die Folge dieser Anweisungen:

Das aufrufende Dynpro 400, aus dessen Ablauflogik die obigen Zeilen stammen, sei das erste Popup, das von einem Grundbild aus aufgerufen wird. Das Dynpro 500 wird aufgerufen und abgearbeitet, ebenfalls als Popup. Innerhalb dieses Dynpros sollen die Funktionstasten DATEN ÜBERNEHMEN (Funktionscode DATU) und ABBRECHEN (EABB) zur Verfügung stehen. Die Taste DATEN ÜBERNEHMEN soll die Rückkehr zum Grundbild veranlassen, also in einem Zug alle Popups beenden. Aus diesem Grund wird FCODE im Dynpro 500 nicht gelöscht, sondern an Dynpro 400 weitergereicht. Dort wird unmittelbar nach der CALL SCREEN-Anweisung FCODE ausgewertet und das Dynpro gegebenenfalls beendet. Damit im rufenden Grundbild keine unerwünschten Nebeneffekte auftreten, wird FCODE vorher initialisiert.

Falls ein Abbruch über mehrere Stufen hinweg erfolgen soll, werden die Anweisungen aus Dynpro 400 sinngemäß in alle anderen Dynpros übernommen. Das Initialisieren von FCODE darf dann natürlich nur in dem Dynpro erfolgen, das vom Grundbild aus aufgerufen wurde.

8.2.3 Einfache Ausgabe von Char-Feldern

In vielen Testreports werden Tabelleninhalte, meist Schlüsselfelder der Typen C und N, aufgelistet. Dies erfordert in der Ausgabeanweisung die Notation aller auszugebenden Felder. Dieser Vorgang kann erleichtert werden, falls wirklich nur C- und N-Felder ausgegeben werden. Zunächst wird eine Feldleiste angelegt, die alle auszugebenden Felder der Tabelle enthält. Zwischen diesen Feldern wird jeweils ein Leerfeld eingefügt, z.B.:

```
DATA: BEGIN OF F_TEST,
  NAME LIKE TAB-NAME,
  TRENN1,
  VORNAME LIKE TAB-VORNAME,
  TRENN2,
  GEBDAT LIKE TAB-GEBDAT,
END OF F_TEST.
```

Im Programm wird diese Feldleiste dann in der SELECT-Schleife mit MOVE-CORRESPONDING der Inhalt zugewiesen und die Feldleiste mit

```
WRITE / F_TEST.
```

ausgegeben. Da die Feldleiste als Ganzes vom System als C-Feld behandelt wird, erfolgt eine korrekte Anzeige. Speziell auf Datentypen beruhende Transformationen, z.B. über das Datenelement gesteuerte Konvertierungsfunktionen, werden so allerdings nicht wirksam.

8.3 FALLEN

8.3.1 Stringvergleich mit CA (Contains any) und NA (Not any)

Strings besitzen eine vorgegebene Länge. Wird das Vergleichsmuster eines Zeichenkettenvergleichs nicht als Stringkonstante, sondern als Feld angegeben, muss das Muster exakt dieselbe Länge besitzen wie das Feld, dieses also komplett ausfüllen. Ansonsten tritt folgendes Problem auf: Die nicht belegten Stellen werden mit Leerzeichen gefüllt, eventuell nicht belegte Stellen des zu verprobenden Strings ebenfalls. Der Vergleich wird dann das Ergebnis `TRUE` melden, da in beiden Strings Leerzeichen enthalten sind. Natürlich ist dieses Verhalten unerwünscht, zudem ist der Fehler nur schwer zu entdecken.

8.3.2 Suche nach Strings mit SEARCH

Mit dem Kommando `SEARCH` kann innerhalb eines Strings nach dem Auftreten eines bestimmten Musters gesucht werden. Dabei werden abschließende Leerzeichen bei der Musterangabe automatisch ignoriert. Falls nach einem Leerzeichen gesucht werden soll, führt die Anweisung

```
SEARCH string FOR ' '.
```

nie zum Erfolg. Falls die automatische Interpretation von Sonderzeichen oder dem Leerzeichen unterdrückt werden soll, sind die Zeichen mit zwei Punkten einzuschließen. Das korrekte Kommando lautet also

```
SEARCH string FOR '. .'.
```

8.3.3 Zahlenwerte in Zeichenfeldern

Die automatische Typumwandlung ermöglicht es, auch in Feldern vom Typ C Zahlenwerte anzulegen und mit diesen Werten in Ausdrücken problemlos zu arbeiten. Problematisch wird es, wenn die Länge des Zeichenfelds nicht ausreicht, um den Zahlenwert aufzunehmen.

8.3.4 Datenübergabe aus Dynpros

Nach dem Verlassen eines Dynpros sind die gerade eingegebenen Daten erst nach einer `AT EXIT-COMMAND`-Aweisung gültig. Bis zu diesem Zeitpunkt stehen in den Dynprofeldern noch die alten Werte. Dies bedeutet, dass in den `AT EXIT-COMMAND`-Modulen keine Datensicherung möglich ist!

Die AT EXIT-COMMAND-Module eignen sich also nur zum Beenden der Abarbeitung ohne Datensicherung. Die Information, dass Daten geändert wurden, kann aber auch innerhalb der AT EXIT-COMMAND-Module mittels der Systemvariablen SY-DATAR ermittelt werden, die in diesem Fall ein „X" enthält. Eine Sicherheitsabfrage kann zumindest auf den drohenden Datenverlust hinweisen. Dieses Feld wird allerdings bei jedem PBO-Durchlauf neu initialisiert und nur dann gesetzt, wenn im jeweiligen Durchlauf Daten geändert wurden.

Auch in den Help-Request-Modulen sind die Dynprofelder nicht ohne weiteres verfügbar. Das Auslesen kann in speziellen Fällen mit Basis-Funktionsbausteinen direkt aus dem Dynpro erfolgen (siehe Kapitel 3, Funktionsbausteine).

8.3.5 Initialisierung von Dynprofeldern

In den PBO-Modulen können Dynprofelder initialisiert werden. Bei Transaktionen, die neue Datensätze anlegen, ist es wünschenswert, die Felder für die neu zu erfassenden Werte bei jedem Aufruf des Dynpros zu leeren. Es könnte sonst passieren, dass die Dynprofelder noch Werte aus vorangegangenen Transaktionen besitzen, falls das zweite (das eigentliche Bearbeitungs-) Dynpro nach Dateneingabe abgebrochen wurde. Die Initialisierung darf aber in diesem speziellen Fall nicht in den PBO-Modulen des zweiten Dynpros, sondern muss beim Verzweigen aus dem ersten Dynpro heraus stattfinden, weil ansonsten bei jeder Fehlermeldung im zweiten Dynpro die Felder neu initialisiert würden, da ja E-Messages das Dynpro komplett neu abarbeiten.

Eine zweite Möglichkeit besteht im Verwenden eines Flags, das beim ersten Aufruf nach Initialisieren der Dynprofelder gesetzt und bei jedem LEAVE zurückgesetzt wird. Die Initialisierung kann dann in den PBO-Modulen erfolgen. Dies ist aber nur eine Problemverlagerung, da dieses Flag korrekt an allen Ein- und Aussprungstellen beachtet und bearbeitet werden muss.

8.3.6 SELECT INTO

In der Regel werden die Sätze, die mit einem SELECT-Kommando gefunden werden, im Arbeitsbereich (Kopfzeile) der durchsuchten Tabelle zur Verfügung gestellt und können dort ausgewertet werden. Nach einer SELECT-ENDSELECT-Schleife steht in der Kopfzeile der Tabelle der letzte gefundene Satz, nicht der letzte Satz der Tabelle. Dies ist nützlich, um zu prüfen, ob Sätze mit einem nicht vollständig bekannten Schlüssel existieren. Innerhalb der SELECT-ENDSELECT-Schleife finden keinerlei Aktivitäten statt. Erst nach dem Verlassen der Schleife wird der Kopfsatz bearbeitet.

Eine Abweichung vom geschilderten Verhalten ergibt sich, wenn der Zusatz

```
INTO TABLE Itab
```

bzw.

```
INTO workarea
```

benutzt wird. In diesem Fall ist die Kopfzeile der durchsuchten Tabelle auf jeden Fall leer! Die Ergebnisse stehen entweder in der Feldleiste oder in der Tabelle, die nach INTO angegeben wurde. Dabei ist die Kopfzeile der internen Tabelle zunächst auch leer. Sofort nutzbar ist nur der Arbeitsbereich, er enthält nach ENDSELECT den letzten gelesenen Satz.

Falls mit einer internen Tabelle gearbeitet wird, muss entweder der Kopfsatz der internen Tabelle mit

```
READ itab INDEX 1.
```

vorsorglich gefüllt werden, oder aber die Tabelle wird in einer Loop bearbeitet.

8.3.7 Mehrere Zuweisungen in UPDATE-Anweisung

Wenn in einer UPDATE-Anweisung mehrere Zuweisungen mit einer WHERE-Klausel verknüpft werden sollen, müssen die Zuweisungen ohne Komma aufeinander folgen, z. B. in der Form

```
UPDATE YZ3STOCK
  SET BRANCH = 'S' CURRENCY = 'DEM'
  WHERE WKZ = '123456'.
```

In diesem Fall werden beide Zuweisungen nur dann ausgeführt, wenn die Bedingung erfüllt ist. In der UPDATE-Anweisung ist auch die Doppelpunkt-Variante für die SET-Klausel syntaktisch korrekt. Dies verändert das Verhalten der UPDATE-Anweisung aber auf entscheidende Weise. In der Anweisung

```
UPDATE YZ3STOCK
  SET: BRANCH = 'S', CURRENCY = 'DEM'
  WHERE WKZ = '123456'.
```

gilt die WHERE-Klausel nur noch für die letzte Zuweisung. Die erste Zuweisung wird für alle Datensätze der Tabelle ausgeführt.

```
UPDATE YZ3STOCK
  SET: BRANCH = 'S' WHERE WKZ = '123456',
       CURRENCY = 'DEM' WHERE WKZ = '888888'.
```

8.3.8 READ auf interne Tabellen

In internen Tabellen kann mit READ ein Satz mit einem beliebigen Schlüssel gelesen werden. Dazu sind die Nicht-Suchfelder aber mit dem Leerzeichen zu füllen. Dies muss mit dem Kommando

```
MOVE SPACE TO record.
```

erfolgen. Eine Initialisierung mit

```
CLEAR record.
```

führt mitunter nicht zum gewünschten Erfolg, da in diesem Fall einige Felder (je nach Typ) nicht mit Leerzeichen, sondern mit anderen Zeichen gefüllt werden, nach denen dann automatisch gesucht wird. Üblicherweise wird dadurch kein Satz gefunden!

8.3.9 Falsche Struktur bei Form-Routinen

In Form-Routinen kann einer als Parameter übergebenen Feldleiste mittels des Schlüsselwortes `STRUCTURE` (oder einigen anderen Konstrukten) eine Struktur zugewiesen werden. Diese ermöglicht es, auf einzelne Felder der Feldleiste mit ihrem Namen zuzugreifen. Diese Strukturzuweisung hat keine typprüfende Funktion. Die einzige Prüfung, die beim Aufruf von Form-Routinen durchgeführt wird, betrifft die Länge des Parameters. Der reale Parameter darf nicht kürzer sein als die als Struktur angegebene Feldleiste. Es ist also möglich, eine völlig falsche Struktur anzugeben. Diese wird den realen Daten einfach überlagert. Passen Struktur und Feldleiste nicht zusammen, kann es zu Fehlinterpretationen der Daten kommen, wie das nachfolgende Beispiel zeigt.

```
REPORT BMPRIV20.
DATA: BEGIN OF S1,
  A(5) TYPE C,
  B(10) TYPE C,
END OF S1.

DATA: BEGIN OF S2,
  A(10) TYPE C,
  B(3) TYPE C,
END OF S2.

S1-A = 'aaaaa'.
S1-B = '1234567890'.

PERFORM F1 USING S1.
WRITE /.
WRITE /.
PERFORM F2 USING S1.

FORM F1 USING P STRUCTURE S1.
  WRITE / P-A.
  WRITE / P-B.
ENDFORM.
```

```
FORM F2 USING P STRUCTURE S2.
  WRITE / P-A.
  WRITE / P-B.
ENDFORM.
```

Dieser Report erzeugt folgende Ausgabe:

```
aaaaa
1234567890
aaaaa12345
678
```

8.3.10 Globale Daten

Globale Daten sind nur innerhalb des Programms global, in dem sie deklariert wurden. Wird beispielsweise aus einem Modul-Pool heraus ein Funktionsbaustein aufgerufen, kann in dessen Function-Pool ein gleichnamiges Datenfeld wie im rufenden Modul-Pool existieren, wobei beide unterschiedliche Inhalte haben können.

8.3.11 Globale Felder als Unterprogramm-Parameter

Parameter von Unterprogrammen sind oft Referenzparameter. Sie stellen somit nur Zeiger auf das eigentliche Datenfeld dar. Wird der Inhalt des Ursprungsfelds in verschachtelten Unterprogrammen geändert, hat dies auch Auswirkungen auf Unterprogramm-Parameter. Das folgende Programm demonstriert dieses Verhalten. Besonders häufig können Probleme auftreten, wenn Systemfelder (z.B. `SY-SUBRC`) an Referenzparameter übergeben werden, da diese Felder oft implizit durch diverse Anweisungen geändert werden.

```
report  zxxxxx001.
data: global value '1'.

perform sub1 using global.

form sub1 using parameter like global.
  write: / 'before:', parameter.
  perform sub2.
  write: / 'after :', parameter.
endform.

form sub2.
  global = '2'.
endform.
```

8.3.12 LOCAL

Die Anweisung LOCAL wird in Form-Routinen verwendet. Sie sorgt dafür, dass der Inhalt eines globalen Feldes in einem internen Zwischenspeicher gerettet und später bei Beendigung der Routine wieder zurückgeschrieben wird. Innerhalb der Routine kann das Feld daher beliebig benutzt werden. Probleme ergeben sich, wenn durch schlechten Programmierstil oder durch Zufall die LOCAL-Variable zusätzlich als Parameter übergeben wird:

```
REPORT BMPRIV04.
DATA V1(10) TYPE C.
V1 = 'A'.
WRITE: / 'vorher:  ', V1.
PERFORM TEST USING V1.
WRITE: / 'nachher  ', V1.

FORM TEST USING V2.
LOCAL V1.
  V1 = 'B'.
  WRITE: / 'test 1  ', V1.
  V2 = V1.
ENDFORM.
```

Der Inhalt des globalen Feldes V1 wird innerhalb der FORM-Routine geändert, da V1 der Routine als Referenzparameter mit dem Namen V2 übergeben wird. Dieses Verhalten ist erwünscht. Beim Verlassen der Routine wird wegen der LOCAL-Anweisung allerdings wieder der alte Inhalt in V1 gestellt. Derartige Nebeneffekte lassen sich durch Verwendung der Anweisung DATA statt LOCAL vermeiden, was auch einem besseren Programmierstil (echte lokale Variable) entspricht.

8.3.13 Löschen von Views (Release 2.2)

Wenn Views erzeugt werden, die später mit der Transaktion SM30 benutzt werden sollen, ist dazu ein Eintrag in der Datei TVDIR erforderlich. Dieser Eintrag sorgt für das Generieren einiger Dynpros zur Viewpflege. Wird die View später gelöscht, ist zusätzlich der Eintrag in der TVDIR zu löschen.

8.3.14 Geltungsbereich Funktionsgruppen

Funktionsgruppen bzw. deren interner Datenbereich gelten nur innerhalb der Grenzen des aufrufenden Programms. Durch mehrfache Aufrufe einer Funktionsgruppe aus verschiedenen Ebenen eines Programms heraus können durchaus unterschiedliche Instanzen und damit Datenbereiche einer Funktionsgruppe entstehen. Folgendes Beispiel soll das erläutern:

Die Funktionsgruppe 1 enthalte die Funktionen A und B, die Funktionsgruppe 2 die Funktion C. Innerhalb von A wird C aufgerufen, von dort aus wieder B. Dadurch entstehen drei Datenbereiche, da für den Aufruf von B aus C heraus eine neue Instanz der FG 1 erzeugt wird. Wenn also B Daten in globalen Daten der Funktionsgruppe ablegt, sind diese in A nicht verfügbar. In diesem Fall müssen Daten über den Memory (z.B. mit `EXPORT` und `IMPORT`) übergeben werden.

8.3.15 Externe Performs

Werden externe Unterprogramme aufgerufen, arbeitet der Kommandointerpreter den Deklarationsteil des externen Programms ab. Greift die externe Routine auf globale Variablen zu, sind immer diejenigen des Programms gültig, in dem sich die Unterroutine befindet, nicht die des aufrufenden Programms.

8.3.16 SY-DATAR

Die Systemvariable `SY-DATAR` wird dann automatisch auf „X“ gesetzt, wenn in einem Dynpro Daten eingegeben werden. Diese Variable wird bei jedem Dynprodurchlauf zurückgesetzt, auch wenn z.B. in einem `EXIT`-Modul durch eine Sicherheitsabfrage der Abbruch unterdrückt und das Dynpro von vorn bearbeitet wird. In diesem Fall wird `SY-DATAR` zurückgesetzt, obwohl Änderungen durchgeführt und noch nicht abgespeichert wurden.

8.3.17 Parameter und Selektionen bei Report-Aufrufen

Wird ein Report mit der Anweisung `SUBMIT` von einem Dynpro aus aufgerufen, so wird nicht geprüft, ob alle Parameter oder Selektionen des Reports wirklich gefüllt werden. Es wird auch nicht geprüft, ob alle im Aufruf notierten Parameter oder Selektionen tatsächlich im Report existieren. Tipppfehler führen so dazu, dass Werte nicht an den Report übergeben werden.

8.3.18 Datendeklarationen in Modulen

Module bilden bezüglich des Geltungsbereiches von Datenfeldern keine in sich abgeschlossene Einheit. Felder, die in einem Modul eines Modul-Pools deklariert werden, sind ab dem Zeitpunkt der Deklaration global gültig. Das bedeutet, dass bei mehrmaligem Aufruf des Moduls die Deklaration und damit auch die Zuweisung eines Initialwerts nur beim ersten Mal ausgeführt wird. Das ist auch dann der Fall, wenn das Dynpro, zu dem das Modul gehört, verlassen und anschließend erneut aufgerufen wird. Wenn in einem Modul aus Versehen ein Feld deklariert wird, das als Zählvariable für Schleifen oder Ähnliches dienen soll, muss dieses Feld mittels einer Zuweisung auf seinen Startwert gesetzt werden. Erfolgt die Zuweisung des Startwertes nur bei der Deklaration, kann es zu unerwünschten Folgen kommen.

8.3.19 SELECT ... INTO TABLE

Mit SELECT INTO TABLE kann das Ergebnis einer SELECT-Anweisung direkt in einer internen Tabelle abgelegt werden. Dabei muss der Aufbau der internen Tabelle zur Struktur der von SELECT gelieferten Ergebnismenge passen, denn die von SELECT gefundenen Datensätze werden ohne Berücksichtigung der Struktur in die interne Tabelle geschrieben.

8.3.20 Gebiets-Angabe für Datencluster

In den Kommandos IMPORT FROM DATABASE bzw. EXPORT TO DATABASE ist eine Gebietsangabe als ergänzender Schlüssel erforderlich. Dieser Wert ist als Konstante zu notieren, er kann nicht dynamisch als Feldinhalt übermittelt werden. Wird an der entsprechenden Stelle irrtümlich ein Feldname eingetragen, so wird dies nicht als Syntaxfehler beanstandet. Das Kommando interpretiert einfach die ersten beiden Zeichen des Feldnamens als Gebietsangabe. Der Feldinhalt wird nicht ausgewertet.

8.3.21 Falsche Parameter für Funktionsbausteine

Beim Aufruf von Funktionsbausteinen werden Parameter per Namensreferenz übergeben. Falls beim manuellen Eintragen der Parameternamen Tipppfehler auftreten, wird der Parameter vom Funktionsbaustein nicht übernommen. Falls der eigentlich zu übergebende Parameter im Funktionsbaustein ein optionaler Parameter ist, wird der Fehler weder durch eine Syntaxprüfung noch durch die Prüfungen zur Laufzeit entdeckt. Er bewirkt eine Fehlfunktion des Bausteins, da dort ein scheinbar übergebener Parameter nicht ankommt.

8.3.22 Keine Rückgabe von Funktionswerten

Funktionsbausteine ignorieren Parameter, die beim Aufruf zusätzlich zu den in der Schnittstellenbeschreibung deklarierten Parametern übergeben werden. Falls nach dem Einfügen eines Anweisungsmusters für einen Funktionsbaustein die Anweisung IMPORTING nicht durch Löschen des Kommentarzeichens aktiviert wird, erscheinen die IMPORTING-Parameter aus Sicht des rufenden Programms als EXPORTING-Parameter. Es findet daher keine Rückgabe von Werten in den Parametern statt. Im folgenden Beispiel bleibt der Parameter L_ID leer, da er wegen der fehlerhaften Kommentare einen EXPORTING-Parameter darstellt.

```
CALL FUNCTION 'NUMBER_GET_NEXT'
  EXPORTING
    NR_RANGE_NR             = '01'
    OBJECT                  = 'Z123'
    . . .
```

```
*   IGNORE_BUFFER           = ' '
* IMPORTING
    NUMBER                  = L_ID
    . . .
  EXCEPTIONS
    INTERVAL_NOT_FOUND      = 1
    . . .
```

8.3.23 Verbuchung

In Dialoganwendungen löst jeder Bildwechsel einen `COMMIT WORK` aus. Dadurch werden Änderungen in Datenbanktabellen festgeschrieben. Allerdings bewirkt dieser implizite `COMMIT WORK` nicht die Ausführung von Verbuchungsroutinen, beispielsweise von Funktionsbausteinen mit dem Zusatz `ON UPDATE TASK`. Diese werden nur durch die explizite Verwendung der Anweisung

```
COMMIT WORK.
```

ausgeführt.

8.3.24 Sekundärindizes

Sie können zu Dictionary-Tabellen zusätzlich zum immer vorhandenen Primärindex weitere Sekundärindizes anlegen, die beliebige Felder der Tabelle auswerten. Diese zusätzlichen Indizes können mittels des Flags UNIQUE als eindeutige Indizes deklariert werden. Das hat zur Folge, dass neu einzufügende Datensätze nicht nur bezüglich des eigentlichen Tabellenschlüssels eindeutig sein müssen, sondern auch hinsichtlich der Inhalte der Felder im Sekundärindex. Meist ist dieses Verhalten nicht erwünscht. Die Ursache ist häufig ein Entwurfs- oder Handlingfehler. Ein Hinweis auf einen unerwünschten eindeutigen Sekundärindex kann sein, dass beim Einfügen von Datensätzen sowohl das Kommando `MODIFY` als auch das Kommando `INSERT` einen Returncode 4 liefern.

8.3.25 Versionsabhängige Unterschiede bei SELECT

Das `SELECT`-Kommando kennt eine Variante, bei der die Liste der auf der Datenbank zu lesenden Felder dynamisch angegeben werden kann. Diese Variante erfordert eine `INTO`-Klausel. Bis zur Version 3.1x mussten die Datentypen der Felder in der `INTO`-Klausel nicht identisch mit den Datentypen der von der Datenbank gelesenen Felder sein. Ab 4.0 besteht diese Forderung allerdings. Das folgende Programm arbeitet daher bis zur R/3-Version 3.1x korrekt, verursacht ab 4.0 aber einen Syntax-Fehler.

8.3.26 Dictionary-Probleme beim Übergang auf Version 4.x

Bei der Definition eigener Dictionary-Strukturen wird oft auf Datenelemente des SAP-Standards zurückgegriffen. Dies ist an sich korrekt und empfehlenswert, da auf diese Weise eigene Anwendungen immer mit den korrekten Datenformaten arbeiten. Beim Übergang von einer Version 3.x auf 4.x kann es dabei aber Probleme geben, da verschiedene Datenelemente umbenannt wurden. Anlass für diese Umbenennung ist die Tatsache, dass Datenelemente und Dictionary-Strukturen ab Version 4.0 in einem gemeinsamen Namensraum liegen und seitdem ein Datenelement nicht mehr den selben Namen wie eine Dictionary-Tabelle haben darf.

Die Fehlerbehebung muss manuell erfolgen, es existiert kein Automatismus. In den meisten Fällen reicht es aus, den alten Namen des Datenelements als Bestandteil eines Suchmusters zu verwenden, da die neuen Namen oft durch Anhängen eines Postfix entstanden.

Kurzreferenz

9

Der Umfang der vollständigen Referenz zu allen ABAP-Kommandos entspricht in etwa dem Umfang dieses Buches. Da Verfügbarkeit und Funktionsumfang vieler Kommandos vom Releasestand des Systems abhängen und Sie die jeweils gültige Dokumentation jederzeit über die Online-Hilfe des Systems abrufen können, finden Sie in diesem Abschnitt lediglich eine Aufstellung der ABAP-Kommandos und eine kurze Beschreibung. Dies reicht aus, um die Mächtigkeit der Sprache zu erkennen und einen Überblick über die Kommandos und ihre Varianten zu erlangen. Alle Kommandos und Kommando-Varianten, die im Zusammenhang mit ABAP Objects nicht mehr benutzt werden dürfen, wurden weggelassen. Das gilt auch dann, wenn bestimmte Konstrukte außerhalb von ABAP Objects (also außerhalb von Klassen) noch zulässig sind.

9.1 Beschreibung der Metasymbole

Die Kommandosyntax wird in diesem Kapitel durch eine Metasprache beschrieben. Die teilweise sehr komplizierte Syntax lässt sich in manchen Fällen auch mit diesem Hilfsmittel nicht völlig eindeutig darstellen. Insbesondere bezüglich der zulässigen Kombinationen der diversen Zusätze und Parameter und ihrer Anzahl gibt es mitunter Mehrdeutigkeiten. Sie sollten daher im Zweifelsfall auf verbale Beschreibung in diesem Kapitel oder die Online-Hilfe des R/3-Systems zurückzugreifen.

Folgende Symbole und Schriftstile werden in der Syntaxbeschreibung benutzt:

- Fette Schrift: Bestandteile des Kommandos, die in dieser Form notiert werden müssen.
- Kursive Schrift: Platzhalter für andere Elemente, z.B. Feldnamen.
- Normale Schrift: Metasymbole; Beschreibung siehe Tabelle 9.1.

Symbol	Bedeutung
[ ]	In eckige Klammern werden optionale Parameter und Zusätze eingeschlossen. Diese Elemente können maximal einmal auftreten.
{ }	Die in geschweiften Klammern stehenden Elemente können ein- oder mehrmals auftreten.
|	Alternativauswahl. Es kann nur eines der Elemente vor oder nach dem Pipe-Symbol benutzt werden.
( )	Gruppierung von Elementen.
...	Beliebige Wiederholung des Elements vor den Punkten.

Tabelle 9.1
Verwendete Metasymbole

9.2 Verweise und Parameterwiederholung

Die Programmiersprache ABAP kennt zwei spezielle syntaktische Elemente, die in vielen verschiedenen Kommandos benutzt werden können. Es handelt sich dabei um den Verweis und die Parameterwiederholung.

Ein Verweis besteht aus einem in runde Klammern eingeschlossenen Feldnamen. Zur Laufzeit wird der Verweis durch den Inhalt des jeweiligen Feldes ersetzt. Auf diese Weise können Sie Kommandos dynamisch, also zur Laufzeit der Anwendung, modifizieren. Dabei dürfen keine Trennzeichen zwischen den Klammern und dem Feldnamen stehen.

Die Parameterwiederholung ermöglicht es, in einigen Kommandos mehrere Parameter zu benutzen. Dazu wird das Kommando mit einem Doppelpunkt abgeschlossen. Die Parameter werden durch Komma getrennt. Ein Beispiel:

```
MOVE: A TO B,
      C TO D.
```

Es ist nicht gesagt, dass nur nach dem ersten Schlüsselwort ein Doppelpunkt stehen darf. In einigen – von SAP leider nicht explizit beschriebenen Fällen – kann sich die Parameterwiederholung nur auf Teile des Kommandos erstrecken. So ist beispielsweise auch das Kommando

```
MOVE A TO: B, C.
```

korrekt. Es füllt die Felder B und C mit dem Inhalt von Feld A.

Bedauerlicherweise existiert keine allgemeine Regel, in welchen Kommandos diese beiden Sprachelemente zugelassen sind und in welchen nicht. Beachten Sie in diesem Zusammenhang bitte die Beschreibungen der Kommandos.

9.3 Kommandoübersicht ABAP

```
ADD value TO field.
```

Der Inhalt von value (Feld oder Konstante) wird zu field addiert, das Ergebnis wird in field abgespeichert.

```
ADD field_1 THEN field_2 UNTIL field_z GIVING field.
```

Die Inhalte der Felder field_1 bis field_z werden addiert, das Ergebnis wird in field gespeichert. Dabei ist field_1 das erste, field_z das letzte einer Reihe von Feldern mit identischem Abstand. Außerdem müssen alle Felder den gleichen Typ und die gleiche Länge haben. Der Abstand voneinander wird durch den Abstand von field_1 und field_2 bestimmt.

```
ADD field_1 THEN field_2 UNTIL field_z TO field.
```

Summenbildung wie bei vorangegangener Variante. Das Ergebnis wird zum gegenwärtigen Inhalt von field addiert.

```
ADD field_1 THEN field_2 UNTIL field_Z
  ACCORDING TO sel GIVING field.
```

Die Summenbildung und Zuweisung erfolgt wie bei Variante 2. Bei der Summation werden nur die Felder berücksichtigt, deren Index über die Selektionstabelle Sel angegeben wird. Das erste Feld hat den Index 1. Die Selektionstabelle muss mit SELECT-OPTIONS oder RANGES angelegt worden sein.

```
ADD field_1 FROM first TO last GIVING field.
```

Das Feld field_1 ist das erste einer Reihe unmittelbar aufeinander folgender Felder gleichen Typs und gleicher Länge. Die Werte first und last geben den Index des ersten und des letzten aufzusummierenden Feldes an. Das Ergebnis wird in field abgelegt.

```
ADD-CORRESPONDING record_1 TO record_2.
```

Es werden nur Felder berücksichtigt, deren Namen sowohl in record_1 als auch in record_2 vorkommen. Die Inhalte der Felder aus record_1 werden zu den gleichnamigen Feldern aus record_2 addiert.

```
ALIASES alias FOR interface~component.
```

Dieses Kommando erzeugt innerhalb des Deklarationsteils eines Objekts einen Alias-Namen für eine Interface-Komponente. Dies dient zur einfacheren Schreibweise oder zum transparenten Ersatz eines herkömmlichen Elements durch eine Interface-Komponente.

```
APPEND record TO itab
  [SORTED BY [field | (field_name)]]
  [ASSIGNING <fs> | REFERENCE INTO dref].
```

Es wird ein neuer Datensatz an die interne Tabelle itab angefügt. Die Daten werden dem angegebenen Arbeitsbereich record entnommen. Es existiert keine Einschränkung bezüglich der Zahl der Datensätze in der Tabelle.

Durch den Zusatz SORTED BY field wird der neue Datensatz gemäß dem Inhalt von field absteigend in die Tabelle einsortiert. Der SORTED BY-Zusatz beschränkt die Zahl der Datensätze in der Tabelle. Erreicht die Satzzahl den Wert des OCCURS-Parameters, so geht der letzte Datensatz verloren, wenn der neue Datensatz vor ihm einsortiert wird. Das Sortierfeld kann dynamisch angegeben werden. Dabei enthält der Parameter für SORTED BY den Namen des Sortierfelds. Der dynamische Parameter muss in runden Klammern stehen. Das Sortierkriterium kann durch Angabe von Offset und Länge eingeschränkt werden.

Die Zusätze ASSIGNING bzw. REFERENCE INTO füllen das Feldsymbol <fs> bzw. die Objektreferenz dref mit der Referenz auf das gerade eingefügte Tabellenelement.

```
APPEND INITIAL LINE TO itab.
```

An die interne Tabelle wird ein Datensatz angefügt, der mit Initialwerten gemäß der Struktur der Tabelle gefüllt ist.

```
APPEND LINES OF itab_1
  [FROM index_1] [TO index_2] TO itab_2.
```

Der Inhalt der internen Tabelle itab_1 wird an die interne Tabelle itab_2 angefügt. Mit den optionalen Zusätzen FROM und TO können Sie die anzufügenden Datensatznummern aus itab_1 angeben.

```
ASSIGN [LOCAL COPY OF]
  [field[+offset][(length)] |
  (field_name)] |
  objectref->(field)
  (classname)=>(fieldname)
  dref->*
  [INCREMENT n]
  TO <field_symbol>
    [CASTING typ]
    [CASTING TYPE typ]
    [CASTING DECIMALS dec]
    [CASTING LIKE field]
    [RANGE range].
```

Das Feldsymbol field_symbol zeigt auf den Inhalt eines Datenfelds. Ohne die Zusätze werden die Eigenschaften von field (Typ, Länge, Konvertierungsexit) übernommen. Diese Eigenschaften können mit den Zusätzen TYPE, CASTING und DECIMALS überschrieben werden. Je nach Zusatz wird das Quellfeld gemäß dem angegebenen Typ oder dem des Feldsymbols interpretiert oder aber der Typ des Feldsymbols wird zur Laufzeit festgelegt.

Der Zusatz RANGE sorgt für eine Überwachung von Feldgrenzen, falls Offset-Angaben benutzt werden. Der Zugriff wird auf den Bereich von range begrenzt.

Das Quellfeld kann durch direkte Namensangabe, dynamisch durch Übergabe des Namens in einem zusätzlichen Feld oder durch eine Datenreferenz angegeben werden. Weitere Möglichkeiten bestehen in der Verwendung einer Referenz auf ein Attribut eines Objekts oder ein statisches Attribut einer Klasse. Bei direkter Angabe des Quellfeldes können Offset- und Längenangaben benutzt werden. Das Quellfeld kann auch dynamisch angegeben werden (dynamischer ASSIGN), wobei aber keine Längen- und Offsetangaben benutzt werden dürfen. Bei dynamischer Angabe des Feldnamens wird das Feld in mehreren Ebenen (lokaler Datenbereich, globaler Datenbereich, TABLES-Namensraum, externe Namensräume) gesucht.

Durch den optionalen Zusatz LOCAL COPY OF wird vor der Zuweisung eine Kopie von field angelegt. Zuweisungen zum Feldsymbol verändern also nicht den Inhalt des Originalfelds. Dieser Zusatz kann nur in Unterprogrammen benutzt werden. Das verwendete Feldsymbol muss lokal im Unterprogramm definiert werden.

Der INCREMENT-Zusatz verschiebt die Position der Referenz um die Länge des Bezugsfeldes, wobei ein Wiederholungsfaktor angegeben werden muss.

```
ASSIGN COMPONENT index | name OF STRUCTURE record
  TO <field_symbol>.
    [CASTING typ]
    [CASTING TYPE typ]
    [CASTING DECIMALS dec]
    [CASTING LIKE field]
    [RANGE range].
```

Diese Variante der ASSIGN-Anweisung entspricht der vorangegangenen. Allerdings zeigt das Feldsymbol nicht auf ein einzelnes Feld, sondern auf das Teilfeld einer Feldleiste. Das Teilfeld kann entweder über seinen Index (Startwert 1) oder aber über seinen Namen angegeben werden.

```
AT LINE-SELECTION.
```

Dies ist eine Zeitpunktanweisung der Listenerstellung. Der zugehörige Anweisungsblock wird immer dann prozessiert, wenn in einer Liste eine gültige Zeile selektiert wird (Doppelklick mit der Maus, Auslösen des Funktionscodes PICK, üblicherweise mit F2 oder Einfachklick auf einen als Hotspot deklarierten Bereich).

```
AT USER-COMMAND.
```

Der Anweisungsblock dieser Zeitpunktanweisung wird abgearbeitet, wenn der Anwender eine Funktion auslöst oder eine Eingabe im OK-Code-Feld macht.

Sie wird allerdings nur für Funktionscodes wirksam, die nicht durch das System abgefangen werden.

```
AT PFxx.
```

Diese Zeitpunktanweisung wird prozessiert, wenn eine Funktionstaste betätigt wurde, die mit dem Funktionscode PFn belegt ist. Dabei ist n ein nummerischer Wert zwischen 00 und 99.

```
AT NEW field.
  ABAP-statements
ENDAT.
```

Diese Anweisung ist nur innerhalb einer LOOP-Schleife über eine interne Tabelle oder einen EXTRACT-Datenbestand sinnvoll. Dabei ist field ein Teilfeld der internen Tabelle oder des Datenbestandes. Die Anweisungen zwischen AT NEW und ENDAT werden ausgeführt, wenn sich der Inhalt von field oder einem Feld, das in der Definition der Struktur vor field steht, ändert. Der Vergleich wird zwischen dem aktuellen Datensatz und dem im vorhergegangenen Schleifendurchlauf bearbeiteten Datensatz durchgeführt. Es wird der Beginn einer neuen Gruppe erkannt.

```
AT END OF field.
  ABAP-statements
ENDAT.
```

Wie AT NEW, allerdings wird der Vergleich zwischen dem aktuellen und dem darauf folgenden Datensatz durchgeführt. Auf diese Weise wird das Ende einer Gruppe erkannt.

```
AT FIRST.
  ABAP-statements
ENDAT.
```

Diese Anweisung ist nur innerhalb einer LOOP-Schleife sinnvoll. Die Anweisungen dieses Blocks werden vor dem ersten Schleifendurchlauf ausgeführt.

```
AT LAST.
  ABAP-statements
ENDAT.
```

Diese Anweisung ist nur innerhalb einer LOOP-Schleife sinnvoll. Die Anweisungen dieses Blocks werden nach dem letzten Schleifendurchlauf ausgeführt.

```
AT fieldgroup [WITH fieldgroup_1].
  ABAP-statements
ENDAT.
```

Diese Anweisung ist nur innerhalb einer LOOP-Schleife über einen EXTRACT-Datenbestand sinnvoll. Der Anweisungsblock wird nur ausgeführt, wenn der

aktuelle Datensatz zu `fieldgroup` gehört, also mit `EXTRACT fieldgroup` erstellt wurde. Der `WITH`-Zusatz sorgt zudem dafür, dass der Anweisungsblock nur ausgeführt wird, wenn auf den aktuellen Satz ein Datensatz folgt, der mit `EXTRACT fieldgroup_1` erstellt wurde.

```
AT SELECTION-SCREEN ON [parameter | selection].
```

Dies ist eine Zeitpunktanweisung, die nur in Reports mit Selektionsbildschirmen sinnvoll ist. Sie wird nach Eingabe im Selektionsbildschirm prozessiert. Wenn im Anweisungsblock eine Fehlernachricht ausgelöst wird, werden nur der angegebene Parameter oder die Selektion eingabebereit.

```
AT SELECTION-SCREEN ON BLOCK block.
```

Dies ist eine Zeitpunktanweisung, die nur in Reports mit Selektionsbildschirmen sinnvoll ist. Sie wird nach Eingabe im Selektionsbildschirm prozessiert. Wenn im Anweisungsblock eine Fehlernachricht ausgelöst wird, werden nur die Eingabefelder des angegebenen Blocks eingabebereit.

```
AT SELECTION-SCREEN ON END OF selection.
```

Dies ist eine Zeitpunktanweisung, die nur in Reports mit Selektionsbildschirmen sinnvoll ist. Zu jeder Selektion kann ein zweiter Selektionsbildschirm angezeigt werden, in dem mehrere Datensätze oder Intervalle der Selektionstabelle erfasst werden können. Der Zeitpunkt wird nach Beenden der Eingabe in das zweite Dynpro prozessiert. Da zu diesem Zeitpunkt alle eingegebenen Werte in der Selektionstabelle zur Verfügung stehen, können sie hier geprüft werden. Die Fehlernachricht macht das zweite Dynpro allerdings nicht eingabebereit!

```
AT SELECTION-SCREEN ON HELP-REQUEST
  FOR [parameter | selection-LOW | selection-HIGH].
```

Diese Zeitpunktanweisung erwartet als Zusatz entweder den Namen eines Parameters oder den einer Selektion. Im letzteren Fall muss das konkrete Eingabefeld mit dem Zusatz `-LOW` oder `-HIGH` hinter dem Namen der Selektion angegeben werden. Der Zeitpunkt wird dann prozessiert, wenn in einem Selektionsbildschirm der Cursor im benannten Feld steht und die Funktionstaste F1 für Hilfe betätigt wird.

```
AT SELECTION-SCREEN OUTPUT.
```

Dieser Zeitpunkt wird unmittelbar vor der Ausgabe des Selektionsbilds auf dem Bildschirm prozessiert.

```
AT SELECTION-SCREEN ON RADIOBUTTON GROUP radio_button_group.
```

Dies ist eine Zeitpunktanweisung, die nur in Reports mit Selektionsbildschirmen sinnvoll ist. Sie wird nach Eingabe im Selektionsbildschirm prozessiert. Wenn im Anweisungsblock eine Fehlernachricht ausgelöst wird, werden nur die zu `radio_button_group` gehörenden Auswahlfelder eingabebereit.

```
AT SELECTION-SCREEN ON VALUE-REQUEST
  FOR [parameter | selection-LOW | selection-HIGH].
```

Diese Zeitpunktanweisung erwartet als Zusatz entweder den Namen eines Parameters oder den einer Selektion. Im letzteren Fall muss das konkrete Eingabefeld mit dem Zusatz `-LOW` oder `-HIGH` hinter dem Namen der Selektion angegeben werden. Der Zeitpunkt wird dann prozessiert, wenn in einem Selektionsbildschirm der Cursor im benannten Feld steht und die Funktionstaste F4 (Eingabewerte) betätigt wird. Zur Laufzeit des Selektionsbildschirms wird rechts neben dem Eingabefeld das Symbol für die Eingabehilfe angezeigt.

```
AT SELECTION-SCREEN ON EXIT-COMMAND.
```

Diese Zeitpunktanweisung wird ausgeführt, wenn im Selektionsbild ein Funktionscode mit dem Funktionstyp „Exit-Kommando“ ausgelöst wurde.

```
AUTHORITY-CHECK OBJECT authority_object
  { ID authority_field FIELD value }
  [ ID authority_field DUMMY ].
```

Diese Anweisung prüft vorhandene Berechtigungen. Dabei ist `authority_object` der Name des Berechtigungsobjekts. Mit `authority_field` werden die Berechtigungsfelder des Berechtigungsobjekts angegeben. Nach Ausführung dieses Kommandos liefert `SY-SUBRC` Auskunft darüber, ob der aktuelle Inhalt der angegebenen Berechtigungsfelder mit `value` übereinstimmt. In diesem Fall besitzt der Anwender die erforderliche Berechtigung. Es können maximal zehn Berechtigungsfelder als Parameter an das Kommando übergeben werden. Falls für ein Feld keine Prüfung gewünscht wird, kann diese mit ID `authority_field DUMMY` ausgeschaltet werden.

```
BACK.
```

Positioniert den Cursor in der Listenverarbeitung auf die erste Zeile der aktuellen Seite nach der `TOP-OF-PAGE`-Verarbeitung. Bei Verwendung im Zusammenhang mit der `RESERVE`-Anweisung wird auf die erste nach `RESERVE` ausgegebene Zeile positioniert.

```
BREAK user_name.
```

Die Ausführung des Programms wird unterbrochen und der Debugger eingeschaltet, falls der aktuelle Anwender den angegebenen Namen hat. Damit können in der Entwicklungsphase benutzerspezifische Unterbrechungen eingefügt werden, welche die Arbeit anderer Anwender nicht beeinflussen. Im Batch- oder Verbuchungsmodus erfolgt keine Unterbrechung, sondern lediglich eine Syslog-Meldung.

```
BREAK-POINT [field].
```

Unbedingter Unterbrechungspunkt. Die Programmabarbeitung wird unterbrochen und der Debugger eingeschaltet. Im Batch- oder Verbuchungsmodus erfolgt keine Unterbrechung, sondern lediglich eine Syslog-Meldung. In diese Meldung kann der Inhalt von `field` übernommen werden.

```
CALL DIALOG dialog
  [ AND SKIP FIRST SCREEN ]
  [ EXPORTING { dialogfield_x [FROM program_field_x] } ]
  [ IMPORTING { dialogfield_x [TO program_field_x]   } ]
  [ USING itab MODE mode].
```

Der Dialogbaustein `dialog` wird aufgerufen. Falls dieser Baustein aus mehreren aufeinander folgenden Dynpros besteht, kann mit `AND SKIP FIRST SCREEN` das erste Dynpro dunkel prozessiert werden. Die Übergabe von Daten an den Dialogbaustein erfolgt mit `EXPORTING FROM`. Dabei wird einem im Dialogbaustein definierten Feld der Wert eines Programmfelds zugewiesen. Analog dazu erfolgt die Rückgabe von Werten vom Dialogbaustein an das rufende Programm mit `IMPORTING TO`. Falls die Feldnamen im Dialogbaustein und im rufenden Programm identisch sind, können die Zusätze `FROM` bzw. `TO` entfallen.

Der Zusatz `USING` übergibt eine interne Tabelle, die Daten im Batch-Input Format enthält. Der Dialogbaustein führt dann eine Massenverarbeitung durch. Der Wert von `mode` (siehe Tabelle 9.2) legt die Art der Anzeige bei der Verarbeitung von Batch-Input-Daten fest.

Modus	Beschreibung
A	Dynpros werden angezeigt (Standardvorgabe).
E	Dynpros werden nur angezeigt, wenn ein Fehler auftrat.
N	Es erfolgt keine Anzeige.

Tabelle 9.2
Abarbeitungs-Modus für den Batch-Betrieb

```
CALL FUNCTION function [
  [ EXPORTING    { parameter = value } ]
  [ IMPORTING    { parameter = field } ]
  [ TABLES       { parameter = itab  } ]
  [ CHANGING     { parameter = field } ]
  [ EXCEPTIONS [{ exception = value } ]
               [ OTHERS = value ]
               [ ERROR_MESSAGE = value ]]
] |
[
  [ PARAMETER-TABLE itab ]
```

```
  [ EXCEPTION-TABLE itab ]
].
```

Der Funktionsbaustein *function* wird aufgerufen. Beim Namen kann es sich entweder um eine in Hochkommas einzuschließende Zeichenkettenkonstante oder um ein Feld handeln, das den Namen enthält.

Nach EXPORTING werden die an den Funktionsbaustein zu übergebenden Werte aufgeführt, hinter IMPORTING die Felder zur Aufnahme der Rückgabewerte. Interne Tabellen sind nach TABLES anzugeben. Der Zusatz CHANGING deklariert kombinierte Im- und Exportparameter. Die vom rufenden Funktionsbaustein zu behandelnden Ausnahmen müssen nach EXCEPTIONS aufgeführt werden.

Als Alternative zur Aufzählung der diversen Parameter und Ausnahmen können diese auch in internen Tabellen an den Funktionsbaustein übermittelt werden. Dabei müssen diese Tabellen einem speziellen Datentyp entsprechen.

```
CALL FUNCTION function STARTING NEW TASK task_name
  [DESTINATION [dest | IN GROUP [group_name | DEFAULT ]]]
  [PERFORMING subroutine ON END OF TASK ]
  [EXPORTING  { parameter = value } ]
  [TABLES     { parameter = itab  } ]
  [EXCEPTIONS   exception = value [MESSAGE field ]].
```

Der Funktionsbaustein function wird in einem neuen Modus (gegebenenfalls mit DESTINATION auch in einem anderen System dest) asynchron gestartet. Das rufende Programm setzt seine Arbeit fort, ohne auf die Beendigung des aufgerufenen Funktionsbausteins zu warten. Die Rückgabe von Werten und Ausnahmen durch den gerufenen Funktionsbaustein ist daher nicht über Parameter des Funktionsbausteins möglich, sondern kann bei Bedarf über ein mit PERFORMING benanntes Unterprogramm erfolgen. Eine Ausnahme stellen Fehler dar, die beim Aufbau der Verbindung zu einem entfernten System entstehen können. Sie müssen mit EXCEPTIONS abgefangen werden. Mit dem Zusatz MESSAGE kann ein Feld benannt werden, in das beim Auftreten eines Systemfehlers ein erläuternder Text gestellt wird. Bei dieser Form des Aufrufs müssen alle beteiligten Systeme mindestens mit Release 3.0 des R/3-Systems arbeiten.

Beim Aufruf mit DESTINATION kann zusätzlich mit IN GROUP eine Gruppe von Applikationsservern benannt werden, auf denen der Funktionsbaustein parallel abgearbeitet wird.

```
CALL FUNCTION function IN UPDATE TASK
  [EXPORTING { parameter = value } ]
  [TABLES    { parameter = itab  } ].
```

Der Funktionsbaustein wird nicht sofort ausgeführt, sondern lediglich zur Abarbeitung vorgemerkt. Die übergebenen Werte werden automatisch zwischengespeichert. Erst bei einem COMMIT WORK wird der Baustein unter Verwendung der zwischengespeicherten Werte wirklich ausgeführt.

```
CALL FUNCTION function DESTINATION dest [
  [EXPORTING    { parameter = value } ]
  [IMPORTING    { parameter = field } ]
  [TABLES       { parameter = itab  } ]
  [CHANGING     { parameter = field } ]
  [EXCEPTIONS [{ exception = value } ]
              [OTHERS = value]
              [ERROR_MESSAGE = value]
              [MESSAGE field]]
] |
[
  [ PARAMETER-TABLE itab ]
  [ EXCEPTION-TABLE itab ]
].
```

Der angegebene Funktionsbaustein wird im System `dest` aufgerufen und ausgeführt. Diese Form der Ausführung wird als *Remote Function Call* (RFC) bezeichnet. Dabei kann `dest` eine Zeichenkettenkonstante oder ein Feld sein, das den Namen des zu rufenden Systems enthält. Das rufende Programm wartet auf die Beendigung der Abarbeitung, kann also auch Rückgabewerte entgegennehmen. Die Behandlung von Ausnahmen kann wie bei normalen Funktionsbausteinen erfolgen, aber auch die spezielle Behandlung einiger verbindungsbezogener Ausnahmen mit `MESSAGE` ist möglich. Auch hier können Parameter und Ausnahmen dynamisch in internen Tabellen an den Funktionsbaustein übermittelt werden.

```
CALL FUNCTION function IN BACKGROUND TASK
  [AS SEPARATE UNIT],
  [DESTINATION dest],
  [EXPORTING { parameter = value } ]
  [TABLES    { parameter = itab  } ].
```

Der Funktionsbaustein wird zur asynchronen Verbuchung vorgemerkt. Die übergebenen Werte werden zwischengespeichert. Bei Bedarf kann mit dem Zusatz `DESTINATION` die Ausführung des Funktionsbausteins auf einem anderen System erzwungen werden. Der Zusatz `AS SEPARATE UNIT` erzwingt die Ausführung des Funktionsbausteins in einer eigenständigen Transaktion.

```
CALL CUSTOMER-FUNCTION identifier [
  [EXPORTING    { parameter = value } ]
  [IMPORTING    { parameter = field } ]
  [TABLES       { parameter = itab  } ]
  [CHANGING     { parameter = field } ]
  [EXCEPTIONS [ { exception = value } ]
              [ OTHERS = value ]
              [ ERROR_MESSAGE = value ]]
] |
```

```
[
  [ PARAMETER-TABLE itab ]
  [ EXCEPTION-TABLE itab ]
].
```

Zur einfachen Erweiterung bestehender Anwendungen durch den Kunden stellt SAP so genannte Customer-Exits bereit. Dies sind Funktionsbausteine, die leer ausgeliefert und durch den Kunden ergänzt werden können. Die Schnittstelle, der Aufrufzeitpunkt und der konkrete Name werden durch SAP festgelegt. Bei `identifier` handelt es sich um eine dreistellige, nur aus Ziffern bestehende Zeichenkettenkonstante oder um ein Feld mit entsprechendem Inhalt. Die Zusätze zur Parameterübergabe entsprechen der Grundform der `CALL FUNCTION`-Anweisung. Zur Definition bzw. Aktivierung von Customer-Exits existieren die Transaktionen `SMOD` (bei SAP) und `CMOD` (beim Kunden).

Kundenfunktionen entsprechen technisch der Standardform der Funktionsbausteine. Daher können sie auch dynamisch mit Parametern und Ausnahmen versorgt werden.

```
CALL METHOD method [
  [EXPORTING    { parameter = value } ]
  [IMPORTING    { parameter = field } ]
  [CHANGING     { parameter = field } ]
  [RECEIVING      parameter = field   ]
  [EXCEPTIONS [ { exception = value } ]
              [ OTHERS = value ]
              [ ERROR_MESSAGE = value ] ]
] |
[
  [PARAMETER-TABLE param_tab ]
  [EXCEPTION-TABLE excep_tab ]
].
```

Aufruf einer Methode eines Objekts in ABAP Objects. Es bestehen hinsichtlich der Parameterübergabe starke Ähnlichkeiten zum Aufruf eines Funktionsbausteins. Die Parameterübergabe wird durch zwei Verfahren ermöglicht, die sich gegenseitig ausschließen. Es handelt sich dabei um die Einzelübergabe jedes Parameters durch die ersten 5 Zusätze oder die Übergabe der Parameter in internen Tabellen. Nach `EXPORTING` werden die Parameter aufgeführt, die an die Methode gesendet werden, nach `IMPORTING` die, welche die Methode zurückliefert. Die Kombination von Import- und Export-Parameter stellen die mit `CHANGING` zu übergebenden Parameter dar. Sofern die Methode nur einen einzigen Parameter zurückliefert, kann dieser mit `RECEIVING` entgegengenommen werden. Derartige Parameter können in Spezialfällen auch direkt als Wert des Methodenaufrufs in Ausdrücken benutzt werden. Ausnahmen werden ebenso wie bei Funktionsbausteinen mit Hilfe des Zusatzes `EXCEPTIONS` behandelt.

Zur Angabe des Methodennamens können sowohl statische als auch dynamische Varianten des Kommandos benutzt werden. Die verschiedenen Varianten unterscheiden sich bezüglich der möglichen Aufrufoptionen.

```
CALL METHOD method ( parameter ).
```

Mit dieser Form des CALL-Kommandos ist die vereinfachte Übergabe eines einzelnen Import-Parameters an eine ABAP-Objects-Methode möglich.

```
CALL METHOD method ( { parameter = value } ).
```

Dieses Kommando ruft eine Methode in ABAP Objects auf. Bei dieser Form des Aufrufs darf die Methode nur Import-Parameter besitzen, die innerhalb der zur Syntax des Kommandos gehörenden runden Klammern per Namensreferenz übergeben werden.

```
CALL METHOD OF object method
  [ = field ]
  [ EXPORTING { parameter = field } ]
  [ NO FLUSH ]
  [ QUEUE-ONLY ].
```

Die angegebene Methode des OLE-Objekts object wird aufgerufen. Bei Bedarf kann mit dem ersten Zusatz der Rückgabewert der Methode in einem Datenfeld abgelegt werden. Falls an die Methode Parameter übergeben werden sollen, ist dies ähnlich wie bei Aufrufen von Funktionsbausteinen mit EXPORTING möglich. Mehrere aufeinander folgende OLE-Aufrufe werden vom System gepuffert und gemeinsam übertragen, wenn eine Nicht-OLE-Anweisung folgt. Mit dem Zusatz NO FLUSH kann eine weitergehende Pufferung bewirkt werden. Die OLE-Aufrufe werden dann erst nach dem Kommando FREE ausgeführt. Der Zusatz QUEUE-ONLY unterdrückt in bestimmten Fällen die Rückübertragung von Werten an das ABAP-Programm.

```
CALL SCREEN screen
  [ STARTING AT x1 y1 [ ENDING AT x2 y2 ]].
```

Das Dynpro screen des aktuellen Hauptprogramms wird aufgerufen, wobei auch dessen Ablauflogik ausgeführt wird. Nach Beendigung der Dynpro-Verarbeitung wird mit der Anweisung nach der CALL-Anweisung fortgesetzt. Bei Bedarf kann das gerufene Dynpro mit STARTING AT auf dem Bildschirm positioniert werden. Es erscheint dann als Popup. Die Größe wird implizit durch die real benutzte Fläche des Dynpros (Dynpro-Attribute *BelegtUsed*) oder explizit durch die Parameter von ENDING AT bestimmt.

```
CALL SELECTION-SCREEN screen
  [ STARTING AT x1 y1 [ ENDING AT x2 y2 ]]
  [ USING SELECTION-SET set ].
```

Aufruf eines Selektionsbildschirms. Selektionsbildschirme werden mit den Anweisungen SELECTION SCREEN BEGIN OF SCREEN und SELECTION SCREEN END OF SCREEN definiert.

```
CALL TRANSACTION transaction
   [ AND SKIP FIRST SCREEN ]
   [ USING itab
       [ MODE mode]
       [ OPTION FROM params]
       [ UPDATE update]
       [ MESSAGES INTO message_table ]].
```

Die angegebene Transaktion wird aufgerufen. Nach Beendigung der gerufenen Transaktion wird das aktuelle Programm mit der Anweisung nach der CALL-Anweisung fortgesetzt. Der Name der Transaktion kann entweder durch eine Zeichenkettenkonstante oder ein Feld mit entsprechendem Inhalt angegeben werden. Der Zusatz AND SKIP FIRST SCREEN überspringt das erste Dynpro der gerufenen Transaktion, falls dort alle Eingabefelder durch geeignete Mechanismen (z.B. Get/Set-Parameter) mit gültigen Werten versorgt wurden.

Der Zusatz USING übergibt der gerufenen Transaktion eine Tabelle mit Dynpros im Batch-Input-Format. In diesem Fall kann das Verhalten der Transaktion durch einige weitere Parameter (siehe Tabelle 9.3) modifiziert werden.

	Parameter	Wirkung
	A	Dynpros anzeigen.
	E	Dynpro nur im Fehlerfall anzeigen.
	N	Dynpro nicht anzeigen.
UPDATE	A	Asynchrone Verbuchung.
	S	Synchrone Verbuchung.

Tabelle 9.3
Zusätze für Batch-Verarbeitung

Alternativ zu den Kommando-Zusätzen MODE und UPDATE kann die gerufene Transaktion auch durch Werte in der Struktur *params* gesteuert werden, die mit dem Zusatz OPTIONS FROM verwendet wird.

Da bei der Batch-Verarbeitung keine Fehlernachrichten auf dem Bildschirm ausgegeben werden können, ist es möglich, diese per Kommandozusatz MESSAGES INTO in einer vorhandenen internen Tabelle zu sammeln. Diese Tabelle muss die Struktur BDCMSGCOLL besitzen. Eine derartige Struktur ist im Dictionary definiert.

```
CALL TRANSFORMATION trans | (name)
  [ PARAMETERS ( { param = value } | (itab) ) ]
  [ OBJECTS ( { param = object } | (itab) ) ]
  SOURCE XML xmlref
  RESULT ( XML xmlref | ( {result = target} | (itab) ).
```

Mit diesem Kommando ist es möglich, auf einen XML-Datenbestand eine XSL-Transformation anzuwenden. Die aufzurufende Transformation ist ein eigenständiges Repository-Objekt. Der Transformation können Parameter oder Objekte übergeben werden, die diese zur Laufzeit auswertet. Über der Parameter SOURCE werden die Eingabedaten bestimmt, während durch den Parameter RESULT festgelegt wird, wo das Ergebnis der Transformation abzulegen ist.

```
CASE field.
```

Fallunterscheidung. Die Anweisung CASE leitet einen komplexen Anweisungsblock ein. Dieser Block wird mit ENDCASE abgeschlossen. Innerhalb des Blocks kann mit der Anweisung WHEN value der Inhalt von field mit value verglichen werden. Stimmen beide überein, werden die Anweisungen bis zum nächsten WHEN-Zweig ausgeführt. Mit der Anweisung WHEN OTHERS, die als letzter WHEN-Zweig notiert werden muss, kann ein Anweisungsblock für alle anderen Werte eingeleitet werden.

```
CATCH exception { exception } [ INTO ref ].
```

Innerhalb eines TRY-Blocks werden mit der Anweisung CATCH eine oder mehrere klassenbasierte Ausnahmen abgefangen. Nach CATCH folgen Anweisungen, um eben diese Ausnahmen zu behandeln. Bei Bedarf können Daten, die von der Ausnahme geliefert werden, in eine Objektreferenz *ref* abgelegt werden. Sie stehen dann innerhalb des Anweisungsblocks, mit dem die Ausnahme behandelt wird, zur Verfügung.

```
CATCH SYSTEM-EXCEPTIONS  { systemexception = returncode }
```

Diese Anweisung dient zum Abfangen von Laufzeitfehlern herkömmlicher ABAP-Kommandos. In der Parameterliste werden die zu prüfenden Laufzeitfehler und ein beliebiger nummerischer Rückgabewert aufgeführt. Falls der entsprechende Laufzeitfehler innerhalb des folgenden CATCH - ENDCATCH-Blocks auftritt, wird mit der Anweisung nach der ENDCATCH-Anweisung fortgesetzt. Dabei wird die Systemvariable SY-SUBRC mit dem zum Laufzeitfehler zugeordneten Rückgabewert gefüllt.

```
CHECK [ condition | selection ].
```

Diese Anweisung prüft den logischen Ausdruck oder die Selektion. Im letzteren Fall ist zu beachten, dass die Tabellen-Kopfzeile oder Feldleiste, gegen die geprüft werden soll, bereits bei der Deklaration der Selektionstabelle angegeben wurde. Ergibt die Prüfung den Wert für wahr, dann wird mit der nächsten Anweisung fortgesetzt, falls nicht, hängt die weitere Reaktion von der Art des

aktuellen Anweisungsblocks ab. In Schleifen wird zum Schleifenanfang gesprungen und damit der nächste Durchlauf begonnen. Innerhalb von Modularisierungseinheiten (Unterprogramme, Module, Funktionsbausteine oder Zeitpunktanweisungen) erfolgt der Aussprung aus der Modularisierungseinheit.

```
CHECK SELECT-OPTIONS.
```

Diese Anweisung ist nur nach einem GET-Zeitpunkt bei der Verarbeitung logischer Datenbanken sinnvoll. Geprüft wird der Inhalt des aktuellen Datensatzes für die nach GET angegebene Datenbanktabelle gegen alle Selektionen für diese Tabelle. Diese Form der Prüfung wird durch die so genannte *freie Abgrenzung* überflüssig.

```
CLASS classname DEFINITION
  [ PUBLIC ]
  [ INHERITING FROM superclass ]
  [ ABSTRACT ]
  [ FINAL ]
  [ CREATE [ PUBLIC | PROTECTED | PRIVATE ] ]
  [ FRIENDS | GLOBAL FRIENDS | LOCAL FRIENDS
      object_type { object_type } ].
```

Mit der Anweisung CLASS ... DEFINITION wird die Definition einer Klasse in ABAP Objects eingeleitet. In der Definition werden alle Felder deklariert sowie die Methoden mit ihren Schnittstellen definiert.

Der Zusatz PUBLIC wird nur durch den Class Builder gesetzt. Er kennzeichnet Klassen, die global im gesamten System bekannt sind. Durch INHERITING FROM wird der Vererbungsmechanismus genutzt. Die neue Klasse erbt alle Eigenschaften der angegebenen Superklasse. Der Zusatz ABSTRACT verbietet, dass von dieser Klasse Instanzen erzeugt werden können. Klassen, die mit dem Zusatz FINAL versehen sind, können nicht mehr als Superklasse in einer Vererbungskette benutzt werden. Sie beenden somit Vererbung. Der CREATE-Zusatz legt fest, wo überall Instanzen der Klasse erzeugt werden dürfen.

Die diversen FRIENDS-Zusätze ermöglichen es den als Parameter aufgeführten Klassen, auf private bzw. geschützte Attribute der aktuellen Klasse zuzugreifen.

```
CLASS classname DEFINITION DEFERED.
```

Diese Variante des CLASS-Kommandos dient lediglich dazu, den Namen der neuen Klasse bekannt zu machen. Die eigentliche Definition erfolgt später.

```
CLASS classname DEFINITION LOAD.
```

Der Zusatz LOAD lädt eine Klasse, falls ein Zugriff auf statische Komponenten erfolgt oder die Klasse zur Definition einer Ereignisbehandlungsroutine benötigt wird.

```
CLASS classname IMPLEMENTATION.
```

Mit dieser Anweisung wird der Implementierungsabschnitt einer Klasse in ABAP Objects eingeleitet. In diesem Abschnitt erfolgt die Programmierung der einzelnen Methoden.

```
CLASS-DATA data definitions.
```

Innerhalb einer Klasse legt diese Anweisung statische Klassenattribute an. Die Syntax der Datendefinitionen entspricht der DATA-Anweisung

```
CLASS-EVENTS event.
```

Definieren von statischen (Klassen-) Ereignissen innerhalb von ABAP Objects.

```
CLASS-METHODS method parameters
  [FOR EVENT event OF class].
```

Mit dieser Anweisung wird innerhalb einer Klassendefinition eine statische Methode definiert. Die Syntax ist mit der METHODS-Anweisung identisch. Auch Klassenmethoden können durch den Zusatz FOR EVENT als Behandlungsroutinen für Events dienen.

```
CLASS-POOL [MESSAGE-ID id].
```

Diese Anweisung leitet einen Klassen-Pool ein. Ein derartiges Programm und damit auch diese Anweisung wird vom Class Builder generiert und nicht manuell eingegeben.

```
CLEANUP.
```

Innerhalb eines TRY-Blocks kann durch die Anweisung CLEANUP ein Anweisungsblock eingeleitet werden, der dann ausgeführt wird, wenn eine Ausnahme nicht durch die vorangegangenen CATCH-Anweisungen abgefangen wurde.

```
CLEAR field [ WITH
  ( value [ IN CHARACTER MODE | IN BYTE MODE ] | NULL )].
```

Das Feld wird auf seinen typabhängigen Initialwert gesetzt. Der Zusatz WITH NULL bewirkt hingegen das Füllen des Feldes mit dem Null-Zeichen. Steht statt NULL ein Wert value (Feld oder Direktwert), so wird field mit dem ersten Byte von Wert gefüllt.

```
CLOSE CURSOR cursor.
```

Der Datenbank-Cursor cursor wird geschlossen. Dieses Kommando ist nur erforderlich, wenn der Cursor mehrmals zum Lesen von Datenbanksätzen benutzt werden soll.

```
CLOSE DATASET filename.
```

Die angegebene Datei wird geschlossen. Dieses Kommando ist nur erforderlich, wenn die Datei in einer Anwendung mehrmals hintereinander geöffnet werden soll. Im Kommando kann `filename` sowohl eine Zeichenkettenkonstante als auch ein Feld mit entsprechendem Inhalt sein.

```
CNT(field).
```

Die Anweisung `CNT` ist keine echte ABAP-Anweisung, sondern ein automatisch erzeugtes und gefülltes Feld. Die Auswertung des Feldinhaltes ist nur in einer `LOOP`-Schleife über einen sortierten Extrakt-Datenbestand sinnvoll. Außerdem muss `field` zum Sortierschlüssel gehören. Nach einem Gruppenende liefert `CNT` die Zahl der unterschiedlichen Werte in `field`.

```
COLLECT [record INTO] itab
  [ ASSIGNING <field symbol> ]
  [ REFERENCE INTO ref ].
```

Es wird ein neuer Datensatz in die interne Tabelle `itab` angefügt oder zu einem vorhandenen Eintrag mit identischem Schlüssel hinzuaddiert. Die Daten werden der Kopfzeile `itab` der internen Tabelle oder einer zusätzlich angegebenen Feldleiste `record` entnommen. Der Schlüssel besteht standardmäßig aus allen Feldern, die nicht Zahlenfelder sind, deren Typ also ungleich I, F oder P ist. Falls `COLLECT` in der internen Tabelle einen Satz findet, dessen Schlüssel mit dem des einzufügenden Datensatzes übereinstimmt, werden die Werte aller Zahlenfelder des neuen Satzes zu denen des bereits vorhandenen addiert. In diesem Fall wird kein neuer Satz in die interne Tabelle eingefügt. Es besteht keine Einschränkung bezüglich der maximalen Satzzahl in der Tabelle.

Nach erfolgreicher Ausführung der Anweisung kann mit den beiden Zusätzen ein Verweis auf die eingefügte Zeile entgegengenommen werden.

```
COMMIT WORK [ AND WAIT ].
```

Mit dieser Anweisung werden alle Datenbankänderungen endgültig festgeschrieben. Alle Verbuchungsroutinen (`PERFORM ON COMMIT`, `CALL FUNCTION IN UPDATE TASK`) und vorgemerkte Hintergrundverarbeitungen (`CALL FUNCTION IN BACKGROUND TASK`) werden ausgeführt. Datenbanksperren werden freigegeben. Der Bereich zwischen zwei `COMMIT WORK`-Anweisungen wird auch als logische Verarbeitungseinheit (*logical unit of work*, *LUW*) bezeichnet.

Eine Ausnahme ist der Aufruf von `COMMIT WORK` in Dialoganwendungen, die mit `CALL DIALOG` aufgerufen wurden. Dort werden nur Datenbanksperren freigegeben. Der echte Commit, also das Bestätigen von Datenbankänderungen, und der Start von Verbuchungsroutinen erfolgen erst nach einem `COMMIT WORK` im rufenden Programm.

Der Zusatz `AND WAIT` veranlasst die Anwendung, auf die Beendigung aller Verbuchungsvorgänge zu warten.

```
COMMUNICATION ...
```

Die unterschiedlichen Varianten der COMMUNICATION-Anweisung ermöglichen die Kommunikation mehrerer Programme auf Grundlage des CPI-C Protokolls (CPI-C: Common Programming Interface-Communication). Folgende Kommando-Varianten existieren:

```
... INIT DESTINATION dest ID ident [ RETURNCODE ret ].
```

Aufbau der Verbindung zum System dest. In ident wird eine Kennnummer für die Verbindung zurückgeliefert. Bei Bedarf kann der Returncode mit dem gleichnamigen Zusatz in ein ausdrücklich benanntes Feld ret übergeben werden. In der Regel ist der Rückkehrcode in SY-SUBRC verfügbar.

```
... ALLOCATE ID ident [ RETURNCODE ret ].
```

Die initialisierte Verbindung wird eingerichtet.

```
... ACCEPT ID ident [ RETURNCODE ret ].
```

Mit dieser Variante wird eine angeforderte Verbindung aufgenommen und initialisiert.

```
... SEND ID ident BUFFER buffer
      [ RETURNCODE ret ]
      [ LENGTH length].
```

Der Inhalt von buffer wird an die Gegenstelle gesendet. Mit dem Zusatz LENGTH kann die Zahl der zu übertragenden Zeichen angegeben werden.

```
... RECEIVE ID ident
      BUFFER buffer
      DATAINFO info
      STATUSINFO status
        [ RETURNCODE ret ]
        [ LENGTH length ]
        [ RECEIVED number ]
        [ HOLD ].
```

Es werden die von der Gegenstelle gesendeten Daten empfangen und in buffer abgelegt. Die Felder info und status liefern Informationen zur Übertragung. Mit dem Zusatz LENGTH können Sie die Anzahl der zu lesenden Zeichen einschränken, RECEIVED ermittelt die Zahl der tatsächlich gelesenen Zeichen. Mit HOLD wird der asynchrone Empfangsmodus abgeschaltet, der empfangende Prozess wartet bis zum Ende der Datenübertragung.

```
... DEALLOCATE ID ident [ RETURNCODE ret ].
```

Die Verbindung wird geschlossen und abgebaut. Bei Bedarf kann der Erfolg der Aktion über einen Rückgabewert ausgewertet werden.

```
COMPUTE field = expression.
```

Das (arithmetische) Ergebnis von Ausdruck wird berechnet und in Feld abgelegt. Auf die Anweisung COMPUTE kann verzichtet werden, wodurch sich die Kurzform

```
field = expression.
```

ergibt.

```
COMPUTE ref1 ?= ref2.
```

Es erfolgt eine Zuweisung zwischen Referenzvariablen. Dabei findet eine Typumwandlung statt.

```
CONCATENATE { value } INTO field
  [ SEPARATED BY separator ]
  [ IN BYTE MODE | IN CHARACTER MODE ].
```

Alle Werte (Feldinhalte oder Konstanten) werden als Zeichenketten interpretiert und zu einer Zeichenkette verbunden. Dabei werden abschließende Leerzeichen ignoriert. Das Ergebnis wird in field abgelegt. Mit SEPARATED BY kann eine Zeichenkette definiert werden, die als Trennzeichen zwischen die einzelnen Elemente eingefügt wird.

```
CONDENSE field [ NO-GAPS ].
```

Der Inhalt von field wird unabhängig vom tatsächlichen Typ als Zeichenkette interpretiert. Mehrere aufeinander folgende Leerzeichen werden zu einem einzigen zusammengefasst oder ganz unterdrückt (NO-GAPS). Das Ergebnis steht wieder in field.

```
CONSTANTS field[(length)]
  [TYPE typ | LIKE like_field] VALUE ( value | IS INITIAL ).
```

Deklaration eines Datenfelds field mit konstantem Inhalt, der mit VALUE zugewiesen werden muss. Neben der Zuweisung eines konkreten Werts kann mit IS INITIAL der zum jeweiligen Datentyp gehörende Initialwert zugewiesen werden. Der Typ der Konstanten kann durch den zugewiesenen Wert, durch eine Typangabe oder durch Ableitung von einem vorhandenen Feld bestimmt werden.

```
CONSTANTS:
  BEGIN OF record,
    declarations ...
  END OF record.
```

Deklaration einer Feldleiste mit konstantem Inhalt. Jedem einzelnen Feld der Feldleiste muss bei der Deklaration des Teilfeldes mit VALUE ein Wert zugewiesen werden.

```
CONTEXTS context.
```

Mit dieser Anweisung wird ein Datenkontext in einem Programm bekannt gemacht.

```
CONTINUE.
```

Diese Anweisung ist nur in Schleifen sinnvoll. Sie bewirkt den Sprung zum Beginn der Schleife und den Start des nächsten Durchlaufs.

```
CONTROLS name TYPE control_typ.
CONTROLS name TYPE TABLEVIEW USING SCREEN screen_number.
CONTROLS name TYPE TABSTRIP.
```

Mit der Anweisung `CONTROLS` wird ein Objekt definiert, mit dem Daten visuell dargestellt werden können. Die Art des Objekts wird durch `control_typ` bestimmt. Bei diesem Typ handelt es sich nicht um einen der Standarddatentypen, sondern um den Namen eines von SAP programmierten Elements.

Zur Zeit werden zwei Controls unterstützt. Der Table View kann als Weiterentwicklung der Step Loops angesehen werden. Zur Deklaration eines Table Views muss die `CONTROL`-Anweisung mit zusätzlichen Parametern ergänzt werden. Ein Tab Strip ermöglicht die Programmierung eines mehrseitigen Dialogs, dessen Seiten mit Hilfe einiger Register oder Reiter angewählt werden können.

```
CONVERT DATE date INTO INVERTED-DATE field.
```

Das Datum in `date` (Feldinhalt oder Konstante) wird invertiert (9-er Komplement der internen Darstellung) und das Ergebnis in `field` abgelegt. Durch diese Umwandlung hat das jüngste Datum den nummerisch kleinsten Wert, was beim Sortieren nach Datum hilfreich sein kann.

```
CONVERT INVERTED-DATE date INTO DATE field.
```

Umwandlung eines invertierten Datums in den ursprünglichen Wert.

```
CONVERT TEXT value INTO SORTABLE CODE hexfield.
```

Dieses Kommando führt eine von der eingestellten aktuellen Sprache abhängige Konvertierung von `value` durch. Diese Konvertierung ermöglicht eine korrekte Sortierung, da sprachabhängige Sonderzeichen (z.B. Umlaute) berücksichtigt werden.

```
CONVERT TIME STAMP tstamp TIME ZONE zone
  INTO DATE date TIME time.
```

Wandelt einen Zeitstempel in ein Datum und eine Uhrzeit um.

```
CONVERT DATE date TIME time
  INTO TIME STAMP tstamp TIME ZONE zone.
```

Wandelt eine Datums- und Zeitangabe in einen Zeitstempel um.

```
CREATE DATA ref
  (TYPE type | TYPE (namefield) | LIKE field)
  [WITH [ UNIQUE | NON-UNIQUE ] key definition
    [INITIAL SIZE n ]].
```

Erzeugung eines Datenobjekts, auf das die Referenz `ref` zeigt. Der Typ des Datenobjekts wird durch statische oder dynamische Angabe eines Datentyps oder durch Bezug auf ein existierendes Feld bestimmt. Es kann sich sowohl um einfache als auch strukturierte bzw. Tabellentypen handeln. Die Referenz muss zuvor mit `DATA ref TYPE REF TO DATA` definiert worden sein. Der Zugriff auf derartige Datenobjekte ist nur über Feldsymbole möglich.

```
CREATE OBJECT classref
  [TYPE class | TYPE (namefield)].{}
```

Erzeugen einer Instanz einer Klasse. Dabei wird die Klasse entweder durch die bereits getypte Klassenreferenz bestimmt oder aber bei ungetypten Referenzen durch den Zusatz `TYPE` in Verbindung mit der statischen oder dynamischen Übergabe des Klassennamens bestimmt.

```
CREATE OBJECT object class [ NO FLUSH | QUEUE-ONLY ].
```

Erzeugt ein OLE2-Objekt, das mit anderen Kommandos weiterbearbeitet werden kann.

```
DATA field[(length)]
  [ TYPE typ | LIKE template ]
  [ VALUE value ]
  [ LENGTH len ]
  [ DECIMALS decimal_places ]
  [ TYPE REF TO ( DATA | type ) ]
  [ TYPE REF TO class ]
  [ LIKE REF TO referenc ]
  [ READ-ONLY ].
```

Deklaration eines elementaren Datenfelds `field`. Der Typ und andere Eigenschaften können mit den diversen Zusätzen bestimmt werden, falls die Standardvorgaben nicht verwendet werden sollen. Die Angabe von `length` legt die Größe des Datenfeldes fest. Dieser Wert ist in runde Klammern einzuschließen und muss ohne Trennzeichen unmittelbar nach dem Feldnamen stehen. Mit `TYPE` erhält das Objekt den vordefinierten oder benutzerdefinierten Typ, mit `LIKE` werden die Eigenschaften eines existierenden Datenobjekts übernommen. Mit `VALUE` ist gleichzeitig mit der Deklaration die Zuweisung eines Wertes möglich. Bei elementaren Feldern des Typs P legen Sie mit `DECIMALS` die Zahl der Dezimalstellen fest.

Die drei letzten Optionen stehen im Zusammenhang mit der objektorientierten Erweiterung von ABAP. Der Zusatz `TYPE REF TO DATA` erzeugt eine Referenz auf eine Datenobjekt. Mit `TYPE REF TO class` erzeugen Sie ein Feld, das auf

die Instanz einer Klasse zeigen kann. Der Zusatz READ-ONLY ist nur innerhalb des Deklarationsteils einer Klasse einsetzbar. Ein so gekennzeichnetes Feld kann zwar von außerhalb der Klasse gelesen, aber nur über Methoden der Klasse geändert werden.

```
DATA: BEGIN OF record,
  declarations ...
END OF record.
```

```
DATA record[(length)]
  [ TYPE typ | LIKE template ]
  [ TYPE LINE OF itab_typ | LIKE LINE OF itab ].
```

Mit diesen beiden Varianten der DATA-Anweisung wird eine Feldleiste angelegt. In der ersten Variante sind die Teilfelder der Feldleiste zwischen BEGIN OF record und END OF record zu deklarieren, wobei die bei anderen Varianten der DATA-Anweisung beschriebenen Zusätze benutzt werden können. Es ist also auch möglich, komplexe Elemente bis hin zu Tabellen in eine Feldleiste aufzunehmen. Die einzelnen Deklarationen werden durch ein Komma voneinander getrennt.

Die zweite Variante der Anweisung definiert die Struktur der Feldleiste entweder durch Bezug auf einen Feldleistentyp bzw. eine andere, bereits existierende Feldleiste, oder aber sie verwendet die Zeilenstruktur einer internen Tabelle.

```
DATA: BEGIN OF itab OCCURS roll_area,
  declarations ...
END OF itab [ VALID BETWEEN field_1 AND field_2 ].
```

```
DATA itab
  [ ( TYPE linetype | LIKE record ) OCCURS n
       [ WITH HEADER LINE ] ] |
  [ ( TYPE | LIKE ) RANGE OF ( type | field )
       [ INITIAL SIZE size][ WITH HEADER LINE ] ] |
  [  TYPE itabtype [ WITH HEADER LINE ] ] |
  [ ( TYPE | LIKE ) TABLE OF ( linetype | record ) ] |
  [ ( TYPE | LIKE ) tabkind OF ( linetype | lineobj )
      WITH [ UNIQUE | NON-UNIQUE ] key
      [ INITIAL SIZE size ][ WITH HEADER LINE ] ].
```

Mit diesen beiden DATA-Anweisungen werden interne Tabellen deklariert. Die erste Variante erfordert die Auflistung aller Elemente der internen Tabelle. Der Parameter roll_area gibt die Zahl der Datensätze an, die im Hauptspeicher gehalten werden sollen. Erst beim Überschreiten dieser Zahl wird auf den Swap-Bereich der Festplatte ausgelagert. Für Tabellen, die mit dieser Variante angelegt werden, wird automatisch eine Kopfzeile erzeugt. Mit dem Zusatz VALID BETWEEN werden zwei Felder der Tabelle bestimmt, mit deren Hilfe das Kommando PROVIDE ein Gültigkeitsintervall bildet.

Die anderen Varianten der Anweisung benutzen existierende Datentypen für Feldleisten oder interne Tabellen bzw. bereits vorhandene Feldleisten und Tabellen, um den Aufbau der neuen internen Tabelle zu beschreiben. In einigen Fällen muss eine OCCURS-Angabe erfolgen. Alle auf diese Weise erzeugten Tabellen erhalten standardmäßig keine eigene Kopfzeile. Diese muss bei Bedarf mit dem Zusatz WITH HEADER LINE definiert werden.

```
DATA: BEGIN OF COMMON PART name,
  declarations ...
END OF COMMON PART.
```

Diese Anweisung definiert einen Datenblock im globalen Speicherbereich. Dieser Datenblock kann von mehreren Programmen gemeinsam genutzt werden, wenn er in allen Programmen mit identischem Namen und identischer Struktur deklariert wird. In der Praxis wird ein Common Part in einer eigenen Datei deklariert, die per INCLUDE-Anweisung in alle Programme eingebunden wird, die den Common Part benutzen sollen. Falls nur ein Common Part benutzt werden soll, kann der Name entfallen. Die Struktur wird ebenso wie bei Feldleisten deklariert.

```
DEFINE name.
  ABAP-statements  ...
END-OF-DEFINITION.
```

Die Anweisung DEFINE leitet die Definition eines Makros ein. Dieses Makro besteht aus ABAP-Anweisungen. Die Anweisung END-OF-DEFINITION schließt die Definition des Makros ab. Im Makro können die Stellungsparameter &1 bis &9 stehen, die zur Übersetzungszeit (vor der ersten Ausführung) eines Programms durch die Anweisungen der Makrodefinition ersetzt werden.

```
DELETE FROM table | (namefield) WHERE condition [CLIENT
SPECIFIED].
```

Löschen von Datensätzen aus einer Datenbanktabelle. Es werden alle über die WHERE-Bedingung identifizierten Datensätze gelöscht. Der Name der Tabelle kann entweder als Direktwert in der Anweisung notiert oder in einem Datenfeld übergeben werden. Standardmäßig wird nur im aktuellen Mandanten gelöscht. Mit dem Zusatz CLIENT SPECIFIED wird das automatische Mandantenhandling abgeschaltet. Der Mandant muss in diesem Fall wie jedes andere Tabellenfeld auch in der Selektionsbedingung aufgeführt werden.

```
DELETE table | *table | (field_with_tablename)
  [CLIENT SPECIFIED]
  [FROM workarea].
```

Diese Anweisung löscht einen einzelnen Datensatz aus der angegebenen Tabelle. Der Name der Tabelle kann wiederum statisch in der Anweisung oder dynamisch in einem Datenfeld übergeben werden. Der vollständige Schlüssel des zu

löschenden Datensatzes muss im Kopfsatz der Tabelle oder in einem explizit angegebenen Arbeitsbereich (FROM ...) übergeben werden. Im Falle der dynamischen Übergabe des Tabellennamens ist die Angabe eines Arbeitsbereichs obligatorisch. Der Inhalt des Arbeitsbereichs wird zeichenweise in die Schlüsselfelder der Tabelle übertragen, daher muss der Arbeitsbereich alle Schlüsselfelder aufnehmen können. Es ist somit erforderlich, dass im Arbeitsbereich die Schlüsselwerte in der Reihenfolge und in der Länge stehen, wie sie im Data-Dictionary für die jeweilige Tabelle vereinbart wurden. Die innere Struktur des Arbeitsbereichs kann allerdings von der Struktur der Tabelle abweichen. Das bedeutet, dass der Arbeitsbereich nicht zwangsweise eine Feldleiste sein muss, sondern beispielsweise auch ein ausreichend langes Datenfeld vom Typ C oder N sein kann. Mit dem Zusatz CLIENT SPECIFIED wird das automatische Mandantenhandling abgeschaltet.

```
DELETE table | (field_with_tablename)
  FROM TABLE itab
  [CLIENT SPECIFIED].
```

Aus der angegebenen Datenbanktabelle werden alle Datensätze gelöscht, deren Schlüssel mit einem der Datensätze der internen Tabelle itab übereinstimmt. An die Datensätze der internen Tabelle werden dieselben Anforderungen gestellt wie an den Arbeitsbereich der vorher erläuterten Version der DELETE-Anweisung. Mit dem Zusatz CLIENT SPECIFIED wird wiederum das automatische Mandantenhandling abgeschaltet. Der Name der Datenbanktabelle kann statisch oder dynamisch übergeben werden.

```
DELETE table | *table VERSION field_with_tablename.
```

Diese Variante ermöglicht die dynamische Übergabe des Tabellennamens im Feld field_with_tablename, wobei allerdings einige Einschränkungen bezüglich des Namens bestehen. Da inzwischen auch die anderen Versionen der DELETE-Anweisung die dynamische Namensübergabe ermöglichen, wird diese Variante des Kommandos nur noch aus Kompatibilitätsgründen unterstützt und sollte für Neuentwicklungen nicht mehr verwendet werden.

```
DELETE itab
  [[INDEX index] |
   [WHERE condition][FROM Start ][TO last ]].
```

Löschen von Datensätzen aus einer internen Tabelle. Ohne Zusatz ist die Anweisung nur in einer LOOP-Schleife über die Tabelle sinnvoll. Gelöscht wird dann der jeweils aktuelle Satz. Mit dem Zusatz INDEX kann außerhalb von Schleifen ein einzelner Satz gezielt gelöscht werden, sofern seine Datensatznummer bekannt ist. Diese wird beispielsweise von einigen Varianten des Kommandos READ ermittelt. Die Zählung beginnt mit 1. Mit den alternativ zu INDEX verwendbaren Zusätzen WHERE, FROM und TO kann ein zu löschender Bereich be-

stimmt werden. In Verbindung mit WHERE sorgen die Angaben FROM bzw. TO dafür, dass nur der angegebene Bereich der internen Tabelle untersucht wird.

```
DELETE TABLE itab [ WITH TABLE KEY { key = value } ].
```

Aus der internen Tabelle wird der erste Datensatz gelöscht, der dem angegebenen Schlüssel entspricht.

```
DELETE TABLE itab [ FROM record ].
```

Aus der internen Tabelle wird der Datensatz gelöscht, dessen Schlüssel dem des Datensatzes in der Kopfzeile oder dem explizit angegebenen Arbeitsbereich entspricht.

```
DELETE ADJACENT DUPLICATES FROM itab
  [COMPARING [ field_1 ... field_n ]  |
             (field_with_fieldname) |
             COMPARING ALL FIELDS ]].
```

Aus der internen Tabelle werden mehrfach vorhandene Einträge entfernt. Als identisch gelten Einträge dann, wenn sie entweder:

- in den Inhalten aller Felder übereinstimmen, die einen elementaren Datentyp ungleich I, P oder F haben (Default-Funktion ohne COMPARING-Zusatz) oder
- in den Inhalten aller Felder übereinstimmen, die im Kommandozusatz COMPARING aufgeführt werden oder
- in allen Feldern übereinstimmen (COMPARING ALL FIELDS).

Dieses Kommando arbeitet nur dann korrekt, wenn die identischen Datensätze unmittelbar aufeinander folgen. Die interne Tabelle muss entsprechend dem anzuwendenden Vergleichskriterium vorsortiert sein.

```
DELETE FROM DATABASE table(area) ID key
  [CLIENT field].
```

Löschen eines so genannten Daten-Clusters aus der table und dem area. Beide Angaben sind Direktwerte, die beiden runden Klammern um area gehören zur Syntax der Anweisung. Der Cluster wird durch einen key identifiziert, den Sie als Direktwert oder Feldinhalt angeben können.

```
DELETE FROM MEMORY ID key.
```

Löschen eines Daten-Clusters, der im Hauptspeicher gehalten wird. Der Cluster wird durch einen key identifiziert.

```
DELETE FROM SHARED BUFFER table(area) ID key
  [CLIENT field].
```

Löschen eines Daten-Clusters aus dem transaktionsübergreifenden Puffer. Die Angaben table und area sind Direktwerte, die beiden runden Klammern um

area gehören zur Syntax der Anweisung. Der Cluster wird durch einen key identifiziert, den Sie als Direktwert oder Feldinhalt angeben können.

```
DELETE FROM SHARED MEMORY table(area) ID key
  [CLIENT field].
```

Löschen eines Daten-Clusters aus dem Shared Memory. Die Angaben table und area sind Direktwerte, die beiden runden Klammern um area gehören zur Syntax der Anweisung. Der Cluster wird durch einen key identifiziert, den Sie als Direktwert oder Feldinhalt angeben können.

```
DELETE DATASET filename.
```

Löschen der angegebenen Datei auf Betriebssystemebene. Es handelt sich dabei nicht um eine Tabelle, sondern meist um Dateien in Fremdformaten, z. B. Textdateien oder Binärdateien anderer Systeme.

```
DEMAND { context_field  = program_field } FROM CONTEXT
context
  [ MESSAGES INTO itab ].
```

Lesen von Werten aus einem Datenkontext.

```
DESCRIBE FIELD field
  [ LENGTH result_field [IN CHAR MODE | IN BYTE MODE]]
  [ TYPE result_field ]
  [ TYPE typ COMPONENTS number ]
  [ OUTPUT-LENGTH result_field ]
  [ DECIMALS result_field ]
  [ EDIT MASK result_field ]
  [ HELP-ID Id ].
```

Diese Anweisung ermittelt eine oder mehrere Eigenschaften eines Feldes und stellt diese in das Feld result_field. Es muss mindestens ein Zusatz benutzt werden. Werden gleichzeitig mehrere Zusätze angegeben, so sind natürlich unterschiedliche Ergebnisfelder notwendig.

```
DESCRIBE TABLE itab
  [LINES lines]
  [OCCURS roll_area]
  [KIND tablekind].
```

Füllt das Feld lines mit der Zahl der Datensätze der internen Tabelle itab und das Feld roll_area mit dem Wert des bei der Deklaration der internen Tabelle angegebenen OCCURS-Parameters. Über die Option KIND ermitteln Sie die Tabellenart (Standard-Tabelle, sortierte Tabelle oder Hash-Tabelle). Mindestens einer der Zusätze muss angegeben werden.

```
DESCRIBE DISTANCE BETWEEN field_1 AND field_2 INTO distance
[IN BYTE MODE | IN CHARACTER MODE].
```

Der Abstand zwischen den Feldern `field_1` und `field_2` wird in `distance` gestellt. Je nach Zusatz wird dabei entweder in Bytes oder in Zeichen gemessen. Diese Unterscheidung ist seit der Einführung von Uncode-Zeichen von Bedeutung, da diese auch länger als ein Byte sein können.

```
DESCRIBE LIST [INDEX index] NUMBER OF LINES lines.
```

Dieses Kommando schreibt die Zahl der List-Zeilen in das Feld `lines`. Bei Bedarf kann mit `INDEX` die List-Ebene angegeben werden, um auf andere als die gerade angezeigte Liste zuzugreifen.

```
DESCRIBE LIST [ INDEX index ] NUMBER OF PAGES pages.
```

Dieses Kommando schreibt die Zahl der Seiten der Liste in das Feld `pages`. Bei Bedarf kann mit `INDEX` die List-Ebene angegeben werden, um auf andere als die gerade angezeigte Liste zuzugreifen.

```
DESCRIBE LIST [ INDEX index ] LINE lines PAGE pages.
```

Dieses Kommando liefert die Seitennummer, auf der die angegebene Zeile zu finden ist. Auch hier kann mit `INDEX` die List-Ebene angegeben werden, um auf andere als die gerade angezeigte Liste zuzugreifen.

```
DESCRIBE LIST [ INDEX index ] PAGE pages
  [ LINE-SIZE result_field ]
  [ LINE-COUNT result_field ]
  [ LINES result_field ]
  [ FIRST-LINE result_field ]
  [ TOP-LINES result_field ]
  [ TITLE-LINES result_field ]
  [ HEAD-LINES result_field ]
  [ END-LINES result_field ].
```

Die diversen Zusätze zu diesem Kommando liefern Informationen über den Aufbau einer ausgewählten Seite einer Liste. Falls mehrere der Zusätze auf einmal benutzt werden, sind für die Aufnahme der Ergebnisse natürlich unterschiedliche Felder erforderlich. Die Rückgabewerte der einzelnen Zusätze zeigt die Tabelle 9.4.

```
DIVIDE field BY divisor.
```

Der Wert in `field` wird durch `divisor` dividiert. Das Ergebnis steht anschließend in `field`. Der Divisor kann sowohl ein Direktwert als auch ein Feld sein.

```
DO.
  ABAP-statements ...
ENDDO.
```

Zusatz	Liefert
LINE-SIZE	Zeilenbreite.
LINE-COUNT	Erlaubte Zeilenzahl.
LINES	Zahl der ausgegebenen Zeilen.
FIRST-LINE	Absolute Zeilennummer der ersten Zeile der Seite.
TOP-LINES	Zahl der im Seitenkopf ausgegebenen Zeilen (Titel + Spaltenüberschriften).
TITLE-LINES	Zahl der als Titel ausgegebenen Zeilen.
HEAD-LINES	Zahl der als Spaltenüberschrift ausgegebenen Zeilen.
END-LINES	Zahl der für Seitenende reservierten Fußzeilen.

Tabelle 9.4
Zusätze des Kommandos DESCRIBE LIST PAGE

Endlosschleife. Die Anweisungen zwischen DO und ENDDO werden wiederholt, bis die Schleife mit EXIT, STOP oder REJECT verlassen wird.

```
DO VARYING field FROM start NEXT distance.
  ABAP-statements ...
ENDDO.
```

Diese Endlosschleife wird ebenfalls so lange abgearbeitet, bis sie mit EXIT, STOP oder REJECT verlassen wird. In dieser Anweisung sind start und distance die Namen von Feldern. Während jedes Schleifendurchlaufs erhält field (ebenfalls ein Datenfeld) einen neuen Wert. Der erste Wert ergibt sich aus dem Inhalt des Feldes start. Das jeweils nächste Feld wird ermittelt, indem zur Speicheradresse des jeweils aktuellen Feldes der Abstand zwischen start und distance addiert wird. Falls field ein Wert zugewiesen wird, so wird diese Änderung auch an der Stelle im Hauptspeicher wirksam, von welcher der aktuelle Inhalt von field gelesen wurde. Mehrere VARYING-Zusätze können in einer DO-Anweisung kombiniert werden.

```
DO n TIMES.
  ABAP-statements ...
ENDDO.
```

Die Schleife wird n-mal durchlaufen.

```
DO n TIMES VARYING field FROM start NEXT distance.
  ABAP-statements ...
ENDDO.
```

Kombination der DO VARYING-Anweisung mit einer vorgegebenen Anzahl von Schleifendurchläufen. Die Zuweisung von Werten zu field erfolgt wie bei DO VARYING beschrieben.

```
EDITOR-CALL FOR itab
  [ TITLE title ]
  [ DISPLAY-MODE ]
  [ BACKUP INTO backup_tab ].
```

Die interne Tabelle itab, die nur aus Feldern des Typs C besteht und maximal 72 Zeichen breit sein darf, wird in den Programmeditor geladen und kann dort bearbeitet werden. Die Änderungen werden nur in der internen Tabelle gespeichert, wenn im Editor das Kommando SICHERN (Funktionstaste F11) ausgelöst wird. Mit TITLE kann eine Überschrift gesetzt werden. Der Zusatz DISPLAY-MODE verhindert eine Bearbeitung der Tabelle und erlaubt nur deren Anzeige. Der Zusatz BACKUP INTO erzeugt eine Sicherheitskopie der Tabelle in einer zweiten internen Tabelle.

```
EDITOR-CALL FOR REPORT program [ DISPLAY-MODE ].
```

Das angegebene Programm wird in den Programmeditor geladen und kann dort bearbeitet werden. Der Zusatz DISPLAY-MODE versetzt den Editor in den Anzeigemodus, in dem keine Bearbeitung des Programms möglich ist.

```
ELSE.
```

Die ELSE-Anweisung leitet innerhalb einer IF-Anweisung einen Anweisungsblock ein, der durchlaufen wird, wenn die nach IF angegebene Bedingung nicht erfüllt ist.

```
ELSEIF condition.
```

Diese Anweisung verknüpft die ELSE- mit einer neuen IF-Anweisung. Sie erlaubt den Aufbau geschachtelter Vergleiche. Sie kann jederzeit durch eine separate ELSE- und IF-Anweisung ersetzt werden.

```
END-OF-DEFINITION.
```

Diese Anweisung beendet die Definition eines Makros.

```
END-OF-PAGE.
```

Zeitpunktanweisung. Der zugehörige Anweisungsblock wird ausgeführt, wenn während der Ausgabe einer Liste der END-OF-PAGE-Bereich einer Seite erreicht wird oder wenn die RESERVE-Anweisung nicht die erforderliche Zahl freier Zeilen vorfindet. Diese Zeitpunktanweisung wird nicht wirksam, wenn LINE-COUNT auf den Standardwert 0 gesetzt wurde oder mit NEW-PAGE ein expliziter Seitenvorschub erfolgt.

```
END-OF-SELECTION.
```

Der Anweisungsblock dieser Zeitpunktanweisung wird ausgeführt, nachdem alle Anweisungen des eigentlichen Selektionsteils eines Reports ausgeführt wurden. Dies ist beispielsweise dann der Fall, wenn alle Datensätze einer logischen Datenbank gelesen wurden.

```
ENDAT.
```

Diese Anweisung beendet den Anweisungsblock einer `AT`-Anweisung.

```
ENDCASE.
```

Ende einer `CASE`-Verzweigung.

```
ENDCATCH.
```

Ende eines `CATCH`-Blockes.

```
ENDCLASS.
```

Ende eines mit `CLASS` begonnenen Definitions- oder Implementierungsblocks einer Klasse.

```
ENDDO.
```

Ende einer mit `DO` eingeleiteten Schleife.

```
ENDEXEC.
```

Ende einer Native-SQL-Anweisung.

```
ENDFORM.
```

Abschluss eines Unterprogramms.

```
ENDFUNCTION.
```

Ende eines Funktionsbausteins.

```
ENDIF.
```

Beendigung einer `IF`-Anweisung.

```
ENDINTERFACE.
```

Ende des Definitionsteils eines Interface in ABAP Objects.

```
ENDLOOP.
```

Ende einer `LOOP`-Schleife.

```
ENDMETHOD.
```

Ende einer Methodendeklaration.

```
ENDMODULE.
```

Ende der Deklaration eines Moduls.

```
ENDON.
```

Diese Anweisung beendet den Anweisungsblock einer ON-Anweisung.

```
ENDPROVIDE.
```

Abschluss einer PROVIDE-Schleife.

```
ENDSELECT.
```

Ende einer SELECT-Schleife.

```
ENDTRY.
```

Ende eines TRY-Blocks.

```
ENDWHILE.
```

Ende eines WHILE-Abschnitts.

```
EVENTS event EXPORTING {
  VALUE(parameter) [
    TYPE type | LIKE field [OPTIONAL | DEFAULT default]]}.
```

Diese Anweisung deklariert innerhalb einer ABAP-Objects-Klasse Instanzenereignisse. Ereignisse können Parameter erhalten, die stets Wertparameter sein müssen. Sie sind daher mit dem Schlüsselwort VALUE zu kennzeichnen und in runde Klammern einzuschließen. Zwischen den Klammern und dem Parameternamen dürfen keine Leerzeichen stehen. Der Parameter kann durch die TYPE- oder LIKE-Option typisiert werden. Außerdem kann er mit OPTIONAL als optionaler Parameter deklariert werden. Der Zusatz DEFAULT ermöglicht die Vorgabe eines Initialwerts.

```
EXEC SQL [ PERFORMING subroutine ].
  SQL-statement
ENDEXEC.
```

Zwischen EXEC und ENDEXEC steht eine so genannte Native-SQL-Anweisung. Diese Anweisung wird direkt an die Datenbank durchgereicht; sie muss also eventuelle Besonderheiten des benutzten Datenbanksystems berücksichtigen. Der Datenaustausch zwischen ABAP und Native-SQL erfolgt über ABAP-Datenfelder, denen in der SQL-Anweisung ein Doppelpunkt vorangestellt wird. Mit PERFORMING kann ein ABAP-Unterprogramm definiert werden, das für jeden Datensatz, den die SQL-Anweisung zurückliefert, einmal ausgeführt wird. Native-SQL-Anweisungen sind möglicherweise nicht portabel. Sie sollten daher nur dann benutzt werden, wenn keine andere Möglichkeit besteht.

```
EXIT.
```

Verlassen eines Verarbeitungsabschnitts. Die genaue Reaktion ist von der Position der Anweisung innerhalb des Programms abhängig. Steht diese An-

weisung in einer Schleife, so wird nur die Schleife beendet. Innerhalb von Modularisierungseinheiten (Unterprogramme, Anweisungsblock einer Zeitpunktanweisung) – aber außerhalb von Schleifen – führt ein `EXIT` zum Beenden der Verarbeitungseinheit. Innerhalb eines Reports beendet `EXIT` die weitere Verarbeitung und führt zur Anzeige der bisher erzeugten Liste.

```
EXIT FROM STEP-LOOP.
```

Diese Anweisung bezieht sich auf die Loop-Anweisung in der Ablauflogik eines Dynpros. Die entsprechende Schleife wird beendet, wodurch keine weiteren Datensätze in den Step-Loop-Bereich des Dynpros gestellt bzw. von dort gelesen werden.

```
EXPORT (itab) |
  { [newname = field] |
    (field | itab  [FROM source]) }
  TO DATA BUFFER buffer
  [COMPRESSION (ON | OFF)].
```

Felder oder interne Tabellen werden in eine Puffervariable geschrieben, die vom Typ `XSTRING` sein muss. Mit dem Zusatz `FROM`, der für jedes abzulegende Element einzeln angegeben werden muss, können Elemente in der Tabelle unter einem anderen Namen angelegt werden. Ähnliche Wirkung hat die Variante mit dem Gleichheitszeichen. Dabei werden Feldinhalte unter dem vor dem Gleichheitszeichen angegebenen Namen abgelegt.

```
EXPORT (itab) | {
  { [newname = field] |
    (field | itab  [FROM source]) }
  TO MEMORY [ID key]
  [COMPRESSION (ON | OFF)].
```

Felder oder interne Tabellen werden in einen globalen Speicherbereich geschrieben, von wo aus sie von anderen Anwendungen gelesen werden können. Damit ist ein anwendungsübergreifender Datenaustausch innerhalb der Sitzung eines Anwenders möglich. Mit dem Zusatz `FROM` bzw. der Variante mit dem Gleichheitszeichen können Sie ein Objekt unter anderen Namen ablegen. Der Zusatz `ID` versieht die abgelegten Objekte mit einem Identifikator, über den sie später auch wieder eingelesen werden können. Somit können beliebig viele Gruppen von Elementen im Speicher abgelegt werden. Das `EXPORT`-Kommando wirkt überschreibend. Bei mehrmaligem Export werden alle bereits vorhandenen Objekte mit demselben Identifikator gelöscht. Auch hier besteht die Möglichkeit, auf die Werteliste zu verzichten und alle zu übertragenden Werte in einer internen Tabelle zu übergeben.

```
EXPORT (itab) |
  { [newname = field] |
    (field | itab  [FROM source] ) }
```

```
  TO DATABASE table(area) [FROM record]
  [CLIENT client] ID key [FROM record2]
  [COMPRESSION (ON | OFF)].
```

Felder oder interne Tabellen werden in eine Datenbanktabelle `table` geschrieben. Sie werden dort in einem so genannten Gebiet `area` abgelegt. Mehrere Objekte können unter einem gemeinsamen Schlüssel `key` abgelegt werden. Mit dem Zusatz `CLIENT` wird explizit ein Zielmandant angegeben, falls nicht der aktuelle Mandant benutzt werden soll. Mit dem Zusatz `FROM`, der für jedes abzulegende Element einzeln angegeben werden muss, bzw. mit der Verwendung des Gleichheitszeichens können Elemente in der Tabelle unter einem anderen Namen angelegt werden. Die verwendete Datenbanktabelle muss einen vorgegebenen Aufbau besitzen. Der `FROM`-Zusatz hinter `TO DATABASE` bzw. `ID key` verwendet eine separate Feldleiste an Stelle des Tabellenarbeitsbereiches.

Alternativ zur separaten Angabe aller zu übertragenden Felder können alle Werte in einer internen Tabelle übergeben werden. Diese Tabelle muss einen geeigneten Aufbau besitzen. Sie besteht aus einer oder zwei Spalten des Typs Char. In der ersten Spalte stehen die Namen der zu übertragenden Objekte. Die zweite Spalte nimmt einen alternativen Namen auf, der automatisch von der `FROM`- bzw. `TO`-Klausel als Parameter benutzt wird. Fehlt die zweite Spalte der internen Tabelle oder ist sie leer, wird die `FROM`- bzw. `TO`-Klausel nicht wirksam.

```
EXPORT (itab) |
  { [newname = field] |
   (field | itab  [FROM source]) }
  TO SHARED BUFFER table(area)
  [FROM record] [CLIENT client] ID key [FROM record2]
  [COMPRESSION (ON | OFF)].
```

Felder oder interne Tabellen werden in einen transaktionsübergreifenden Puffer geschrieben, der auf dem Applikationsserver verwaltet wird. Die Daten stehen daher nur für Prozesse zur Verfügung, die auf diesem Server laufen. Die Parameter entsprechen denen des `EXPORT TO DATABASE`-Kommandos.

```
EXPORT (itab) |
  { [newname = field] |
   (field | itab  [FROM source]) }
  TO SHARED MEMORY table(area)
  [FROM record] [CLIENT client] ID key [FROM record2]
  [COMPRESSION (ON | OFF)].
```

Felder oder interne Tabellen werden in den Shared Memory geschrieben. Die Parameter entsprechen denen des `EXPORT TO DATABASE`-Kommandos.

```
EXTRACT fieldgroup.
```

Die aktuellen Inhalte der in `fieldgroup` definierten Felder werden in den Extract-Datenbestand eines Programms geschrieben. Falls eine Feldgruppe `HEADER` deklariert wurde, werden deren Feldinhalte als Schlüssel vorangestellt.

```
FETCH NEXT CURSOR cursor INTO record.
```

Lesen des nächsten Datensatzes aus einem mit `OPEN CURSOR` bestimmten Datenvorrat. Das Ergebnis wird in `record` abgelegt.

```
FIELD-GROUPS fieldgroup.
```

Diese Anweisung deklariert eine so genannte Feldgruppe, die im Zusammenhang mit den Extract-Datenbeständen verwendet werden. Eine Feldgruppe ist eine logische Zusammenfassung mehrerer existierender Felder zu einem Objekt, die während der Laufzeit stattfindet, also dynamischen Charakter hat. Feldgruppen unterscheiden sich damit erheblich von Feldleisten.

```
FIELD-SYMBOLS <fieldsymbol>
  [STRUCTURE table DEFAULT record]|
  [TYPE type]|
  [TYPE REF TO DATA]|
  [TYPE REF TO class]|
  [TYPE REF TO type]|
  [TYPE REF TO object]|
  [TYPE LINE OF type]|
  [LIKE field]|
  [LIKE LINE OF record]|
  [TYPE tablekind].
```

Dieses Kommando deklariert ein Feldsymbol. Die spitzen Klammern, in denen der Name des Feldsymbols steht, gehören zur Syntax des Kommandos. Sie sind obligatorisch. Ein Feldsymbol ist ein Zeiger, der später mit dem Kommando `ASSIGN` einem Datenfeld zugewiesen wird. Ohne Verwendung eines der Zusätze wird der Typ des Feldsymbols erst während der Zuweisung (siehe `ASSIGN`) festgelegt. Mit dem Zusatz `STRUCTURE` wird ein Feldsymbol erzeugt, das auf eine komplette im Data-Dictionary definierte Struktur zeigen kann. Auf die einzelnen Felder der Struktur kann später mit `<fieldsymbol>-fieldname` zugegriffen werden. Für derartige Feldsymbole muss ein Arbeitsbereich `record` bereitgestellt werden, dessen Größe der Größe der Dictionary-Struktur entsprechen muss. Die anderen Zusätze legen einen Typ für das Feldsymbol fest. Dieser Typ wird bei einem `ASSIGN` überprüft.

```
FIND pattern IN [SECTION OFFSET off LENGTH len OF] string
  [[ IGNORING CASE | RESPECTING CASE ] [ IN CHARACTER MODE]] |
  [ IN BYTE MODE ]
  [ MATCH OFFSET offset ]
  [ MATCH LENGTH length ].
```

Mit dem Kommando `FIND` wird in einer Zeichenkette `string` ein Teilstring `pattern` gesucht. Mit den beiden Angaben nach `SECTION` kann der Suchbereich eingegrenzt werden. Standardmäßig wird dabei zwischen Groß- und Kleinschreibung unterschieden und im Zeichen-Mode gesucht (Optionen `RESPEC-`

TING CASE und IN CHARACTER MODE). Die entsprechenden Optionen können daher weggelassen werden. Der Zusatz IGNORING CASE bewirkt das Ignorieren von Groß- und Kleinschreibung beim Suchen, während der Zusatz IN BYTE MODE voraussetzt, dass die beteiligten Felder vom Typ X oder XSTRING sind (hexadezimale Darstellung). Falls das Suchmuster in der zu untersuchenden Zeichenkette enthalten ist, stellt der MATCH OFFSET-Zusatz die Position des ersten Auftretens des Suchbegriffes in das Feld offset. Die Option MATCH LENGTH hingegen füllt das angegebene Zielfeld mit der Länge des Suchmusters.

```
FORM subroutine
  [TABLES   { itab [typing] } ]
  [USING    { [parameter | VALUE(parameter)] [typing] }]
  [CHANGING { [parameter | VALUE(parameter)] [typing] }]
  [RAISING {exception}].
```

Diese Anweisung definiert ein Unterprogramm. Es endet mit der Anweisung ENDFORM. Dem Unterprogramm können sowohl interne Tabellen als auch Felder und Feldleisten als Parameter übergeben werden. Die Zuordnung der aktuellen zu den formalen Parametern erfolgt entsprechend der Reihenfolge der Parameter beim Aufruf des Unterprogramms. Zur Unterscheidung der Parameter ist einer der Zusätze TABLES für interne Tabellen sowie USING und CHANGING für Feldparameter erforderlich. Parameter werden prinzipiell als Referenzparameter behandelt, erst der Zusatz VALUE erzwingt die Behandlung als Wertparameter. Innerhalb der objektorientierten Ergänzung von ABAP gibt der Zusatz RAISING an, welche klassenbasierten Ausnahmen auftreten können. Alle Parameter können mit zusätzlichen Typangaben versehen werden. Zur Laufzeit wird geprüft, ob der aktuelle Parameter den erforderlichen Typ besitzt. Aus Platzgründen wurden die möglichen Anweisungen zur Typisierung in Tabelle 9.5 aufgenommen.

```
FORMAT
  [COLOR [color | OFF]]
  [FRAMES [ON]  | [OFF]]
  [INTENSIFIED [ON] | [OFF]]
  [INVERSE    [ON] | [OFF]]
  [INPUT      [ON] | [OFF]]
  [HOTSPOT    [ON] | [OFF]]
  [RESET].
```

Setzen bzw. Rücksetzen der Ausgabeparameter. Der Schalter ON ist der Standardwert, er muss im Gegensatz zu OFF nicht notiert werden. Die neuen Einstellungen sind ab der nächsten WRITE-Anweisung gültig. Beim Eintritt in neue Verarbeitungszeitpunkte werden einige der Einstellungen wieder auf einen Standardwert gesetzt.

Anweisung	Wirkung	Für einfache Parameter	Für interne Tabellen
`... STRUCTURE record`	Der Parameter entspricht dem Aufbau von `record`.	Ja	Ja
`... TYPE typ`	Der Parameter hat den angegebenen Typ.	Ja	Ja
`... TYPE LINE OF itab`	Der Parameter hat die Struktur eines Datensatzes der kopfzeilenlosen Tabelle `itab`.	Ja	Nein
`...TYPE REF TO class`	Der Parameter ist eine Referenz auf ein Objekt der angegebenen Klasse.	Ja	Nein
`...TYPE REF TO DATA`	Der Parameter ist eine Datenreferenz.	Ja	Nein
`... LIKE element`	Der Parameter hat die Eigenschaften und den Typ von `element`.	Ja	Nein
`... LIKE LINE OF itab`	Der Parameter hat die Struktur der internen Tabelle `itab`.	Ja	Nein
`...TYPE [ C | N | X | P] ...`	Der Parameter hat den angegebenen Typ. Länge und Dezimalstellen bleiben unberücksichtigt.	Ja	Nein
`... TYPE TABLE`	Der aktuelle Parameter muss eine Tabelle ohne Kopfzeile sein.	Ja	Ja
`... TYPE ANY`	Jeder Typ ist zugelassen.		
`... TYPE [ ANY | INDEX | STANDARD| SORTED | HASHED ] TABLE`	Der Parameter ist eine Tabelle des angegebenen Tabellentyps.	Nein	Ja

Tabelle 9.5
Mögliche Zusätze zur Typisierung von Parametern

```
FREE dataobject.
```

Der zur Bearbeitung von `dataobject` benötigte Speicherplatz wird freigegeben. Dabei kann `dataobject` sowohl ein Feld, eine Feldleiste, eine interne Tabelle als auch ein komplexes Datenobjekt sein.

```
FREE MEMORY [ID key].
```

Der globale Speicherbereich wird gelöscht. Die Grundform ohne Zusatz löscht alles, also auch die mit einem Schlüssel versehenen Daten. Das gezielte Löschen von einzelnen Datengruppen erfordert die Angabe des Schlüssels.

```
FREE OBJECT object [NO FLUSH].
```

Das OLE-Objekt `object` wird freigegeben.

```
FUNCTION function.
```

Einleitung der Definition eines Funktionsbausteins.

```
FUNCTION-POOL.
```

Diese Anweisung leitet das Rahmenprogramm einer Funktionsgruppe ein. Diese Anweisung wird beim Anlegen einer neuen Funktionsgruppe vom System automatisch erzeugt. Sie ist äquivalent zu einer `REPORT`-Anweisung.

```
GET node
  [LATE]
  [FIELDS { field } ].
```

Zeitpunktanweisung. Diese Anweisung ist nur in Reports nutzbar, die mit einer logischen Datenbank arbeiten. Der Parameter *node* kann entweder eine Datenbanktabelle oder einen logischen Knoten bezeichnen. Der nächste Datensatz aus `node` wird in der gleichnamigen Kopfzeile bereitgestellt. Gleichzeitig werden die Inhalte aller Tabellen abgerufen, die in der Hierarchie über der Tabelle stehen. Der Zusatz `LATE` bewirkt die Verarbeitung, nachdem auch alle untergeordneten Tabellen gelesen wurden. Mit `FIELDS` kann der Zugriff auf die angegebenen Felder beschränkt werden, was zu Performanceverbesserungen führen kann, da durch die logische Datenbank nur noch die ausgewählten Felder mit Daten versorgt werden müssen.

```
GET BIT n OF field INTO result.
```

Bei `field` muss es sich um ein hexadezimales Feld (Typ X oder XSTRING) handeln. Die Bits in diesem Feld werden von links mit 1 beginnend nummeriert. Das Kommando schreibt das Bit `n` aus `field` in das Feld `result`.

```
GET CURSOR FIELD field
  [OFFSET offset]
  [LINE line]
  [VALUE value]
  [LENGTH length].
  [AREA area].
```

Dieses Kommando liefert den Namen des Feldes, auf dem der Cursor gerade steht. Die Zusätze geben darüber hinaus den Offset des Cursors innerhalb des Feldes, die Nummer einer eventuellen Step-Loop-Zeile bzw. die absolute List-Zeile, den aktuellen Wert, die Ausgabelänge des Feldes oder den Namen eines Controls an. Dieses Kommando kann sowohl in der Dialogverarbeitung als auch im interaktiven Reporting benutzt werden.

```
GET CURSOR LINE line
  [OFFSET offset]
  [VALUE value]
  [LENGTH length].
```

Dieses Kommando liefert die Nummer der List-Zeile oder der Step-Loop-Zeile, in der sich der Cursor befindet. Die Zusätze sind nur innerhalb der List-Verarbeitung wirksam. Sie liefern den Offset des Cursors in der Zeile, den Inhalt oder die Länge der gesamten Zeile.

```
GET DATASET dataset
  [POSITION pos]
  [ATTRIBUTES attrib].
```

Mit diesem Kommando können Eigenschaften einer Datei ermittelt werden. Ohne die Zusätze wird lediglich geprüft, ob die Datei geöffnet ist. Die Zusätze dienen der Ermittlung der aktuellen Schreibposition und der Datei sowie einer Reihe von Attributen.

```
GET LOCAL LANGUAGE lang COUNTRY country MODIFIER mod.
```

Ermitteln der aktuellen Textumgebung und Ablege der gelesenen Werte in die Felder `lang`, `country` und `mod`.

```
GET PARAMETER ID parameter_name FIELD field.
```

Der Parameter mit dem angegebenen Namen wird aus dem globalen Speicher gelesen und in das angegebene Feld gestellt. Der Parametername kann als Feldinhalt oder Direktwert angegeben werden.

```
GET PF-STATUS status [PROGRAM prog] [EXCLUDING itab].
```

Der Name des aktuellen Status wird gelesen und in `status` abgelegt. Falls der aktuelle Status zu einem anderen Programm gehört, kann dessen Name durch den Zusatz `PROGRAM` ermittelt werden. Die momentan deaktivierten Funktionscodes können durch den Zusatz `EXCLUDING` in eine interne Tabelle geschrieben werden.

```
GET PROPERTY OF object attribute = field [NO FLUSH] [QUEUE-
ONLY].
```

Es wird eine Eigenschaft eines OLE-Objekts gelesen und in das angegebene Feld gestellt. Der Zusatz `NO FLUSH` bewirkt das Sammeln mehrerer OLE-Anforde-

rungen mit dem Ziel der gemeinsamen Ausführung. Mit QUEUE-ONLY wird unter bestimmten Bedingungen das Zurückschreiben des Rückgabewerts unterdrückt.

```
GET REFERENCE OF field INTO ref.
```

Die Referenz ref wird mit einer Referenz auf das Feld field gefüllt.

```
GET RUN TIME FIELD field.
```

Der erste Aufruf von GET RUN TIME initialisiert field. Jeder weitere Aufruf füllt field mit der seit dem ersten Aufruf verstrichenen Laufzeit der Anwendung. Die Einheit sind Mikrosekunden.

```
GET TIME [FIELD field].
```

Das Feld SY-UZEIT wird mit der aktuellen Uhrzeit gefüllt. Außerdem wird SY-DATUM neu gesetzt. Falls der Zusatz FIELD verwendet wird, entfällt die Aktualisierung der Systemfelder. Stattdessen wird lediglich die aktuelle Zeit in field geschrieben.

```
GET TIME STAMP FIELD field.
```

Liefert einen Zeitstempel.

```
HIDE data.
```

Der Inhalt von data wird in einen verborgenen Speicherbereich gestellt und mit der Nummer der aktuellen Zeile verknüpft. Der verborgene Inhalt wird durch Zeilenselektionen im interaktiven Reporting wieder reaktiviert. Bei data kann es sich sowohl um ein einzelnes Feld, eine Aufzählung von Feldern als auch um eine Feldleiste handeln.

```
IF condition.
  ABAP-statements ...
[ELSE. | ELSEIF condition.]
  ABAP-statements ...
ENDIF.
```

Der logische Ausdruck wird ausgewertet. Ist er erfüllt, werden die unmittelbar folgenden Anweisungen bis zum nächsten ELSE, ELSEIF oder ENDIF ausgeführt. Ist er nicht erfüllt, wird der nächste ELSEIF-Ausdruck ausgewertet bzw. nach einer ELSE-Anweisung fortgesetzt.

```
IMPORT (itab) | [field | itab] [TO target | =target] FROM
DATA BUFFER buffer.
```

Mit dieser Anweisung werden Objekte aus dem als Puffer benutzten String gelesen. Sie müssen mit EXPORT TO DATA BUFFER dort abgelegt worden sein. Mittels des Zusatzes TO bzw. = können Werte beim Import in andere Felder abgelegt werden.

```
IMPORT (itab) | [field | itab] [TO target | =target]
  FROM DATABASE table(area) TO record
    ID key [CLIENT client] |
    MAJOR-ID maid MINOR-ID miid.
```

Es werden so genannte Datencluster, also Gruppen von Datenfeldern oder internen Tabellen aus einem Gebiet einer speziellen Tabelle importiert. Diese Werte müssen mit `EXPORT TO DATABASE` exportiert worden sein. Die gelesenen Daten werden entweder komplett in einer internen Tabelle oder einzeln abgelegt. Mit dem Zusatz `TO`, der für jeden Einzelparameter separat angegeben werden muss, können gelesene Werte in anderen Feldern abgelegt werden. Der Schlüssel kann entweder komplett nach `ID` oder in zwei Teile aufgespalten und mit `MAJOR-ID` und `MINOR-ID` angegeben werden. Falls die Tabelle mandantenabhängig ist, kann auch die Mandantennummer mit `CLIENT` übergeben werden. In der Anweisung sind `table` und `area` Direktwerte, die runden Klammern um `area` gehören zur Syntax des Kommandos und kennzeichnen hier ausnahmsweise keinen Verweis auf einen Feldinhalt. Die Anweisung `TO record` nach der Tabellen- und Gebietsangabe ermöglicht die Ablage der Steuerdaten in einem separaten Arbeitsbereich.

Eine Vielzahl weiterer Optionen erlaubt die Anpassung des Kommandos an aktuelle Gegebenheiten. Dies betrifft vor allem das Anhängen von Feldern und das Konvertieren von Zeichensätzen.

```
IMPORT DIRECTORY INTO itab
  FROM DATABASE table(area) [CLIENT client] ID key.
```

Diese Anweisung liefert ein Verzeichnis der Objekte, die in der angegebenen Tabelle im jeweiligen Gebiet mit einem bestimmten Schlüssel abgelegt wurden. Die Tabelle `itab` muss einen vorgegebenen Aufbau besitzen.

```
IMPORT (itab) | [field | itab] [TO target | =target] FROM
MEMORY [ID key].
```

Mit dieser Anweisung werden Objekte aus dem globalen Speicher gelesen, die mit `EXPORT` dort abgelegt wurden. Mittels des Zusatzes `TO` können Werte in anderen Feldern abgelegt werden.

```
IMPORT (itab) | [field | itab] [TO target | =target]
  FROM SHARED BUFFER table(area) TO record
    ID key [CLIENT client].
```

Importieren von Daten aus dem transaktionsübergreifenden Puffer. Die Parameter entsprechen denen von `IMPORT DATABASE`.

```
IMPORT (itab) | [field | itab] [TO target | =target]
  FROM SHARED MEMORY table(area) TO record
    ID key [CLIENT client].
```

Importieren von Daten aus dem Shared Memory.

```
INCLUDE file.
```

Die Datei `file` enthält ABAP-Quelltext. Diese Anweisungen werden in das aktuelle Programm einbezogen. Das Programm verhält sich so, als wären die Anweisungen direkt im eigenen Programm notiert. Eine Include-Datei kann von mehreren Programmen eingebunden werden.

```
INCLUDE STRUCTURE record
  [AS name [RENAMING WITH SUFFIX suffix]].
```

Diese Anweisung ist nur innerhalb einer `BEGIN OF REC ... END OF REC`-Anweisung sinnvoll. Sie fügt die Struktur der angegebenen Feldleiste in die Deklaration ein. Dabei wird die Struktur aufgelöst, eingefügt wird also nicht die Feldleiste als Ganzes, sondern nur die in ihr enthaltenen Felder. Durch Angabe eines Alias-Namens für die eingefügte Struktur kann auf sie aber weiterhin als eigenständiges Element zugegriffen werden. Eine Struktur kann auch mehrfach eingebunden werden. Dabei sorgt der Suffix, der an alle Feldnamen der Substruktur angehängt wird, für eindeutige Feldbezeichner.

```
INCLUDE TYPE record_typ
  [AS name [RENAMING WITH SUFFIX suffix]].
```

Diese Anweisung ist nur innerhalb einer `BEGIN OF REC ... END OF REC`-Anweisung sinnvoll. Sie fügt die Struktur des angegebenen Feldleistentyps in die Deklaration ein, wobei die Struktur in ihre Einzelfelder aufgelöst wird. Trotz der Auflösung der Struktur kann durch die Vergabe eines Alias-Namens weiterhin auf die Substruktur als Ganzes zugegriffen werden. Ein Suffix, der an Feldnamen der Substruktur angehängt wird, verhindert Konflikte durch identische Feldnamen verschiedener Substrukturen.

```
INFOTYPES nnnn
  [NAME name]
  [OCCURS roll_area]
  [MODE mode]
  [VALID FROM first TO last].
```

Es wird ein so genannter Infotyp definiert. Ein Infotyp ist, vereinfacht beschrieben, eine interne Tabelle, die mit einer Datenbanktabelle korrespondiert. Infotypen werden nur selten in speziellen Anwendungen benutzt.

```
INITIALIZATION.
```

Dies ist eine Zeitpunktanweisung im Reporting. Der angegebene Zeitpunkt wird einmal zu Beginn eines Reports unmittelbar nach Erstellung der Parameter prozessiert.

```
INSERT table | *table | (field_with_tablename)
  [CLIENT SPECIFIED]
  [FROM record].
```

Diese Anweisung fügt einen Datensatz in eine Datenbanktabelle ein. Der Name der Tabelle kann als Direktwert über ein Feld mit entsprechendem Inhalt übergeben werden. Die Daten werden entweder der Kopfzeile der Tabelle oder aus dem Arbeitsbereich `record` entnommen, der unstrukturiert sein kann. Auch bei dieser Variante des Kommandos muss der Datensatz einen eindeutigen Schlüssel bezüglich der Tabelle und aller `UNIQUE`-Indizes besitzen. Der Zusatz `CLIENT SPECIFIED` schaltet das automatische Mandantenhandling ab.

Die Angabe eines Arbeitsbereichs ist im Moment noch optional. Allerdings sollte für Neuentwicklungen immer ein Arbeitsbereich (Zusatz `FROM`) angegeben werden, da die Variante ohne Arbeitsbereich als veraltet gekennzeichnet wurde.

```
INSERT table | (field_with_tablename) [CLIENT SPECIFIED]
  FROM TABLE itab [ACCEPTING DUPPLICATE KEYS].
```

Mit dieser Open-SQL-Anweisung werden mehrere Datensätze in eine Datenbanktabelle eingefügt. Der Name der Tabelle kann als Direktwert über ein Feld mit entsprechendem Inhalt übergeben werden. Die einzufügenden Datensätze werden der internen Tabelle `itab` entnommen. Der Aufbau dieser Tabelle muss denselben Ansprüchen genügen wie der Arbeitsbereich der beiden vorangegangenen Varianten. Der `ACCEPTING`-Zusatz verhindert einen Laufzeitfehler beim Versuch, Datensätze mit identischem Schlüssel einzufügen. Mit `CLIENT SPECIFIED` kann wiederum das automatische Mandantenhandling abgeschaltet werden.

```
INSERT [[record | INITIAL LINE] INTO] itab
  [INDEX index]
  [ASSIGNING <fieldsymbol> |
   REFERENCE INTO reference ].
```

Dieses Kommando fügt einen Datensatz in eine interne Tabelle des Typs Standard-Tabelle ein. Die Daten werden einem explizit angegebenen Arbeitsbereich oder der Kopfzeile der Tabelle entnommen. Bei Bedarf kann mit `INITIAL LINE` auch ein mit Initialwerten gefüllter Datensatz angefügt werden. Die Position, an der eingefügt wird, ist mittels des Zusatzes `INDEX` anzugeben. Der erste Datensatz hat den Index 1. Innerhalb einer `LOOP`-Schleife über die Tabelle kann auf die Angabe des Index verzichtet werden. Der neue Datensatz wird dann vor dem gerade aktuellen Datensatz eingefügt.

Die beiden Zusätze `ASSIGNING` und `REFERENCE` füllen bei erfolgreicher Ausführung des Kommandos das Feldsymbol oder die Referenz mit einem Verweis auf die gerade angefügte Zeile.

Diese Variante des `INSERT`-Kommandos funktioniert nur für Standard- und sortierte Tabellen.

```
INSERT [[record | INITIAL LINE] INTO] TABLE itab
  [ASSIGNING <fieldsymbol> |
   REFERENCE INTO reference ].
```

Dieses Kommando fügt ebenfalls einen Datensatz in eine interne Tabelle ein. Allerdings arbeitet dieses Kommando generisch, kann also für alle Arten interner Tabellen eingesetzt werden. Die Tabellenart der zu bearbeitenden Tabelle bestimmt, wie der neue Datensatz einsortiert wird. Die Wirkung der Zusätze ASSIGNING und REFERENCE entspricht dem vorangegangenen Kommando.

```
INSERT LINES OF itab_1
  [FROM first] [TO last]
  INTO itab2 [INDEX position].
```

Datensätze aus der internen Tabelle itab1 werden in die interne Tabelle itab2 eingefügt. Durch Angabe zusätzlicher Parameter können die zu übertragenden Datensätze sowie die Position, an der eingefügt werden soll, näher bestimmt werden. Die Tabelle itab2 muss vom Typ Standard-Tabelle sein.

```
INSERT LINES OF itab_1
  [FROM first] [TO last]
  INTO TABLE itab2.
```

Generisches Einfügen von Datensätzen aus itab1 in itab2. Dabei ist die Tabellenart von itab2 beliebig.

```
INSERT { field_x } INTO fieldgroup.
```

Mit dieser Anweisung werden Datenfelder einer vorher mit FIELD-GROUPS deklarierten Feldgruppe hinzugefügt. Mit diesem Vorgang ist kein Datentransport verbunden, es wird lediglich bestimmt, welche Datenfelder zur Feldgruppe gehören sollen. Obwohl diese Anweisung laut SAP-Dokumentation eine operationale und keine deklarative Anweisung ist, erfüllt sie also eine mit einer Deklaration vergleichbare Aufgabe.

```
INSERT INTO table | (field_with_tablename)
  [CLIENT SPECIFIED]
  VALUES record.
```

Dies ist eine Open-SQL-Anweisung. Mit dieser Anweisung wird ein Datensatz in eine Datenbanktabelle eingefügt. Die Daten werden gemäß der Struktur der Tabelle aus record entnommen. Der Arbeitsbereich selbst kann unstrukturiert sein, muss dann aber alle Daten in der richtigen Reihenfolge und Länge enthalten. Der Name der Tabelle kann als Direktwert über ein Feld mit entsprechendem Inhalt übergeben werden. Der Datensatz wird nur eingefügt, wenn weder in der Tabelle noch in irgendeinem UNIQUE-Index ein Datensatz mit identischem Schlüssel existiert. Mit CLIENT SPECIFIED kann das automatische Mandantenhandling abgeschaltet werden. Das Feld MANDT muss dann wie jedes andere Schlüsselfeld im Programm mit Daten versorgt werden.

```
INSERT REPORT program FROM itab.
```

Einfügen eines Programms in den globalen Programmbestand. Der Quelltext wird aus der internen Tabelle itab gelesen.

```
INSERT TEXTPOOL program FROM itab LANGUAGE language.
```

Einfügen von Textelementen für das Programm in der angegebenen Sprache in den Programmbestand. Die Texte werden einer internen Tabelle entnommen.

```
INTERFACE interface
  [SUPPRTING REMORE INVOCATION]
  [DEFERRED]
  [LOAD].
```

Definition eines Interface innerhalb von ABAP Objects. Ein Interface ist eine Beschreibung einer Schnittstelle ohne ausprogrammierte Methoden. Die Definition wird durch die Anweisung `ENDINTERFACE` abgeschlossen.

Der Zusatz `SUPPORTING REMOTE INVOCATION` ermöglicht den Remote-Aufruf der Methoden des Interface.

Die beiden anderen Zusätze liefern dem Laufzeitsystem lediglich eine steuernde Information. Wenn diese Zusätze benutzt werden, handelt es sich nicht um eine echte Definition. Die Anweisung steht dann für sich allein und leitet keinen durch `ENDINTERFACE` abgeschlossenen Block ein.

Durch den Zusatz `DEFERRED` wird nur der Name des Interface eingeführt, die Definition erfolgt später. Mit `LOAD` wird die Definition eines global existierenden Interface geladen.

```
INTERFACE-POOL.
```

Diese Anweisung leitet einen Interface-Pool ein, in dem der Class Builder globale Interfaces ablegt. Diese Anweisung wird nie manuell notiert, sondern durch den Class Builder generiert.

```
INTERFACES interface
  ABSTRACT METHODS { method }
  FINAL METHODS { method }
  ALL METHODS ABSTRACT
  ALL METHODS FINAL
  DATA VALUES { attribute = value }.
```

Einfügen eines Interface in eine Klasse oder in ein anderes Interface. Die Zusätze ermöglichen es, ausgewählte oder alle Methoden mit den Eigenschaften `ABSTRACT` oder `FINAL` zu versehen. Abstrakte Methoden müssen überschrieben werden, während finale Methoden nicht mehr überschrieben werden dürfen. Der `DATA VALUES`-Zusatz ermöglicht es, Attributen des Interface Anfangswerte zuzuordnen.

```
LEAVE.
```

Mit dieser Anweisung wird eine mit `CALL` aufgerufene Anwendung (Transaktion, Dialog, Report) beendet und die rufende Anwendung mit der Anweisung nach dem `CALL`-Aufruf fortgesetzt.

```
LEAVE LIST-PROCESSING.
```

Die in einen Dialog eingebettete und mit `LEAVE TO LIST-PROCESSING` gestartete Listenverarbeitung wird beendet.

```
LEAVE PROGRAM.
```

Das aktuelle Programm wird beendet.

```
LEAVE SCREEN.
```

Die Bearbeitung des aktuellen Dynpros wird beendet und das in den Dynpro-Attributen eingetragene oder vorher mit `SET SCREEN` gesetzte Dynpro ausgeführt.

```
LEAVE TO LIST-PROCESSING [AND RETURN TO SCREEN
screen_number].
```

Innerhalb einer Dialoganwendung wird vorübergehend die List-Verarbeitung aktiviert. Nach Beenden der List-Verarbeitung wird der PBO-Teil des aktuellen Dynpros oder aber der PBO-Teil des mit `RETURN TO SCREEN` angegebenen Dynpros ausgeführt.

```
LEAVE TO SCREEN screen_number.
```

Die Bearbeitung des aktuellen Dynpros wird beendet und dafür das Dynpro mit der angegebenen Nummer ausgeführt.

```
LEAVE TO TRANSACTION transaction_code
  [AND SKIP FIRST SCREEN].
```

Die aktuelle Transaktion wird beendet und dafür eine neue Transaktion gestartet. Der `SKIP`-Zusatz sorgt dafür, dass das erste Dynpro der gerufenen Transaktion nicht auf dem Bildschirm prozessiert wird, falls alle Eingabefelder dieses Dynpros über geeignete Mechanismen mit gültigen Werten versorgt werden.

```
LOAD-OF-PROGRAM.
```

Zeitpunktanweisung. Dieser Zeitpunkt wird immer dann prozessiert, wenn ein Programm vom Typ 1 (Online-Report), M (Modul-Pool), F (Funktionsgruppe) und S (Unterprogramm-Pool) in einen internen Modus geladen wird.

```
LOCAL field.
```

Diese Anweisung ist nur innerhalb von Unterprogrammen, also nach einer `FORM`-Anweisung möglich. Sie rettet den augenblicklichen Inhalt von `field` und stellt ihn nach Verlassen des Unterprogramms wieder her. Innerhalb des Unterprogramms kann das globale Feld daher wie ein lokales Datenfeld behandelt werden. Zuweisungen zu diesem Feld sind möglich, bleiben aber außerhalb des Unterprogramms ohne Wirkung. Innerhalb von ABAP Objects kann diese Anweisung nicht mehr eingesetzt werden.

```
LOOP.
  ABAP-statements ...
ENDLOOP.
```

Schleife über alle Datensätze des EXTRACT-Datenbestands einer Anwendung. Die Daten werden in den Feldern der jeweiligen Feldgruppen bereitgestellt.

```
LOOP AT itab
  [INTO record]
  [FROM first]
  [TO last]
  [WHERE condition]
  [ASSIGNING <field symbol>]
  [REFERENCE INTO reference]
  [TRANSPORTING NO FIELDS].
    ABAP-statements
ENDLOOP.
```

Schleife über eine interne Tabelle. Die Datensätze werden entweder in der Kopfzeile der Tabelle oder in dem mit INTO angegebenen Arbeitsbereich bereitgestellt oder aber es wird ein Feldsymbol auf den entsprechenden Datensatz innerhalb des Speicherbereichs der Tabelle gesetzt. In diesem Fall entfällt das Kopieren von Daten, Modifikationen der Daten finden dann aber sofort in der Tabelle und nicht in einem separaten Arbeitsbereich statt. Die Zusätze FROM bzw. TO beschränken die Schleifendurchläufe auf die entsprechenden Datensätze. Die Zählung beginnt mit 1. Mittels WHERE ist eine Auswahl der in der Schleife zu bearbeitenden Datensätze über eine logische Bedingung möglich. Der Zusatz TRANSPORTING NO FIELDS verhindert den Transport der Daten aus dem selektierten Datensatz in den Arbeitsbereich.

Die Zusätze ASSIGNING bzw. REFERENCE setzen einen Zeiger auf den jeweiligen Datensatz und vermeiden somit die Kopie der Daten. Über die Zeiger wird somit direkt auf den Datensatz zugegriffen.

```
LOOP AT SCREEN [ INTO record ].
  ABAP-statements ...
ENDLOOP.
```

Diese Anweisung ist nur im PBO-Modul eines Dynpros sinnvoll. Sie stellt nacheinander die Beschreibung der Felder des aktuellen Dynpros in der vordefinierten Kopfzeile SCREEN oder dem optional anzugebenden Record zur Verfügung. Dabei kann SCREEN wie der Kopfsatz einer internen Tabelle bearbeitet werden, Änderungen der Kopfzeile werden mit MODIFY in die interne Tabelle SCREEN geschrieben und werden bei der Ausgabe des Dynpros wirksam. Der Aufbau der Struktur ist vom Releasestand abhängig.

```
MESSAGE tnnn[(class)]
  [WITH parameter_1 ... parameter_4]
  [RAISING exception]
```

```
  [INTO field]
  [DISPLAY LIKE type].
```

Die Nachricht nnn mit dem Nachrichtentyp t wird ausgegeben. Im Nachrichtentext können bis zu vier mit dem Zeichen & dargestellte Platzhalter durch die Parameter parameter_1 bis parameter_4 ersetzt werden. Die Nachrichtenklasse wird normalerweise in der REPORT- oder PROGRAM-Anweisung festgelegt. Falls von dieser Vorgabe abgewichen werden soll, muss die Nachrichtenklasse in runden Klammern hinter der Nachrichtennummer stehen. Innerhalb von Funktionsbausteinen löst der Zusatz RAISING eine Ausnahme aus. Die Nachricht wird in diesem Fall nur angezeigt, wenn die Ausnahme vom rufenden Programm nicht behandelt wird. Das Aussenden einer Nachricht unterbleibt ebenfalls beim Zusatz INTO. Dieser Zusatz bewirkt, dass der Nachrichtentext mit eingefügten Parametern in das Feld geschrieben wird. Der Zusatz DISPLAY fügt ein zum Nachrichtentyp passendes Symbol in den Nachrichtentext ein.

```
MESSAGE ID class TYPE typ NUMBER number
  [WITH parameter_1 ... parameter_4]
  [RAISING exception]
  [INTO field]
  [DISPLAY LIKE type].
```

Diese Anweisung entspricht der vorangegangenen, allerdings werden die drei Werte zur Identifikation der Nachricht (Klasse, Typ und Nummer) einzeln angegeben. Sie dürfen bei dieser Form der Anweisung auch dynamisch gesetzt werden.

```
MESSAGE message TYPE type
  [RAISING exception]
  [DISPLAY LIKE type].
```

Bei dieser Variante des Message-Kommandos wird die Nachricht direkt als Zeichenkette angegeben.

```
METHOD method.
```

Implementierung einer Methode innerhalb einer Klasse in ABAP Objects.

```
METHODS method
  [ IMPORTING { [[VALUE | REFERENCE](parameter)] | parameter
      [ TYPE type | LIKE field ]
      [ OPTIONAL  | DEFAULT default ] }
      PREFERRED PARAMETER param
  ]
  [ EXPORTING { [[VALUE | REFERENCE](parameter)] | parameter
      [ TYPE type | LIKE field]
      [ OPTIONAL  | DEFAULT default ] } ]
  [ CHANGING  { [[VALUE | REFERENCE](parameter)] | parameter
      [ TYPE type | LIKE field ]
```

```
      [ OPTIONAL | DEFAULT default ] } ]
  [ RETURNING VALUE(parameter)
      [ TYPE type | LIKE field ] ]
  [ EXCEPTIONS { exception } ]
  [ RAISING { exception } ]
  [ FOR EVENT event OF class]
  [ ABSTRACT ]
  [ FINAL ]
  [ REDEFINITION ].
```

Deklaration einer Methode innerhalb einer Klasse oder eines Interface in ABAP Objects. In der Deklaration wird nur die Schnittstelle der Methode bestimmt. Durch die Parameter `IMPORTING`, `EXPORTING` und `CHANGING` werden Übergabeparameter bestimmt. Der Zusatz `RETURNING` kennzeichnet einen einzelnen Rückgabewert, der vom rufenden Programm auf spezielle Weise ausgewertet werden kann. Durch `EXCEPTIONS` kann die Methode herkömmliche Ausnahmen an das rufende Programm melden, für klassenbasierte Ausnahmen wird `RAISING` benutzt. Mit `ABSTRACT` wird die Methode als abstrakte, also nicht funktionsfähige Methode gekennzeichnet. Derartige Methoden müssen überschrieben werden. Der Zusatz `FINAL` verhindert das Überschreiben der Methode, während `REDEFINITION` kennzeichnet, dass mit dieser Methode eine bereits existierende Methode überschrieben werden soll. In diesem Fall können keine weiteren Schnittstellenparameter angegeben werden, da beim Überschreiben die Schnittstelle nicht geändert werden kann.

Durch den Zusatz `FOR EVENT` wird eine Methode deklariert, die ein Ereignis behandeln soll.

```
MODIFY table | *table | (field_with_tablename)
  [FROM record]
  [CLIENT SPECIFIED].
```

Modifizieren bzw. Einfügen eines Datensatzes in einer Datenbanktabelle. Der Tabellenname kann statisch oder dynamisch angegeben werden. Die Daten werden aus der Kopfzeile der Tabelle oder dem explizit angegebenen Datensatz entnommen. Der Datensatz wird durch den Inhalt der Schlüsselfelder identifiziert. Der Zusatz `CLIENT SPECIFIED` schaltet die automatische Behandlung des Mandantenfeldes ab. Dieses Feld kann bzw. muss dann – ebenso wie alle anderen Felder – mit einem korrekten Wert belegt werden. Damit ist die mandantenübergreifende Bearbeitung von mandantenabhängigen Tabellen möglich. Das Kommando `MODIFY` erkennt selbständig, ob ein vorhandener Datensatz modifiziert oder ein neuer Datensatz angefügt werden muss. Da sich diese Prüfung nachteilig auf die Performance auswirken kann, sollte `MODIFY` nur verwendet werden, wenn im Programm keine eindeutige Entscheidung für `INSERT` oder `UPDATE` möglich ist.

```
MODIFY table | (field_with_tablename) FROM TABLE itab
  [CLIENT SPECIFIED].
```

Dieses Kommando arbeitet wie die vorangegangene Version des MODIFY-Kommandos. Allerdings werden die Daten einer internen Tabelle entnommen, so dass mehrere Datensätze mit einem einzigen Kommando modifiziert werden können.

```
MODIFY table | *table VERSION field_with_tablename.
```

Dieses Kommando ist veraltet. Es gestattet die dynamische Angabe eines Tabellennamens. Seine Arbeitsweise entspricht ansonsten der ersten Form des MODIFY-Kommandos.

```
MODIFY itab [FROM record]
  [ INDEX index ]
  [ ASSIGNING <field symbol> ]
  [ REFERENCE INTO reference ]
  [ TRANSPORTING fieldlist | (field_with_fieldname)
    [ WHERE condition ]].
```

Ändern von Datensätzen einer internen Tabelle des Typs Standard-Tabelle. Das Einfügen von neuen Datensätzen wie beim MODIFY-Kommando für Datenbanktabellen ist nicht möglich! Die Daten werden entweder der Kopfzeile der Tabelle oder einem mit dem Zusatz FROM anzugebenden Arbeitsbereich entnommen. Mit dem Zusatz TRANSPORTING kann eine Auswahl der zu übertragenden Felder erfolgen, alle nicht in der Feldliste aufgeführten Felder werden nicht vom Arbeitsbereich in die Tabelle transportiert. Die Felder können als Konstante oder dynamisch angegeben werden.

Der oder die zu bearbeitenden Datensätze können auf unterschiedliche Weise festgelegt werden. Ohne INDEX- oder WHERE-Zusatz kann das Kommando nur innerhalb einer LOOP-Schleife über eine interne Tabelle benutzt werden. Geändert wird in diesem Fall der gerade aktuelle Datensatz. Außerhalb von LOOP-Schleifen wird ein einzelner Datensatz durch die Angabe der Datensatznummer mit INDEX identifiziert. Dieser Index kann beispielsweise bei einer vorangegangenen READ-Anweisung ermittelt werden. Der WHERE-Zusatz ermöglicht einen Massen-Update der internen Tabelle. In allen Datensätzen der internen Tabelle, die der Bedingung genügen, werden die mit TRANSPORTING benannten Felder auf den aktuellen Wert der entsprechenden Felder des Arbeitsbereiches oder der Kopfzeile gesetzt. Da sich dadurch identische Feldinhalte in mehreren Datensätzen ergeben, ist der Einsatz der TRANSPORTING-Option zwingend erforderlich.

Der ASSIGNING- bzw. REFERENCE-Zusatz setzen nach erfolgreicher Ausführung einen Zeiger auf den jeweiligen Datensatz.

```
MODIFY TABLE itab
  [ FROM record ]
  [ ASSIGNING <field symbol> ]
  [ REFERENCE INTO reference ]
  [ TRANSPORTING fieldlist | (field_with_fieldname) ].
```

Generisches Kommando zum Modifizieren von Datensätzen einer internen Tabelle. Es arbeitet mit allen drei Arten interner Tabellen.

```
MODIFY LINE line
  [INDEX level]
  [LINE FORMAT  { formats } ]
  [LINE VALUE FROM record]
  [FIELD VALUE  { target FROM source } ]
  [FIELD FORMAT { field formats } ].
```

Dieses Kommando ist nur in der Listenverarbeitung sinnvoll. Es modifiziert die Zeile `line` in der aktuellen Liste bzw. der Liste der List-Stufe `level`. Mit dem Zusatz `LINE FORMAT` werden der Zeile ein oder mehrere neue Formate zugewiesen. Die Formate entsprechen denen des Kommandos `FORMAT`. Mit `FIELD VALUE` wird einem Feld, das in die Liste geschrieben wurde, ein neuer Wert zugewiesen, mit `LINE VALUE` ändern Sie die gesamte Zeile. Die Änderungen werden in der Liste sofort wirksam. Mit `FIELD FORMAT` ist das gezielte Zuweisen von Formaten zu einem ausgewählten Feld möglich.

```
MODIFY LINE i ( OF CURRENT PAGE | OF PAGE page )
  [LINE FORMAT  { formats } ]
  [LINE VALUE FROM record]
  [FIELD VALUE  { target FROM source} ]
  [FIELD FORMAT { field formats } ].
```

Dieses Kommando ändert die angegebene Zeile der aktuellen Liste. Dabei erfolgt die Zeilenzählung bezogen auf eine Seite, die mit `OF PAGE` angegeben wird. Mit `OF CURRENT PAGE` anstelle von `OF PAGE` wird die aktuelle Seite gewählt. Zeilen in anderen Listen können nicht geändert werden.

```
MODIFY CURRENT LINE
  [LINE FORMAT  { formate } ]
  [FIELD VALUE  { target FROM source} ]
  [LINE VALUE FROM record]
  [FIELD FORMAT { field formats } ].
```

Die Formate der aktuellen Zeile (das ist die letzte per Zeilenselektion oder `READ LINE` gelesene Zeile) werden geändert. Die Zusätze entsprechen denen der beiden vorangegangenen Varianten.

```
MODIFY SCREEN [ FROM record ].
```

Dieses Kommando schreibt innerhalb der `LOOP AT SCREEN`-Schleife die modifizierten Attribute eines Dynpro-Feldes in die Dynpro-Beschreibung zurück. Dabei werden die Daten entweder dem vordefinierten Datensatz `SCREEN` oder dem explizit angegebenen Datensatz entnommen.

```
MODULE modulname [ INPUT | OUTPUT ].
```

Definition eines Moduls, das in der Ablauflogik eines Dynpros aufgerufen werden kann. Das Modul ist mit ENDMODULE abzuschließen. In der PBO-Phase eines Dynpros werden nur Module gesucht und ausgeführt, die den Zusatz OUTPUT besitzen. In der PAI-Phase sind nur Module ohne Zusatz oder mit dem Zusatz INPUT wirksam.

```
MOVE source[+offset][(length)] | method() TO
     target[+offset][(length)]
     [PERCENTAGE percentage [LEFT | RIGHT]].
```

Der Inhalt des Quellfelds wird in das Zielfeld übertragen. Dabei können sowohl bei der Quelle als auch beim Ziel statische oder dynamische Angaben für einen Offset innerhalb des Feldes und eine Längenangabe stehen. Die Anweisung kann auch komplexe Datenstrukturen und Tabellen kopieren. Als Dezimalzeichen dient immer der Punkt. Bei Verwendung der objektorientierten Erweiterung ist es möglich, den RETURNING-Parameter einer Methode als Quellwert zu verwenden.

Der Zusatz PERCENTAGE erwartet als Quell- und Zielfeld Felder des Typs C. Er überträgt den mit percentage angegebenen Teil von Zeichen aus dem Quellfeld in das Zielfeld. Falls dabei keine Längen- bzw. Offsetangaben notiert werden, wird auf das Quellfeld unabhängig von seinem aktuellen Inhalt in der definierten Länge zugegriffen. Mit den Zusätzen LEFT bzw. RIGHT kann die Ausrichtung der übertragenen Zeichenkette im Zielfeld bestimmt werden. Der Defaultwert ist LEFT, dieser Zusatz muss also nicht notiert werden.

Das oder die Quellfelder können durch das Ergebnis einer Methode geliefert werden.

```
MOVE ref1 ?TO ref2.
```

Zuweisen einer Objekt- oder Datenreferenz zu einer anderen.

```
MOVE-CORRESPONDING source TO target.
```

Diese Anweisung kopiert alle Felder aus der Feldleiste source in die gleichnamigen Felder der Feldleiste target. Felder, die nicht in beiden Feldleisten vorhanden sind, bleiben unberücksichtigt.

```
MULTIPLY field BY factor.
```

Der Inhalt von field wird mit factor multipliziert und das Ergebnis in field abgelegt.

```
MULTIPLY-CORRESPONDING record_1 BY record_2.
```

Dieses Kommando erzeugt für alle Felder, die sowohl in record_1 als auch in record_2 enthalten sind, eine einfache MULTIPLY BY-Anweisung und führt diese aus. Dabei dienen die Felder in record_1 als Quell- und Zielfeld; der Faktor wird record_2 entnommen.

```
NEW-LINE [NO-SCROLLING].
```

Diese Anweisung ist nur in der Listenverarbeitung sinnvoll. Sie führt einen Zeilenvorschub aus. Der Zusatz NO-SCROLLING definiert die folgende Ausgabezeile als horizontal unverschiebbar. Ihre Position ändert sich beim horizontalen Blättern in der Liste nicht. Diese Einstellung gilt nur für die nächste ausgegebene Zeile und wird dann automatisch deaktiviert! Wenn kein Zusatz benutzt wird, ist die nachfolgende Zeile automatisch verschiebbar.

```
NEW-PAGE [primary_options [secondary_options]].
```

Mit NEW-PAGE wird in der Listenverarbeitung eine neue Seite begonnen. Leere Seiten können nicht erzeugt werden, da dieses Kommando erst im Zusammenhang mit einer echten Ausgabe in die Liste wirksam wird. Zahlreiche Zusätze steuern den weiteren Aufbau der Liste bzw. die Ansteuerung des Druckers beim Ausdruck. Die Zusätze für dieses Kommando sind Tabelle 9.6 zu entnehmen.

Zusatz	Beschreibung
NO-TITLE	Keine Ausgabe von Titelzeilen ab der nächsten Seite.
WITH-TITLE	Ausgabe der Titelzeile.
NO-HEADING	Keine Spaltenüberschriften mehr ausgeben.
WITH-HEADING	Spaltenüberschriften ausgeben.
LINE-COUNT lines	Ab der nächsten Seite neue Zeilenzahl.
LINE-SIZE colums	Ab der nächsten Seite neue List-Breite.
NO-TOPOFPAGE	Das TOP-OF-PAGE-Ereignis wird unterdrückt.
PRINT ON secondary options	Alle folgenden Ausgaben als Druckanweisungen interpretieren. Es sind rund 25 differenzierende sekundäre Zusätze möglich.
PRINT OFF	Beenden der mit PRINT ON eingeleiteten Druckersteuerung.

Tabelle 9.6
Zusätze zum Kommando NEW-PAGE

```
NODES node TYPE type.
```

Dieses Kommando ist nur im Zusammenhang mit logischen Datenbanken interessant. Die Knoten der logischen Datenbank können unterschiedliche Typen besitzen. Das Kommando NODES deklariert in einem Report den Namen eines Knotens der logischen Datenbank und ermöglicht so, dass mit dem Kommando GET Daten aus der logischen Datenbank übernommen werden können. Falls es sich beim Hierarchieknoten in der logischen Datenbank um eine Dictionary-Tabelle handelt, kann auch das Kommando TABLES benutzt werden.

```
ON CHANGE OF field [OR field_2].
  ABAP-statements ...
ENDON.
```

Diese Anweisung wird in SELECT-Schleifen oder in GET-Verarbeitungsblöcken benutzt. Sie bewirkt, dass die zwischen ON und ENDON eingeschlossenen Anweisungen nur dann ausgeführt werden, wenn sich der Inhalt von field im Vergleich zur vorangegangenen Ausführung des Kommandos ändert. Mit dem Zusatz OR können beliebig viele weitere Felder definiert werden, die ebenfalls die Verarbeitung der Anweisungen auslösen sollen.

```
OPEN CURSOR [WITH HOLD] cursor FOR SELECT select_statement.
```

Der Datenbankcursor für die in der SELECT-Anweisung angegebenen Tabelle wird geöffnet. Der Cursor muss vom Typ CURSOR sein, also mit TYPE CURSOR deklariert werden.

```
OPEN DATASET filename
  [FOR OUTPUT | FOR INPUT | FOR APPENDING | FOR UPDATE]
  [IN [ LEGACY ] ( BINARY MODE | TEXT MODE ) [ options ] ]
  [AT POSITION pos]
  [TYPE attribut]
  [REPLACEMENT CHARACTER char]
  [IGNORING CONVERSION ERRORS]
  [MESSAGE textfield]
  [FILTER statement].
```

Die angegebene Datei wird geöffnet. Es handelt sich um eine Datei auf Betriebssystemebene des Applikationsservers. Der Dateiname kann bzw. muss daher den Anforderungen des verwendeten Betriebssystems genügen und gegebenenfalls Laufwerks- und Pfadangaben enthalten. Über einen der FOR-Zusätze wird festgelegt, ob aus der Datei gelesen oder in sie geschrieben werden soll. Der Zusatz IN TEXT MODE bewirkt, dass Lesevorgänge immer eine ganze Zeile lesen. Das Zeilenende wird durch ein (betriebssystemabhängiges) Zeilenendezeichen erkannt. Nach jedem Schreibvorgang wird dieses Zeilenendezeichen automatisch angefügt. Der Zusatz IN BINARY MODE (Standardvorgabe) sorgt für die Unterdrückung jedweder Verarbeitung der Daten. Mit AT POSITION kann eine beliebige Position innerhalb der Datei als Startposition gesetzt werden. Mit TYPE werden betriebssystemabhängige Attribute an das System übergeben. Die Verwendung der Option MESSAGE stellt eine vom Betriebssystem gelieferte Fehlermeldung in das angegebene Feld, falls das Öffnen der Datei fehlschlägt. Das Feld statement im Zusatz FILTER gibt ein Kommando des Betriebssystems an, mit dem die zu bearbeitenden Daten behandelt werden.

Diverse weitere Zusätze und Optionen regeln das Verhalten bei der Zeichensatzkonvertierung.

```
OVERLAY field_1 WITH field_2 [ONLY field_3].
```

Der Inhalt von field_2 wird zeichenweise in field_1 übertragen, allerdings nur für die Positionen in field_1, an denen ein Leerzeichen steht. Mit ONLY kann eine alternative Menge von Zeichen definiert werden, die in field_1 überschrieben werden soll. Alle beteiligten Felder werden unabhängig von ihrem tatsächlichen Typ als Zeichenfelder (Typ C) behandelt.

```
PACK source TO target.
```

Der Inhalt des Quellfelds wird gepackt und im Zielfeld abgelegt.

```
PARAMETERS parameter [options].
```

In der Listenverarbeitung wird ein Parameter deklariert, der auf dem Selektionsbildschirm erscheint. Über eine Vielzahl von Zusätzen können Eigenschaften des Parameters gezielt gesetzt werden. Die möglichen Zusätze sind Tabelle 9.7 zu entnehmen.

Zusatz	Beschreibung
AS CHECKBOX	Der Parameter erscheint als Ankreuzfeld.
AS LISTBOX	Das Eingabefeld wird als Drop-Down-Listbox dargestellt. Die Wertvorgabe erfolgt über die Eingabehilfe des Dictionary-Felds, auf dem der Parameter beruht.
AS MATCHCODE STRUCTURE	Datenbankspezifischer Parameter zur Selektion über Matchcode.
AS SEARCH PATTERN	Es wird ein komplexes Objekt zur Aufnahme der Daten eines komplexen Musters erzeugt.
DECIMALS	Angabe der Nachkommastellen bei nummerischen Feldern.
DEFAULT value	Der Parameter wird vor der Einblendung des Selektionsbildes mit einem Vorgabewert gefüllt.
FOR NODE node	Zuordnung eines datenbankspezifischen Parameters zu einer Tabelle (nur im Zugriffsprogramm einer logischen Datenbank).
FOR TABLE table	Zuordnung eines datenbankspezifischen Parameters zu einer Tabelle (nur im Zugriffsprogramm einer logischen Datenbank).
HELP-REQUEST	Nur für datenbankspezifische Parameter. Dem Parameter kann eine individuelle Hilfe über den Zeitpunkt AT SELECTION SCREEN ON HELP REQUEST zugeordnet werden.

Tabelle 9.7
Zusätze der Anweisung PARAMETERS

Zusatz	Beschreibung
LENGTH len	Das Eingabefeld erhält die angegebene Länge.
LIKE field	Die Eigenschaften des Parameterfeldes werden von einem vorhandenen Feld abgeleitet.
LIKE (namefield)	Die Eigenschaften des Parameterfeldes werden von einem vorhandenen Feld abgeleitet, dessen Name dynamisch übergeben wird.
LOWER CASE	Keine automatische Umwandlung in Großbuchstaben.
MATCHCODE OBJECT matchcode	Ein Matchcode wird mit dem Parameterfeld des Selektionsbildes verbunden.
MEMORY ID G/S-parameter	Das Parameterfeld im Selektionsbild wird mit einem Get-/Set-Parameter verbunden.
MODIF ID modification_group	Das Parameterfeld im Selektionsbild erhält als Attribut die angegebene Modifikationsgruppe.
NO-DISPLAY	Der Parameter erscheint nicht auf dem Selektionsbild. Der Wert für den Parameter wird nur bei SUBMIT übergeben.
OBLIGATORY	Für diesen Parameter muss im Selektionsbild eine Eingabe erfolgen.
RADIOBUTTON GROUP group	Der Parameter erscheint als Auswahlfeld innerhalb der angegebenen Gruppe.
TYPE type	Das Parameterfeld erhält den angegebenen Datentyp.
USER-COMMAND command	Für Parameter des Typs Checkbox oder Radiobutton wird beim Anklicken das angegebene Kommando ausgelöst.
VALUE CHECK	Prüft den eingegebenen Wert gegen eine Prüftabelle. Der Parameter muss dazu aber mit LIKE von einem Dictionary-Feld abgeleitet sein.
VALUE-REQUEST	Nur für datenbankspezifische Parameter. Dem Parameter kann eine Eingabehilfe über den Zeitpunkt AT SELECTION SCREEN ON VALUE REQUEST zugeordnet werden.
VISIBLE LENGTH length	Setzen der Breite des Eingabefeldes auf dem Selektionsbildschirm.

Tabelle 9.7
Zusätze der Anweisung PARAMETERS (Fortsetzung)

```
PERFORM [subroutine | index]
  [ (program) |
    IN PROGRAM program  |
    OF { subroutine_n } |
    ON COMMIT [LEVEL index] |
    ON ROLLBACK ]
  [TABLES   { itab_n } ]
  [USING    { parameter_n } ]
  [CHANGING { parameter_n } ]
  [IF FOUND].
```

Das angegebene, mit `FORM` definierte Unterprogramm wird aufgerufen. Mittels der Zusätze `TABLES`, `USING` und `CHANGING` werden interne Tabellen oder Feldparameter als aktuelle Parameter übergeben.

Mittels der Zusätze `(program)` oder `IN PROGRAM` werden Unterprogramme aus anderen Programmen aufgerufen. Die Übergabe des Programm- und Unterprogrammnamens kann im Falle von `IN PROGRAM` auch dynamisch erfolgen. Eventuelle Laufzeitfehler, die entstehen würden, wenn das aufgerufene externe Unterprogramm nicht verfügbar ist, werden mit `IF FOUND` unterbunden. Mit `OF` kann das aufzurufende Unterprogramm über einen Index angesprochen werden. Die in Frage kommenden Unterprogrammnamen folgen nach `OF`, nach `PERFORM` steht kein Name, sondern ein Datenfeld, das einen gültigen Index enthalten muss.

Mittels des Zusatzes `ON COMMIT` erfolgt die Ausführung des Unterprogramms erst bei einem `COMMIT WORK`. Eine Parameterübergabe ist in diesem Fall nicht möglich, die Daten müssen in programminternen Feldern oder im globalen Memory bis zum Verbuchungszeitpunkt aufbewahrt werden. Durch einen optionalen Index können Sie festlegen, in welcher Reihenfolge die Unterprogramme beim Commit aufgerufen werden.

```
POSITION column.
```

In der Listenausgabe wird die Spaltenposition für die nächste Ausgabe gesetzt.

```
PRINT-CONTROL [options].
```

Über die Zusätze werden verschiedene Einstellungen für die Druckformate der nachfolgenden Ausgabezeilen gesetzt. Die zahlreichen Zusätze, die zum Teil weitere Optionen erhalten können, sind in Tabelle 9.8 aufgeführt.

```
PRINT-CONTROL INDEX-LINE field.
```

Ausgabe des Inhaltes von `field` als unsichtbare Indexzeile. Diese Indexzeile wird nur im Zusammenhang mit der optischen Archivierung benötigt. Das Feld muss einer vorgegebenen Struktur entsprechen und mit speziell aufbereiteten Werten gefüllt werden.

```
PRIVATE SECTION.
```

Diese Anweisung leitet in ABAP Objects innerhalb der Definition einer Klasse den privaten Komponentenbereich ein. Alle in diesem Abschnitt deklarierten Elemente sind außerhalb der Klasse nicht sichtbar.

Zusatz	Beschreibung
CPI	Angabe der Zahl der Zeichen je Zoll.
LPI	Angabe der Zahl der Zeilen je Zoll.
SIZE	Angabe der Schriftgröße.
COLOR	Angabe der Ausgabefarbe bei farbfähigen Druckern (BLACK, RED, BLUE, GREEN, YELLOW oder PINK).
LEFT MARGIN	Setzen des linken Randes.
FONT	Angabe des Zeichensatzes.
FUNCTION	Direktes Ansprechen einer Funktion.
LINE	Angabe der Ausgabezeile, für die PRINT-CONTROL wirksam werden soll.
POSITION	Angabe der Spalte in der mit LINE bestimmten Ausgabezeile, für die PRINT-CONTROL wirksam werden soll.

Tabelle 9.8
Zusätze des Kommandos PRINT-CONTROL

```
PROGRAM.
```

Diese Anweisung leitet ein neues Programm ein. Sie ist äquivalent zur REPORT-Anweisung und kann alternativ zu dieser benutzt werden. Es sind dieselben Zusätze wie bei REPORT möglich. Beim automatischen Generieren von Programmrümpfen durch das System wird REPORT für Reports und PROGRAM für Modul-Pools benutzt.

```
PROTECTED SECTION.
```

Diese Anweisung leitet in ABAP Objects innerhalb der Definition einer Klasse den geschützten Komponentenbereich ein. Auf alle in diesem Abschnitt deklarierten Elemente kann nur von der aktuellen Klasse aus und von allen Subklassen zugegriffen werden.

```
PROVIDE { {field_x} FROM itab} BETWEEN first AND last.
  ABAP-statements ...
ENDPROVIDE.
```

Zunächst werden die Inhalte der angegebenen Felder der internen Tabellen intervallbezogen bereitgestellt. Anschließend werden für jedes Intervall die Anweisungen zwischen PROVIDE und ENDPROVIDE ausgeführt.

```
PUBLIC SECTION.
```

Diese Anweisung leitet in ABAP Objects innerhalb der Definition einer Klasse den öffentlichen Komponentenbereich ein. Auf die in diesem Abschnitt deklarierten Elemente kann von außen ohne Einschränkung zugegriffen werden.

```
PUT node | <node>.
```

Diese Anweisung ist nur im Zugriffsprogramm einer logischen Datenbank verwendbar. Sie löst das Ereignis GET im zugehörigen Report aus. Anschließend werden alle PUT-Unterprogramme aller untergeordneten Knoten ausgeführt, sofern für diese Knoten GET-Zeitpunkte im Report existieren. Falls der Knoten vom Typ dynamischer Dictionary-Typ ist, muss statt der statischen Angabe eines Knotens (node) der Zugriff auf ein Feldsymbol erfolgen (<node>).

```
RAISE exception.
```

Mit diesem Kommando wird innerhalb eines Funktionsbausteins eine Ausnahme ausgelöst. Falls diese Ausnahme im rufenden Programm nicht behandelt wird, erzeugt das System einen Laufzeitfehler.

```
RAISE EVENT event
  [EXPORTING { parameter = field }].
```

Auslösen einer klassenbasierten Ausnahme innerhalb ABAP Objects. Dieses Ereignis kann von speziellen Methoden empfangen und ausgewertet werden.

```
RAISE EXCEPTION
  TYPE class [EXPORTING { parameter = field }] |
  reference.
```

Auslösen eines Ereignisses innerhalb ABAP Objects. Die Ausnahme ist dabei ein Objekt. Es wird entweder neu instanziiert oder aber es wird ein existierendes Ausnahmeobjekt weitergereicht. Beim Instanziieren mit TYPE können optional durch EXPORTING Parameter an das neue Objekt übergeben werden.

```
RANGES selection_table FOR field [OCCURS n].
```

Es wird eine Selektionstabelle für das angegebene Feld definiert. Die Struktur der Tabelle entspricht der von SELECT-OPTIONS erzeugten Selektionstabelle. Die Tabelle kann im Programm wie eine Selektion benutzt werden. Die mit RANGES definierten Tabellen, auch Ranges-Tabellen genannt, erscheinen allerdings nicht auf Selektionsbildern, sondern müssen im Programm manuell mit den Werten für eine gültige Selektion gefüllt werden. Bei Bedarf kann ein OCCURS-Parameter angegeben werden, um den Speicherbereich für die Tabelle den realen Bedürfnissen anzupassen.

```
READ CURRENT LINE
  [FIELD VALUE {source INTO target}]
  [LINE VALUE INTO record].
```

Die aktuelle Zeile der Liste (ausgewählt durch Zeilenselektion oder `READ LINE line`) wird nochmals gelesen. Die gelesenen Felder oder die gesamte Zeile können mit `FIELD VALUE` bzw. `LINE VALUE` in anderen als den Ursprungsfeldern abgelegt werden.

```
READ DATASET filename
  INTO field [
    [ [ACTUAL] LENGTH length].
    [ MAXIMUM LENGTH maxlength].
```

Aus einer mit `OPEN` geöffneten Betriebssystemdatei wird ein Datensatz gelesen und im angegebenen Feld bereitgestellt. Wurde die Datei im Binärmodus geöffnet, werden so viele Zeichen gelesen wie notwendig sind, um das Zielfeld zu füllen. Die Struktur des Zielfelds sollte daher der Struktur der abgelegten Daten entsprechen. Aus Dateien, die im Textmodus geöffneten wurden, liest das Kommando jeweils eine Zeile. Dabei gehen Zeichen verloren, wenn die Länge des Zielfeldes geringer ist als die Länge der gelesenen Zeile. Mit dem Zusatz `LENGTH` kann die Zahl der tatsächlich gelesenen Zeichen im Feld `length` bereitgestellt werden, durch den Zusatz `MAXIMUM LENGTH` ist die Vorgabe .

```
READ LINE line_number
  [INDEX list]
  [FIELD VALUE {source INTO target}]
  [LINE VALUE INTO record]
  [OF CURRENT PAGE | OF PAGE page].
```

Es wird eine über die Zeilennummer identifizierte Zeile aus einer Liste gelesen. Ohne Zusätze erfolgt die Zeilenzählung global. Die beiden `PAGE`-Zusätze bewirken eine seitenbezogene Zeilenzählung; gelesen wird entweder auf der aktuellen oder der explizit angegebenen Seite. Gelesen wird in der aktuellen Liste, falls mit dem Zusatz `INDEX` nicht eine List-Ebene gewählt wird. Durch den Zusatz `FIELD VALUE` werden die Feldinhalte der List-Zeile nicht in den Ursprungsfeldern bereitgestellt, sondern in ausdrücklich angegebenen anderen programminternen Feldern. Gleiches wird für die gesamte Zeile durch den Zusatz `LINE VALUE` erreicht.

```
READ REPORT program INTO itab.
```

Der Quelltext des angegebenen Programms wird in die interne Tabelle `itab` gelesen. Dieses Kommando liefert in geschützten Systemen nur die Quelltexte von Nicht-SAP-Programmen.

```
READ TABLE table.
```

Dieses Kommando ist veraltet. Seine Aufgabe (Lesen eines Satzes in einer Datenbanktabelle) wurde inzwischen durch die `SELECT`-Anweisung übernommen.

```
READ TABLE itab [INTO record]
  [ WITH KEY {field = key} | = key | key [BINARY SEARCH] |
    INDEX index]
  [ASSIGNING <field_symbol> | REFERENCE INTO reference]
  [COMPARING fieldlist | ALL FIELDS]
  [TRANSPORTING fieldlist | NO FIELDS].
```

Diese Anweisung liest einen Datensatz aus einer internen Tabelle des Typs Standard- Tabelle. Das Ergebnis wird entweder in der Kopfzeile der Tabelle oder in einem mit `INTO` benannten Arbeitsbereich zur Verfügung gestellt. Übertragen werden dabei alle Felder, sofern nicht mit `TRANSPORTING` eine Auswahl angegeben wird. Falls es nur darauf ankommt, einen Datensatz zu finden (Existenzprüfung bzw. Indexermittlung), spart das Unterdrücken der Wertübertragung mit `TRANSPORTING NO FIELDS` Abarbeitungszeit.

Der zu lesende Datensatz kann durch verschiedene Methoden ausgewählt werden. Mit `INDEX` wird die Datensatznummer angegeben. Durch die unterschiedlichen `WITH KEY`-Zusätze wird ein Suchschlüssel definiert, entweder feldweise oder durch Angabe eines Arbeitsbereiches mit mehreren Schlüsselwerten. Im Zusammenhang mit `WITH KEY` kann mit dem Zusatz `BINARY SEARCH` die Suche beschleunigt werden, wenn die Tabelle entsprechend dem verwendeten Schlüssel vorsortiert ist. Die Anweisung führt dann anstelle der sequenziellen eine binäre Suche durch. Die Standardform des `READ`-Kommandos (ohne `WITH KEY`-Zusatz) erwartet den Schlüssel in der Kopfzeile der internen Tabelle. Der Schlüssel besteht dann aus allen Feldern der Kopfzeile, die nicht den Typ I, F oder P haben und deren Inhalt ungleich dem Leerzeichen ist.

Der Zusatz `COMPARING` führt einen zusätzlichen Vergleich zwischen den Feldern des gelesenen Satzes und der Kopfzeile aus. Er verhindert nicht die Übertragung des Satzes, sondern setzt nur das Systemfeld `SY-SUBRC`. Er ist zur Überprüfung von Nicht-Schlüsselfeldern hilfreich.

Durch Zuweisung eines Feldsymbols oder einer Referenz mittels der Optionen `ASSIGNING` oder `REFERENCE INTO` entfällt das Kopieren der Daten in einen Arbeitsbereich oder in die Kopfzeile. Statt dessen wird ein Zeiger direkt auf den Datensatz innerhalb des Datenbereiches der internen Tabelle gesetzt. Modifikationen des Datensatzes über das Feldsymbol werden somit sofort wirksam.

Dieses Kommando wird nur noch aus Kompatibilitätsgründen unterstützt und kann problemlos durch eine der generischen Varianten abgelöst werden.

```
READ TABLE itab
  [ [FROM record] |
    [WITH TABLE KEY {field = value}] |
    [WITH KEY {field = value} [BINARY SEARCH] ] |
    [INDEX index] ]
  [INTO record]
  [ASSIGNING <field_symbol> | REFERENCE INTO reference]
  [COMPARING fieldlist | ALL FIELDS]
  [TRANSPORTING fieldlist | NO FIELDS].
```

Dieses Kommando liest ebenfalls einen Datensatz einer internen Tabelle. Es arbeitet allerdings generisch und kann daher alle Tabellenarten verarbeiten. Da beim Zugriff auf sortierte oder Hash-Tabellen die Angabe eines Schlüssels obligatorisch ist, kann mit FROM der Verweis auf einen Datensatz erfolgen, der den erforderlichen Schlüssel enthält. Der Schlüssel kann natürlich auch durch die WITH-Klausel übergeben werden. Beachten Sie dabei, dass mit WITH TABLE KEY der in der Tabellendefinition festgelegte komplette Schlüssel benutzt werden muss, während bei WITH KEY beliebige Felder zur Suche verwendet werden können.

```
READ TEXTPOOL program INTO itab LANGUAGE language.
```

Diese Anweisung liest die Textelemente eines Programms der angegebenen Sprache in eine interne Tabelle.

```
RECEIVE RESULTS FROM FUNCTION function
  [KEEPING TASK]
  [IMPORTING   { parameter = field }]
  [TABLES      { parameter = itab  }]
  [EXCEPTIONS [{ exception = value }]
              [OTHERS = value ]
              [ERROR_MESSAGE = value ]].
```

Mit diesem Kommando werden die Rückgabewerte eines asynchron aufgerufenen RFC-Funktionsbausteins abgefragt. Der Einsatz dieses Kommandos erfordert eine spezielle Programmstruktur und die Beachtung einiger Randbedingungen.

```
REFRESH itab.
```

Die interne Tabelle itab wird zurückgesetzt, alle in ihr enthaltenen Datensätze werden gelöscht. Die Kopfzeile bleibt allerdings unverändert.

```
REFRESH CONTROL control FROM SCREEN screen.
```

Das mit der Anweisung CONTROLS definierte Element control wird im angegebenen Dynpro neu initialisiert.

```
REFRESH SCREEN.
```

Dieses Kommando stellt das aktuelle Dynpro nach Empfang eines RFC-Ereignisses neu dar. Es entspricht in seiner Wirkung der manuellen Betätigung der Datenfreigabetaste. Es ist nur im Zusammenhang mit dem Kommando RECEIVE RESULTS FROM FUNCTION sinnvoll verwendbar. Das REFRESH SCREEN-Kommando ist inzwischen veraltet, stattdessen sollte SET USER-COMMAND genutzt werden.

```
REJECT [table].
```

Dieses Kommando beendet die Verarbeitung des aktuellen Datensatzes der aktuellen Datenbanktabelle. Die Verarbeitung wird mit dem nächsten Satz fortgesetzt. Mit expliziter Angabe eines Dateinamens wird der nächste Datensatz der angegebenen Tabelle verarbeitet. Diese Form des Aufrufs ist beispielsweise innerhalb der GET-Verarbeitung bei logischen Datenbanken nützlich. Die angegebene Tabelle muss sich auf derselben oder einer höheren Hierarchieebene befinden als die aktuelle Tabelle.

```
REPLACE
  pattern_1 WITH pattern_2 INTO field
  [LENGTH length]
  [ IN BYTE | CHARACTER MODE ].
```

Dieses Kommando führt eine Zeichenkettenverarbeitung durch. Unabhängig von ihrem tatsächlichen Typ werden alle beteiligten Felder als Felder des Typs C behandelt. Im Zielfeld field wird das erste Auftreten von pattern_1 durch pattern_2 ersetzt. Dabei werden auch abschließende Leerzeichen berücksichtigt! Dies kann nur durch den Zusatz LENGTH verhindert werden, mit dem die relevante Länge des Suchmusters pattern_1 bestimmt werden kann.

Diese Variante des Replace-Kommandos ist inzwischen veraltet und sollte nicht mehr benutzt werden.

```
REPLACE [ FIRST OCCURRENCE OF | ALL OCCURRENCES OF ]
  [ SUBSTRING ] pattern
  IN [ SECTION OFFSET offset LENGTH length OF ] field
  WITH new
  [ IGNORING | RESPECTING CASE [ IN CHARACTER MODE ] |
    IN BYTE MODE ]
  [ REPLACEMENT COUNT rep_count ]
  [ REPLACEMENT OFFSET rep_offset ]
  [ REPLACEMENT LENGTH rep_length ].
```

Dies ist das aktuelle Kommando zum Ersetzen von Zeichen innerhalb einer Zeichenkette. Innerhalb der Zeichenkette field wird das Muster pattern gesucht und durch die neue Zeichenkette new ersetzt. Verschiedene Optionen beeinflussen das Verhalten des Kommandos. Zunächst kann durch die OCCURENCE-Zusätze festgelegt werden, ob nur das erste oder alle Fundstellen von pattern ersetzt werden sollen. Die SECTION-Option schränkt den Suchbereich auf einen Teilbereich der zu bearbeitenden Zeichenkette ein. Des Weiteren ist es möglich, zwischen Groß- und Kleinschreibung zu unterscheiden. Außerdem können binäre Daten, die in einem Zeichenkettenfeld abgelegt wurden, bearbeitet werden.

Die drei REPLACEMENT-Zusätze stellen Informationen zur Ausführung des Kommandos in die angegebenen Felder. Es handelt sich dabei um die Zahl der durchgeführten Ersetzungen, die Position der letzten Ersetzung sowie die Länge der eingefügten Zeichenkette.

```
REPLACE SECTION [ OFFSET offset] [ LENGTH length]
  OF string
  WITH new
 [ IN BYTE | CHARACTER MODE ].
```

Diese Form des REPLACE-Kommandos führt eine positionsbasierte Ersetzung aus. Dabei wird der zu ersetzende Bereich durch eine Offset- und Längenangabe bestimmt.

```
REPORT program
  [NO STANDARD PAGE HEADING]
  [LINE-SIZE number_of_columns]
  [LINE-COUNT number_of_lines[(number_of_footer_lines)]]
  [MESSAGE-ID message_class]
  [DEFINING DATABASE logical_database].
```

Diese Anweisung leitet einen Report ein. Einige Eigenschaften des Reports werden durch die optionalen Zusätze bestimmt. Mit NO STANDARD PAGE HEADING wird die Ausgabe des standardmäßigen Seitenkopfes abgeschaltet. Der Zusatz LINE-SIZE bestimmt die Breite der Liste in Zeichen. Der Maximalwert beträgt 255. Fehlt der Zusatz, wird die Seitenbreite zur Laufzeit des Reports anhand der aktuellen Fenstergröße bestimmt. Dies kann Auswirkungen auf den Aufbau der Liste haben! Mit LINE-COUNT werden die Seitenlänge und der für eventuelle Fußzeilen zu reservierende Bereich angegeben. Der Standardwert für Fußzeilen ist 0. Fehlt auch die Seitenlänge, werden Seitenwechsel nur noch durch das Kommando NEW-PAGE ausgelöst. Wenn im Report Nachrichten ausgegeben werden sollen, kann mit MESSAGE-ID der Name einer Nachrichtenklasse gesetzt werden, die für die vereinfachte Form des MESSAGE-Kommandos notwendig ist.

Der Zusatz DEFINING DATABASE ist nur in den Zugriffsprogrammen einer logischen Datenbank erforderlich. Da diese Programme automatisch generiert werden, muss dieser Zusatz in der Regel nie manuell notiert werden.

```
RESERVE number_of_lines LINES.
```

Diese Anweisung löst in der List-Verarbeitung einen Seitenvorschub aus, wenn die angegebene Zahl von Zeilen auf der aktuellen Seite nicht mehr verfügbar ist.

```
RETURN.
```

Der aktuelle Verarbeitungsblock wird verlassen. Ein Verarbeitungsblock ist dabei nicht der Körper einer Schleifenanweisung, sondern eine komplette Modularisierungseinheit wie z.B. ein Unterprogramm oder einer der Blöcke, die im Reporting durch eine Zeitpunkt-Anweisung gebildet werden.

```
ROLLBACK WORK.
```

Alle Änderungen der Datenbank seit dem letzten COMMIT WORK werden rückgängig gemacht.

```
SCROLL LIST [INDEX index]
  [LINE line] TO FIRST PAGE    |
  [LINE line] TO LAST PAGE     |
  [LINE line] TO PAGE page     |
  [LINE line] TO COLUMN column |
  [ FORWARD | BACKWARD [page PAGES] ] |
  [ LEFT | RIGHT [BY characters PLACES]].
```

Mit diesem Kommando kann in der List-Verarbeitung ein programmgesteuertes Rollen des aktuellen Ausschnittes der Liste durchgeführt werden. Das Kommando SCROLL ist nur im Zusammenhang mit einem der Zusätze (TO FIRST PAGE ... RIGHT) vollständig. Die Wirkung dieser Zusätze ergibt sich aus ihrem Namen.

Außer den obligatorischen Zusätzen existieren weitere optionale Ergänzungen. Mit dem Zusatz INDEX wird dabei die Nummer der Liste (List-Stufe) angegeben. Die übrigen Zusätze geben Richtung und Umfang des Scrollens an. Mit LINE wird die Zeile auf der Zielseite angegeben, auf die positioniert werden soll. Mit PAGES wird bei den Zusätzen FORWARD und BACKWARD die Zahl der Seiten festgelegt, um die der Ausschnitt bewegt werden soll. Analog dazu kann beim horizontalen Rollen mit LEFT oder RIGHT die Zahl der Zeichen, um die der Ausschnitt bewegt werden soll, mit dem Zusatz BY PLACES definiert werden.

```
SEARCH [ field | itab ] FOR string
  [ABBREVIATED]
  [STARTING AT first]
  [ENDING AT last]
  [AND MARK]
  [IN BYTE | CHARACTER MODE].
```

Das angegebene Feld oder die interne Tabelle (die gesamte Tabelle, nicht aber die Kopfzeile) werden nach der angegebenen Zeichenkette durchsucht. Die Zeichenkette kann das Musterzeichen „*“ als Platzhalter für beliebige Zeichen entweder am Anfang oder am Ende enthalten. Mit dem Zusatz ABBREVIATED findet das Kommando das Suchmuster auch dann, wenn die zu suchenden Zeichen im Suchfeld durch andere Zeichen voneinander getrennt sind. Mit STARTING bzw. ENDING wird die Suche auf den angegebenen Bereich im Suchfeld beschränkt. Bei internen Tabellen geben diese Zusätze die zu durchsuchenden Zeilen an. Der Zusatz AND MARK wandelt die gefundene Zeichenkette im Suchfeld oder in der internen Tabelle in Großbuchstaben um.

Alle beteiligten Felder, insbesondere die Datensätze der internen Tabelle, werden als Felder des Typs C behandelt.

```
SELECT result [target]
  [INTO fieldlist]
  FROM source
  [WHERE condition]
```

```
  [GROUP BY fieldlist]
  [HAVING condition]
  [ORDER BY sortorder].
  [ ABAP-statements ]
[ ENDSELECT. ]
```

Die SELECT-Anweisung dient zum Lesen von Datensätzen aus Datenbanktabellen. Sie besteht aus mehreren so genannten Klauseln, die das Verhalten der Anweisung bestimmen. Die Funktionalität der SELECT-Anweisung ist in starkem Maße vom aktuellen Releasestand abhängig. In ihrer Grundform ist die SELECT-Anweisung eine Schleifenanweisung, die mit der Anweisung ENDSELECT abgeschlossen werden muss.

Wegen des Umfangs der Anweisung werden die einzelnen Klauseln getrennt voneinander beschrieben. Allerdings kann die korrekte Syntax der SELECT-Anweisung durch die Meta-Beschreibung nicht völlig korrekt beschrieben werden, da starke Abhängigkeiten der Klauseln untereinander bestehen.

Die SELECT-Klausel

```
SELECT [[SINGLE [FOR UPDATE]] | [DISTINCT]] fieldlist |
(itab)
```

Die Angaben der SELECT-Klausel geben die Art der Ausgabedaten (einzelner Satz oder Datenmenge) sowie die zurückzuliefernden Felder an. Mit dem Zusatz SINGLE wird die Rückgabe eines einzelnen Satzes erzwungen. Bei diesem Zusatz entfällt die ENDSELECT-Anweisung. Der Datensatz muss durch die WHERE-Klausel eindeutig identifiziert werden. Der optionale Zusatz FOR UPDATE sperrt diesen Satz gegen Bearbeitung durch Dritte.

Der Zusatz DISTINCT unterdrückt identische Ausgabesätze. Die SELECT-Klausel erfordert die Angabe einer Feldliste, in der alle zurückzuliefernden Felder enthalten sind. Diese Feldliste kann aus einem Stern als Symbol für alle Felder, einer Aufzählung von Feldern oder Aggregatausdrücken bestehen.

Die INTO-Klausel

```
INTO record |
     (field [,field]) |
     [CORRSPONDING FIELDS OF] workarea |
     [CORRSPONDING FIELDS OF] TABLE itab
          [PACKAGE SIZE number_of_records ]]
     [APPENDING [CORRSPONDING FIELDS OF] TABLE itab
          [PACKAGE SIZE number_of_records ]]
```

Die INTO-Klausel bestimmt das Ziel der gelesenen Datensätze. Je nach Feldliste der SELECT-Klausel können diese ein Arbeitsbereich (eine Feldleiste), eine Liste von Einzelfeldern oder eine interne Tabelle sein. Das Ziel wird vor dem Überschreiben gelöscht. Im Zusammenhang mit internen Tabellen wird durch die

anstelle von INTO zu verwendende Klausel APPENDING das Anfügen von Daten an die interne Tabelle veranlasst. Durch die optionalen CORRESPONDING-FIELDS-Zusätze werden nur die Inhalte namensgleicher Felder übertragen.

Beim Übertragen des Selektionsergebnisses ist ein ENDSELECT nicht notwendig, es sei denn, dass mit PACKAGE SIZE eine Zahl von Datensätzen vorgegeben wurde, die maximal in die interne Tabelle geschrieben werden soll. In diesem Fall werden so viele Schleifendurchläufe ausgeführt, bis alle Datensätze gelesen wurden.

Die FROM-Klausel

```
FROM table [AS alias] |
     (field_with_tablename) |
     tab1 [INNER] JOIN tab2 ON condition |
     tab1 LEFT [OUTER] JOIN tab2 ON condition
     [CLIENT SPECIFIED] [BYPASSING BUFFER] [UP TO n ROWS]
```

Neben der SELECT-Klausel ist die FROM-Klausel obligatorisch. Sie bestimmt, aus welcher Tabelle die Daten gelesen werden sollen. Der Tabellenname kann dabei statisch oder dynamisch angegeben werden. Drei optionale Zusätze bestimmen Details des Selektionsvorgangs. Sie gehören logisch nicht unbedingt zur FROM-Klausel, die Zuordnung ist eher historisch bedingt. Mit CLIENT SPECIFIED wird das automatische Mandantenhandling abgeschaltet. Das Mandantenfeld muss dann manuell ausgewertet werden. Der Zusatz BYPASSING BUFFER löst das direkte Lesen von der Datenbank aus, eventuelle Puffer werden dabei ignoriert. Mit UP TO n ROWS kann bestimmt werden, wie viele Datensätze zurückgeliefert werden sollen.

In der FROM-Klausel können weiterhin durch die Angabe von Tabellenverknüpfungen durch das Schlüsselwort JOIN mehrere Tabellen miteinander verbunden werden. Ein so genannter INNER JOIN verbindet zwei Tabellen anhand einer logischen Bedingung. Die Ergebnismenge des Joins enthält alle Spalten der beteiligten Tabellen, wobei nur Datensätze aus tab1 berücksichtigt werden, zu denen in tab2 ein Datensatz existiert, der die angegebene Bedingung erfüllt. Bei einem OUTER JOIN sind in der Ergebnismenge alle Datensätze aus tab1 enthalten. Diese werden durch die Daten aus tab2 ergänzt, falls ein passender Datensatz existiert. Wenn nicht, werden die übrigen Felder mit Null aufgefüllt.

Die WHERE-Klausel

```
[FOR ALL ENTRIES IN itab_param ] WHERE condition |
(itab_condition)
```

Die WHERE-Klausel enthält die logische Bedingung, nach der Datensätze aus der Tabelle selektiert werden. Die in der logischen Bedingung enthaltenen Parameter werden entweder direkt notiert oder aber durch den Inhalt einer internen Ta-

belle vorgegeben. Die Bedingung kann direkt notiert oder in einer internen Tabelle übergeben werden.

Die GROUP BY-Klausel

```
GROUP BY fieldlist | (itab)
```

Mit GROUP BY können Datensätze mit identischen Merkmalen zu Gruppen zusammengefasst werden. Die Feldliste dieser Anweisung muss mit der Feldliste der SELECT-Klausel korrespondieren. Die Feldliste kann auch in einer internen Tabelle übergeben werden.

Die HAVING-Klausel

```
HAVING condition | (itab)
```

Die durch die HAVING-Klausel vorgegebene Bedingung wird auf die von der GROUP BY-Klausel gelieferten Datensätze angewendet. Die Bedingung kann auch in einer internen Tabelle übergeben werden.

Die ORDER BY-Klausel

```
ORDER BY fieldlist | PRIMARY KEY
```

Die Klausel ORDER BY sortiert die auszugebenden Datensätze entsprechend der angegebenen Feldliste oder nach dem Schlüssel, falls statt der Feldliste der Zusatz PRIMARY KEY benutzt wird.

```
SELECT-OPTIONS selection FOR field | (namefield).
```

Diese Anweisung ist nur in Reports sinnvoll. Sie dient zur Deklaration einer Selektionstabelle. Diese Tabelle kann im Selektions-Dynpro eines Reports durch den Anwender mit Werten gefüllt werden. Die Eigenschaften der Selektion können durch eine Vielzahl unterschiedlicher Parameter modifiziert werden (siehe Tabelle 9.9)

Zusatz	Wirkung
DEFAULT ...	Vorschlagswert(e) setzen.
MEMORY ID	Globaler Parameter als Vorgabewert.
MATCHCODE OBJECT	Linkes Feld im Selektions-Dynpro mit Matchcode.
MODIF ID	Zuweisen einer Modifikationsgruppe zum Attribut SCREEN-GROUP1 zu den Bildschirmfeldern einer Selektion.
NO-DISPLAY	Selektion erscheint nicht auf Selektionsbild.

Tabelle 9.9
Zusätze zur Anweisung SELECT-OPTIONS

Zusatz	Wirkung
LOWER CASE	Unterscheidung von Groß- und Kleinschreibung.
OBLIGATORY	Eingabe erforderlich.
NO-EXTENSION	Nur eine Zeile kann eingegeben werden.
NO INTERVALS	In der Selektion können nur Einzelwerte, aber keine Bereichsangaben erfasst werden.
NO DATABASE SELECTION	Selektion wird in logischen Datenbanken nicht verwendet.
VALUE REQUEST...	Selbst programmierte Werthilfe erlauben.
HELP REQUEST...	Selbst programmierte Hilfe erlauben.
VISIBLE LENGTH len	Breite des Eingabefelds in der Eingabemaske festlegen.

Tabelle 9.9
Zusätze zur Anweisung SELECT-OPTIONS (Fortsetzung)

```
SELECTION-SCREEN BEGIN | END OF SCREEN screen
  [TITLE title]
  [AS WINDOW]
```

Diese Anweisung definiert einen Selektionsbildschirm, der durch die Anweisung CALL SELECTION-SCREEN aufgerufen werden kann. Der Selektionsbildschirm kann mit einem Titel versehen werden, der in der Kopfzeile des Fensters erscheint. Der Zusatz AS WINDOW ermöglicht den Aufruf des Selektionsbilds als modale Dialogbox.

```
SELECTION-SCREEN BEGIN | END OF SCREEN screen AS SUBSCREEN
  [NESTING LEVEL level]
  [NO INTERVALS].
```

Diese Anweisung definiert einen Selektionsbildschirm, der als Subscreen in andere Dynpros eingebunden werden kann. Der Zusatz NESTING LEVEL bestimmt die Zahl der Rahmen innerhalb eines Tab Strips. Durch NO INTERVALLS werden Felder für die HIGH-Parameter ausgeblendet.

```
SELECTION-SCREEN option.
```

Die Anweisung SELECTION-SCREEN dient zur Gestaltung des Selektionsbildschirms. Sie ist daher nur in Reports sinnvoll. Die Anweisung erfordert die Verwendung einer der in Tabelle 9.10 dargestellten Optionen.

Option	Wirkung
BEGIN OF LINE	Alle nachfolgend deklarierten Elemente werden in einer Zeile angeordnet.
END OF LINE	Ende der Deklaration einer Zeile.
SKIP	Einfügen von Leerzeilen.
ULINE	Unterstreichen einer Zeile.
POSITION	Positionieren eines Elements.
COMMENT	Ausgabe eines statischen Textes.
PUSHBUTTON	Erzeugen einer Drucktaste im Selektionsbild.
BEGIN OF BLOCK	Beginn eines Blocks, der mit einem Rahmen versehen werden kann.
BEGIN OF TABBED BLOCK block FOR n LINES	Reservieren eines Subscreen-Bereichs für ein Tab-Strip-Control.
TAB (len) register USER-COMMAND fcode	Registerkarten für ein Tab-Strip-Control anlegen.
END OF BLOCK	Ende der Deklaration eines Blocks.
FUNCTION KEY	Deklaration einer Drucktaste in der Drucktastenzeile.
BEGIN OF VERSION	Beginn der Deklaration einer Version eines Selektionsbilds.
END OF VERSION	Ende einer Versionsdeklaration.
EXCLUDE	Ausschließen von Elementen aus einer Version eines Selektionsbilds.
DYNAMIC SELECTIONS FOR TABLE | NODE	Deklaration so genannter freier Abgrenzungen.
FIELD SELECTION FOR TABLE | NODE	Auswahl der in Feldselektionen verwendeten Tabellenfelder.

Tabelle 9.10
Zusätze zum Kommando SELECTION-SCREEN

```
SET BIT n OF field TO value.
```

Innerhalb des hexadezimalen Feldes `field` (Typ X oder XSTRING) wird das Bit `n` gesetzt. Die Zählung beginnt mit 1. Der Zusatz `TO` erlaubt es, den Wert, auf den das Bit gesetzt wird, zu bestimmen (0 oder 1).

```
SET BLANK LINES [ON | OFF].
```

Mit dieser Anweisung wird festgelegt, ob bei der Ausgabe Leerzeilen unterdrückt werden sollen oder nicht. Die Standardeinstellung ist `SET BLANK LINES OFF`.

```
SET COUNTRY country.
```

Mit dieser Anweisung wird das Länderkennzeichen verändert, das Auswirkungen auf die Darstellung von Dezimalpunkt und Datum hat. Diese Anweisung wirkt über Programmgrenzen hinweg.

```
SET CURSOR
  FIELD field [OFFSET offset | LINE line] |
  LINE line [OFFSET offset] |
  column line.
```

Der Cursor wird auf eine bestimmte Position gesetzt. Die drei möglichen Zusätze spezifizieren die Art der Positionierung. Der Zusatz `FIELD` benötigt als Parameter zunächst ein Feld, in dem der Name des Feldes steht, auf das der Cursor platziert werden soll. Der Zusatz `OFFSET` erlaubt die Angabe einer Zeichenposition innerhalb des Feldes. In Listen oder Step Loops muss mit dem Zusatz `LINE` auch die gewünschte Zeile festgelegt werden, da Feldnamen in diesem Fall mehrdeutig sind.

Als primärer Zusatz positioniert `LINE` den Cursor auf die angegebene Zeile einer Liste oder eines Step Loops. Auch hier ist mit `OFFSET` die Positionierung innerhalb der Zeile möglich.

In Dynpros kann der Cursor mit der direkten Angabe von Zeile und Spalte auf eine beliebige Position gesetzt werden.

```
SET DATASET file
  [ POSITION [ posistion | END-OF-FILE ] ]
  [ ATTRIBUTES attribute ]
```

Mit diesem Kommando werden die Eigenschaften der Datei file geändert. Mindestens einer der Kommando-Zusätze muss angegeben werden. Dabei setzt `POSITION` die Lese- bzw. Schreibposition, während `ATTRIBUTES` die Konvertierung zwischen unterschiedlichen Zeichensätzen beeinflusst.

```
SET EXTENDED CHECK [ON | OFF].
```

Diese Anweisung hat während der Programmabarbeitung keine Funktion. Sie wird lediglich durch den Syntaxcheck des Editors bzw. die Prüftransaktion

SLIN ausgewertet. Mit SET EXTENDED CHECK OFF wird die erweiterte Syntaxprüfung ausgeschaltet, mit SET EXTENDED CHECK ON wieder eingeschaltet.

```
SET HANDLER {handler}
  [FOR ref]
  [ALL INSTANCES]
  [ACTIVATION f].
```

Mit diesem Kommando werden innerhalb von ABAP Objects Methoden registriert, die Ereignisse behandeln sollen.

```
SET HOLD DATA ON | OFF.
```

Dieses Kommando schaltet eine spezielle Dynpro-Eigenschaft ein oder aus. Wurde die Funktion eingeschaltet, speichert das System die in den Feldern eines Dynpros eingetragenen Werte und trägt sie später beim Aufruf desselben Dynpros als Vorschlagswerte ein.

```
SET LANGUAGE language.
```

Ab Ausführung dieses Kommandos werden sprachabhängige Elemente in der neuen Sprache ausgegeben. Die Wirkung ist auf das aktuelle Programm beschränkt.

```
SET LEFT SCROLL-BOUNDARY [COLUMN column].
```

Diese Anweisung fixiert alle Spalten links von der aktuellen oder der mit COLUMN abgegebenen Position auf dem Bildschirm. Horizontales Rollen des Bildschirminhalts wirkt sich nicht auf die so fixierten Spalten aus.

```
SET LOCALE LANGUAGE language
  [COUNTRY country]
  [MODIFIER modification].
```

Mit dieser Anweisung wird die Textumgebung einer Anwendung gesetzt. Die Textumgebung hat Einfluss auf die Verarbeitung von sprachspezifischen Sonderzeichen, z. B. auf deren Sortierung. Die beiden Zusätze ermöglichen die Auswahl zwischen unterschiedlichen Ausprägungen der Sprache.

```
SET MARGIN column line.
```

Diese Anweisung ist nur im Reporting sinnvoll. Druckausgaben erfolgen nach diesem Kommando erst ab der angegebenen Zeile und Spalte. Mit der Anweisung wird also ein zusätzlicher Rand erzeugt.

```
SET PARAMETER ID parameter FIELD field.
```

Füllen des globalen Parameters parameter mit dem Inhalt von field.

```
SET PF-STATUS interface
  [EXCLUDING function_code | itab]
```

```
  [IMMEDIATELY]
  [OF PROGRAM program].
```

Setzen eines Status der aktuellen Oberfläche. Mit dem Zusatz EXCLUDING können ausgewählte Funktionscodes aus diesem Status deaktiviert werden. Ein einzelner zu deaktivierender Status kann als Direktwert übergeben werden. Mehrere Deaktivierungen erfordern die Übergabe der entsprechenden Funktionscodes als interne Tabelle. Der Zusatz IMMEDIATELY setzt den neuen Status auch für die vorhergehende Liste. Damit kann innerhalb einer Verzweigungsliste der Status der aufrufenden Liste geändert werden. Mit der Angabe OF PROGRAM kann ein Status eines anderen Programms benutzt werden.

```
SET PROPERTY OF object attribute = field [NO FLUSH].
```

Diese Anweisung dient zur Bearbeitung von OLE-Objekten. Die Eigenschaft eines mit DATA angelegten und CREATE initialisierten Objekts wird auf den angegebenen Wert gesetzt. Der Zusatz NO FLUSH bewirkt das Sammeln von OLE-Anforderungen in einem Puffer, bis dieser mit FREE geleert wird.

```
SET RUN TIME ANALYZER ON | OFF.
```

Ein- bzw. Ausschalten der Laufzeitüberwachung.

```
SET RUN TIME CLOCK RESOLUTION LOW | HIGH.
```

Umschalten der Auflösung der Laufzeitüberwachung zwischen Milli- und Mikrosekunden.

```
SET SCREEN screen_number.
```

Setzen der Nummer des Dynpros, das nach Beendigung der Verarbeitung des aktuellen Dynpros ausgeführt werden soll. Der Wert 0 für das folgende Dynpro führt zur Beendigung des Programms. Die Anweisung ist nur in Dialoganwendungen sinnvoll.

```
SET TITLEBAR title
  [OF PROGRAM program]
  [WITH { parameter } ].
```

Die Überschrift des aktuellen Fensters wird gesetzt. Die Titel werden mit einem eigenständigen Werkzeug unabhängig vom Quelltext gepflegt und über einen dreistelligen Bezeichner identifiziert. Im Titel können einfache oder nummerierte Platzhalter (&, &1-&9) stehen, die durch die Parameter des WITH-Zusatzes ersetzt werden. Durch den Zusatz OF PROGRAM kann ein Titel eines anderen Programms benutzt werden.

```
SET UPDATE TASK LOCAL.
```

Die so genannte lokale Verbuchung wird eingeschaltet. Verbuchungsvorgänge finden dabei im aktuellen Prozess statt, nach einem COMMIT WORK werden daher

erst alle Verbuchungsaufträge abgearbeitet, bevor das Programm fortgesetzt wird.

```
SET USER-COMMAND field.
```

Diese Anweisung ist nur im Reporting sinnvoll. Das angegebene Feld enthält einen Funktionscode, der zwischengespeichert wird. Bei der nächsten Listenanzeige wird sofort die Verarbeitung zu diesem Funktionscode ausgeführt, als ob er vom Anwender ausgelöst worden wäre.

```
SHIFT string [BY n PLACES | UP TO pattern]
  [CIRCULAR | RIGHT | LEFT] |
  [LEFT DELETING LEADING  pattern_2] |
  [RIGHT DELETING TRAILING pattern_2]
  [IN BYTE | CHARACTER MODE].
```

Diese Anweisung dient zur Bearbeitung von Zeichenketten. In der Grundform wird der Inhalt der Zeichenkette um ein Zeichen nach links verschoben, das erste Zeichen geht damit verloren. Mit dem `BY`-Zusatz kann auch um mehr als ein Zeichen verschoben werden. Mit dem Zusatz `UP TO` wird in der zu bearbeitenden Zeichenkette nach `pattern` gesucht und, falls das Muster gefunden wird, bis zu dieser Position verschoben. Alle drei Varianten können mit weiteren Zusätzen versehen werden, die Einfluss auf die Richtung der Rotation haben. Der Zusatz `LEFT` ist Standard, kann also auch weggelassen werden. Mit `CIRCULAR` erfolgt eine Rotation des Inhaltes, die Zeichen gehen also nicht verloren, sondern werden rechts wieder an die Zeichenkette angefügt. Mit `RIGHT` wird nach rechts rotiert. Die beiden `DELETING`-Zusätze verschieben den Inhalt der Zeichenkette so lange, bis eines der Zeichen aus `pattern_2` an der ersten bzw. letzten Stelle steht.

Alle beteiligten Felder werden unabhängig von ihrem tatsächlichen Typ als Zeichenketten behandelt.

```
SKIP [lines] | [TO LINE line].
```

Diese Anweisung ist nur in der List-Verarbeitung sinnvoll. In der Grundform bewirkt diese Anweisung die Ausgabe einer Leerzeile. Optional kann die Zahl der einzufügenden Leerzeilen gesetzt werden. Außerdem ist die Positionierung auf eine absolute Zeile möglich.

```
SORT [DESCENDING | ASCENDING]
     [BY fieldlist | BY fieldgroup]
     [AS TEXT]
     [STABLE].
```

Die Grundform dieser Anweisung sortiert den Extract-Datenbestand entsprechend der Felder der Feldgruppe `HEADER`. Mit den Zusätzen kann sowohl die Sortierreihenfolge bestimmt (`ASCENDING` und `DESCENDING`) als auch ein anderer Sortierschlüssel (Feldliste oder Feldgruppe) gewählt werden. Alle beteiligten

Felder müssen allerdings in der Feldgruppe HEADER definiert sein. Der Zusatz AS TEXT erzwingt eine Sortierung entsprechend der eingestellten Textumgebung (sprachspezifische Sortierung von Sonderzeichen). Der Zusatz STABLE verhindert die Veränderung der Reihenfolge von Datensätzen mit identischem Sortierschlüssel.

```
SORT itab
  [BY fieldlist]
  [DESCENDING | ASCENDING]
  [BY fieldlist]
  [AS TEXT]
  [STABLE].
```

Mit dieser Anweisung wird eine interne Tabelle sortiert. Dabei dienen alle Felder, die keine Zahlenfelder und keine Tabellen sind, als Sortierschlüssel. Wahlweise kann der Sortierschlüssel per Feldliste angegeben werden. Auch die Sortierreihenfolge kann mit ASCENDING oder DESCENDING festgelegt werden. Der Zusatz AS TEXT erzwingt eine Sortierung entsprechend der aktuellen Textumgebung. Mit dem Zusatz STABLE erzwingen Sie, dass die Reihenfolge von Feldern mit identischem Sortierschlüssel nicht geändert wird. Dies kann ohne diesen Zusatz nicht gewährleistet werden.

```
SPLIT string AT separator_sequence
  INTO (TABLE itab) | (fieldlist)
  [IN BYTE | CHARACTER MODE].
```

Diese Anweisung zerlegt eine Zeichenkette entsprechend der Trennzeichenfolge. Dabei wird die Trennzeichenfolge in ihrer gesamten – also definierten – Länge verwendet, einschließlich der abschließenden Leerzeichen, falls die Trennzeichenfolge mit zu wenigen Zeichen gefüllt wurde! Die Teilfelder werden entweder in den Feldern der Feldliste abgelegt oder jeweils in einem eigenen Datensatz der internen Tabelle. Falls bei der Zerlegung mehr Teilfelder entstehen als Felder in der Feldliste bereitstehen, nimmt das letzte Feld alle Teilfelder auf.

```
START-OF-SELECTION.
```

Diese Zeitpunktanweisung der List-Verarbeitung wird unmittelbar vor dem ersten Zugriff auf eine Tabelle prozessiert. Sie leitet damit das „Hauptprogramm" eines Reports ein. Beim Start eines Reports wird automatisch ein derartiger Zeitpunkt ausgelöst.

```
STATICS:
  declarations ...
.
```

Dieses Kommando ist eine Variante der DATA-Anweisung. Mit ihr werden so genannte statische Variablen in Unterprogrammen (Performs) angelegt. Sie kann mit denselben Zusätzen wie die DATA-Anweisung benutzt werden. Der Unter-

schied zu den mit DATA angelegten Datenfeldern besteht in der Lebensdauer der deklarierten Datenfelder. Die Felder werden beim Verlassen des Unterprogramms, in dem sie definiert wurden, nicht zerstört. Sie bleiben, ebenso wie ihre Werte, erhalten. Der Gültigkeitsbereich ist allerdings auf das Unterprogramm beschränkt, in dem sie angelegt wurden.

```
STOP.
```

Beenden der Datenselektion und Ausgabe der Liste. Nach STOP wird der Zeitpunkt END-OF-SELECTION prozessiert.

```
SUBMIT report | (namefield) [options].
```

Diese Anweisung startet einen Report. Dabei können mit rund zwei Dutzend Optionen nahezu alle wichtigen Parameter des gerufenen Reports gesetzt werden. Eine Auswahl der wichtigsten Zusätze zeigt Tabelle 9.11.

Option	Wirkung
TO SAP-SPOOL options	Ausgabe als Druck-Pool.
EXPORTING LIST TO MEMORY	Ablegen der Ausgabeliste im Hauptspeicher ohne Bildschirmausgabe.
VIA SELECTION-SCREEN	Das Selektionsbild des gerufenen Reports wird angezeigt.
AND RETURN	Nach Ende der Abarbeitung Rückkehr zum rufenden Report.
USING SELECTION-SET	Ausführung des Reports mit einer Variante.
WITH	Füllen von Parametern und Selektionen.

Tabelle 9.11
Wichtige Zusätze des Kommandos SUBMIT

```
SUBTRACT field_1 FROM field_2.
```

Der Inhalt von field_1 wird von field_2 subtrahiert. Das Ergebnis wird in field_2 abgelegt.

```
SUBTRACT-CORRESPONDING record_1 FROM record_2.
```

Es werden nur Felder berücksichtigt, deren Namen sowohl in record_1 als auch in record_2 vorkommen. Die Inhalte der Felder aus record_1 werden von den gleichnamigen Feldern aus record_2 subtrahiert.

```
SUM.
```

In einer LOOP-Schleife werden nach einer Gruppenverarbeitung die Zwischensummen aller Zahlenfelder (Typ F, I und P) in der Kopfzeile der internen Tabelle bereitgestellt.

```
SUMMARY.
```

Diese Anweisung löst dieselbe Funktion aus wie FORMAT INTENSIFIED ON.

```
SUMMING field.
```

Diese Anweisung ist nur im Reporting sinnvoll. Für das angegebene Feld wird ein internes Summenfeld mit Namen SUM_field_name angelegt. Bei jeder Ausgabe des Ursprungsfelds mit WRITE wird dessen aktueller Inhalt zum Summenfeld hinzuaddiert. Die aktuelle Summe steht zu jedem beliebigen Zeitpunkt, z. B. auch nach END-OF-SELECTION, zur Verfügung.

```
SUPPLY { parameter = value } TO CONTEXT context.
```

Dieses Kommando füllt die Schlüsselfelder eines Datenkontextes mit Werten.

```
SUPPRESS DIALOG.
```

Diese Anweisung ist nur in einem PBO-Modul eines Dynpros sinnvoll. Sie unterdrückt die Ausgabe des Dynpros auf dem Bildschirm und verhindert somit eine Dateneingabe durch den Anwender. Die Kommandos der Ablauflogik werden jedoch ausgeführt.

```
TABLES table.
```

Die angegebene Tabelle oder View wird im Programm deklariert. Dabei wird eine gleichnamige Feldleiste als Arbeitsbereich (Kopfzeile) angelegt. Das angegebene Objekt muss im Dictionary vorhanden und aktiv sein.

```
TOP-OF-PAGE [DURING LINE-SELECTION].
```

Diese Zeitpunktanweisung wird im Reporting zu Beginn der Ausgabe einer neuen Seite prozessiert. In der Grundform ist die Anweisung nur für die Grundliste (1. List-Ebene) zuständig. Für die Seitenköpfe aller Verzweigungslisten wird die Anweisung TOP-OF-PAGE DURING LINE-SELECTION prozessiert.

```
TRANSFER field TO filename [LENGTH length].
```

Der Inhalt von field wird in die angegebene Datei geschrieben. Sollte die Datei noch nicht existieren, versucht das TRANSFER-Kommando, sie anzulegen. Die Form der Ausgabe hängt davon ab, in welchem Modus die Datei erzeugt wurde (siehe OPEN). Durch die optionale Angabe einer Ausgabelänge kann die Zahl der zu schreibenden Zeichen begrenzt werden.

```
TRANSLATE string
  TO UPPER CASE |
  TO LOWER CASE |
```

```
  USING pattern |
  FROM CODE PAGE codepage_1 TO CODE PAGE codepage_2 |
  FROM NUMBER FORMAT format_1 TO NUMBER FORMAT format_2.
```

Der Inhalt der Zeichenkette wird transformiert, das Ergebnis steht dann wieder in `string`. Neben der Wandlung in Groß- oder Kleinbuchstaben können mit `USING` einzelne Zeichen in andere umgesetzt werden. Die Umsetzung wird in `pattern` definiert. In diesem Parameter (Feld oder Direktwert) stehen das zu ersetzende und das neue Zeichen paarweise hintereinander.

Mit dem Zusatz `FROM` bzw. `TO CODEPAGE` erfolgt die Transformation anhand einer Umsetztabelle, in der verschiedene Zeichensätze bzw. Codetabellen mit der Transaktion `SPAD` gepflegt werden können. Die Zusätze `FROM` bzw. `TO NUMBER FORMAT` wandeln Zahlen zwischen verschiedenen systemabhängigen Darstellungen um.

```
TRY.
```

Dieses Kommando leitet einen Anweisungsblock ein. Innerhalb dieses Blocks können Ausnahmen durch das Kommando `CATCH` abgefangen und behandelt werden.

```
TYPES typ.
```

Die `TYPES`-Anweisung definiert benutzereigene Datentypen. Ihre Parameter entsprechen der `DATA`-Anweisung, es können also Felder, Feldleisten und interne Tabellen definiert werden. Um die mit `TYPES` definierten Typen verwenden zu können, muss ein Datenfeld mit `DATA ... TYPE` angelegt werden.

```
TYPE-POOL typ_pool.
```

Diese Anweisung leitet einen Typ-Pool ein. Sie muss normalerweise nie manuell notiert werden, da sie durch das System automatisch in einen Typ-Pool eingefügt wird.

```
TYPE-POOLS typ_pool.
```

Mit dieser Anweisung wird ein Typ-Pool in eine Anwendung eingebunden. Danach können die im Typ-Pool definierten Datentypen benutzt werden.

```
ULINE [AT [/position(length)]] [NO GAP].
```

In der Grundform erzeugt diese Anweisung eine durchgehende Strichzeile. Mit einer Formatangabe kann sowohl die Anfangsposition als auch die Länge der Strichausgabe vorgegeben werden. Der Zusatz `NO GAP` unterdrückt die Ausgabe von Leerzeichen nach dem Strich.

```
UNASSIGN <fieldsymbol>.
```

Aufheben der Zuweisung des Feldsymbols. Dieses Feldsymbol zeigt nun auf kein gültiges Element.

```
UNPACK source TO target.
```

Der gepackte Inhalt des Quellfelds wird entpackt und in das Zielfeld gestellt. Dabei wird, je nach Länge von target, entweder mit führenden Nullen aufgefüllt oder abgeschnitten.

```
UPDATE table SET { assignment }
  [WHERE condition]
  [CLIENT SPECIFIED].
```

Mit dieser Anweisung werden einzelne Felder einer Tabelle aktualisiert. Die Zuweisung des neuen Wertes erfolgt in der Form table_field = value. Dabei kann der Wert ein Feld oder das Ergebnis eines arithmetischen Ausdrucks sein, in dem aber nur Addition oder Subtraktion möglich sind. Mit der WHERE-Klausel kann die Menge der zu aktualisierenden Datensätze eingeschränkt werden. Ohne WHERE-Klausel werden alle Datensätze modifiziert. Mit dem Zusatz CLIENT SPECIFIED wird das automatische Mandantenhandling abgeschaltet, das Mandantenfeld muss dann manuell mit einem korrekten Suchmuster versorgt werden. In einer UPDATE-Anweisung können mehrere Teilanweisungen per Doppelpunktmechanismus verknüpft werden. Dabei gelten der WHERE- und der CLIENT-Zusatz jeweils nur für eine der Teilanweisungen.

```
UPDATE table | *table | (field_with_tablename)
  [FROM record] | [FROM TABLE itab]
  [CLIENT SPECIFIED].
```

Diese Anweisung modifiziert in ihrer Grundform (ohne FROM TABLE-Zusatz) genau einen Datensatz der Datenbanktabelle. Die neuen Daten einschließlich der Schlüsselfelder, die den zu modifizierenden Datensatz eindeutig identifizieren müssen, werden der Kopfzeile oder dem angegebenen Arbeitsbereich entnommen. Mit dem Zusatz CLIENT SPECIFIED wird das automatische Mandantenhandling abgeschaltet.

Die neuen Daten können auch einer internen Tabelle entnommen werden, die mit dem Zusatz FROM TABLE in die UPDATE-Anweisung aufgenommen wird. In diesem Fall findet ein Massen-Update der Datenbanktabelle statt.

```
WAIT [UNTIL condition] UP TO n SECONDS.
```

Die Programmabarbeitung wird für n Sekunden unterbrochen. Falls eine Bedingung angegeben wurde und diese nicht erfüllt ist, werden von einem zuvor aufgerufenen asynchronen RFCs entgegengenommen, bis die Bedingung erfüllt ist, keine asynchronen Aufrufe mehr existieren oder die angegebene Wartezeit abgelaufen ist. Bei der Angabe einer Bedingung ist der UP TO-Zusatz optional.

```
WHILE condition
  [VARY field FROM field_1 NEXT field_2].
```

Die zwischen WHILE und ENDWHILE stehenden Anweisungen werden so lange ausgeführt, wie die Auswertung des logischen Ausdrucks den Wert für TRUE ergibt. Im logischen Ausdruck sind dieselben Anweisungen möglich wie bei IF.

Der Zusatz VARY ermöglicht in der WHILE-Schleife eine Wertzuweisung zu field. Im ersten Durchlauf erhält Feld den Wert von field_1, im zweiten den von field_2. In den weiteren Durchläufen wird der neue Feldinhalt dem Feld entnommen, dessen Adresse sich durch Addition des Abstands von field_1 und field_2 zum jeweils letzten Feld ergibt.

```
WINDOW STARTING AT x1 y1 [ENDING AT x2 y2].
```

Die aktuelle Liste wird in einem Popup angezeigt, die Werte x1 und y1 geben die linke obere Ecke des Popups innerhalb des aktuellen Fensters an. Die rechte untere Ecke ist identisch mit der des aktuellen Fensters oder wird durch den Zusatz ENDING AT angegeben.

```
WRITE [AT /position(output_length)] field
  [options]
  [format]
  [AS CHECKBOX] |
  [AS SYMBOL]
  [AS ICON] |
  [AS LINE]
  [QUICKINFO info].
```

Die Anweisung WRITE ist nur in der Listenaufbereitung sinnvoll. Sie schreibt den Inhalt eines Feldes in die Ausgabeliste. Dabei erfolgt die Ausgabe entsprechend dem Typ des Feldes in der standardmäßigen Ausgabelänge. Die Ausgabe kann mit dem Zusatz AT innerhalb der Zeile positioniert und in ihrer Länge begrenzt werden. Eine Vielzahl von Zusätzen und Parametern erlaubt die Modifikation des Ausgabebilds. Die Aufbereitungsoptionen (siehe Tabelle 9.12) beeinflussen die Darstellung des Wertes, während über die Ausgabeformate das Setzen von Farbe und Intensität möglich ist. Diese Ausgabeformate entsprechen den Parametern des Kommandos FORMAT. Mit einigen anderen Zusätzen kann die Darstellung spezieller Felder als Checkbox erfolgen. Icons oder Symbole werden im System durch spezielle Zeichenfolgen identifiziert, also nicht zwangsweise als Symbol erkannt. Die Zusätze AS ICON bzw. AS SYMBOL erzwingen die Ausgabe des zum Feldinhalt passenden Symbols. Passende Konstanten sind in den Includes <ICON>, <SYMBOL> bzw. <LIST> definiert.

Mittels des Zusatzes QUICKINFO kann ein bis zu 40 Zeichen langer Text info definiert werden, der in einer speziellen Darstellung angezeigt wird, wenn der Mauszeiger über dem betreffenden Feld verharrt.

Option	Wirkung
NO-ZERO	Führende Nullen unterdrücken.
NO-SIGN	Kein Vorzeichen ausgeben.
NO-GROUPING	Unterdrücken der Tausender-Trennzeichen.
DD/MM/YY	Datumsdarstellung nach Schablone.
MM/DD/YY	Datumsdarstellung nach Schablone.
DD/MM/YYYY	Datumsdarstellung nach Schablone.
MM/DD/YYYY	Datumsdarstellung nach Schablone.
DDMMYY	Datumsdarstellung nach Schablone.
MMDDYY	Datumsdarstellung nach Schablone.
YYMMDD	Datumsdarstellung nach Schablone.
CURRENCY	Währungsgerechte Aufbereitung.
DECIMALS	Auszugebende Dezimalstellen.
ROUND	Runden eines P-Feldes.
UNIT	Aufbereitung entsprechend Maßeinheit.
EXPONENT	Exponentenschreibweise mit vorgegebenem Exponenten.
USING EDIT MASK	Formatierungsmaske benutzen.
USING NO EDIT MASK	Konvertierungsroutine aus Dictionary nicht ausführen.
UNDER	Ausgabe exakt unter einem anderen Feld.
NO-GAP	Trennzeichen (Leerzeichen) nach Ausgabe unterdrücken.
LEFT-JUSTIFIED	Im Ausgabefeld links ausrichten.
CENTERED	Im Ausgabefeld zentrieren.
RIGHT-JUSTIFIED	Im Ausgabefeld rechts ausrichten.
TIME ZONE	Ausgabe als Zeitstempel, bezogen auf die angegebene Zeitzone.

Tabelle 9.12
Aufbereitungsoptionen zum WRITE-Kommando

```
WRITE field | (field_name) TO target[+offset(length)]
  [options]
  [INDEX index].
```

Die WRITE TO-Anweisung schreibt ihre Ausgabe nicht in die Bildschirmliste, sondern in ein anderes Feld. Dabei können für das Ziel auch ein Offset innerhalb des Feldes und die Länge des zu überschreibenden Bereiches angegeben werden. Die zu übertragenden Zeichen können mit einigen der Aufbereitungsoptionen aus Tabelle 8.12 bearbeitet werden. Es ist auch möglich, direkt in einen Datensatz einer internen Tabelle zu schreiben, der über die Datensatznummer identifiziert werden muss.

9.4 Kommandoübersicht Ablauflogik

Neben den echten ABAP-Kommandos existieren einige weitere für die Ablauflogik der Dynpros. Die Syntax dieser Kommandos ähnelt der einiger ABAP-Kommandos, die Funktionalität weicht aber stark von diesen ab.

```
CALL SUBSCREEN subscreen
  [ INCLUDING program screen_number ].
```

Mit diesem Kommando wird die Ablauflogik eines Subscreens aufgerufen. Der Subscreen kann auch in einem anderen Modulpool definiert sein. Der Name des Modulpools und die Dynpro-Nummer können sowohl als Konstante als auch dynamisch in einem Datenfeld übergeben werden. Der Zusatz INCLUDING ist nur zum Zeitpunkt PBO erforderlich.

```
CALL CUSTOMER-SUBSCREEN subscreen
  [ INCLUDING program screen_number ].
```

Mit diesem Kommando wird ein Subscreen aufgerufen, der vom Kunden angelegt werden kann. Programmname und Dynpro-Nummer müssen als Konstante notiert werden. Der Subscreen muss in einem mit den Zeichen SAPLX beginnenden Programm liegen und vom Kunden mittels der Transaktion CMOD als Kundenerweiterung registriert werden.

```
CHAIN.
  flow logic statements
ENDCHAIN.
```

Die beiden Anweisungen CHAIN und ENDCHAIN fassen Anweisungen zu einem Block zusammen. Alle FIELD-Anweisungen in diesem Block gelten für alle nachfolgenden MODULE-Anweisungen. Wenn eines der Module eine Nachricht auslöst, werden alle im Block mit FIELD selektierten Eingabefelder eingabebereit.

```
FIELD fieldlist [:]
  [ MODULE modul ] |
  [ SELECT statement ] |
  [ VALUES value_list ].
```

Die FIELD-Anweisung benennt ein oder mehrere Dynpro-Felder, auf die verschiedene Prüfroutinen angewendet werden. Die Prüfroutinen können entweder direkt in der FIELD-Anweisung stehen oder durch eine CHAIN-ENDCHAIN-Klammer mit den Prüfkommandos zusammengefasst werden. Bei der ersten Variante darf jeweils nur ein Feld angegeben werden. Die zweite Variante gestattet die Aufzählung mehrerer Felder mit Hilfe der Doppelpunkt-Schreibweise.

Wenn eine der Prüfroutinen eine Nachricht vom Typ E auslöst, werden nur das oder die Dynpro-Felder eingabebereit, die in der Feldliste aufgeführt wurden.

```
LOOP [ WITH CONTROL control ].
  flow logic statements
ENDLOOP.
```

Diese Anweisung verarbeitet nacheinander alle Blöcke einer Step Loop oder eines Table View. Im letzten Fall ist die Angabe eines Controls durch den entsprechenden Zusatz erforderlich.

```
LOOP AT itab
 [[ FROM first ] [ TO last ]] |
 [ CURSOR cursor ]            |
 [ WITH CONTROL control ]     |
 [ INTO work_area ].
   flow logic statements
ENDLOOP.
```

Der Inhalt der internen Tabelle itab wird in einer Step Loop oder einem Table View bearbeitet. Dabei sorgt das System ggf. automatisch für die Realisierung einer Blätterfunktion. Falls ein Table View benutzt wird, ist die Angabe eines entsprechenden Controls erforderlich. Mit den beiden Optionen FROM und TO kann der angezeigte Bereich der Tabelle eingeschränkt werden. Die Option INTO ermöglicht die Benutzung eines separaten Arbeitsbereichs anstelle der Kopfzeile der Tabelle. Ein durch die Option CURSOR benanntes Feld erfüllt eine Doppelfunktion. Wird es dynamisch im Programm belegt, erfolgt im Dynpro die Anzeige ab der entsprechenden Tabellenzeile. Damit kann eine programmgesteuerte Blätterfunktion realisiert werden. Innerhalb der LOOP-Anweisung enthält das Feld die Nummer der jeweils aktuellen Tabellenzeile.

```
LOOP AT table.
  flow logic statements
ENDLOOP.
```

Mit dieser Anweisung wird der Inhalt einer Datenbanktabelle satzweise in eine Step Loop übertragen. Das Zurückschreiben der geänderten Werte in die Tabelle erfordert den Einsatz des Kommandos MODIFY zum Zeitpunkt PAI. Dieses Kommando wurde insbesondere für die Realisierung der Standard-Tabellenpflege (Transaktion SM30) eingesetzt. Ab Release 3.0 stehen für die Tabellenpflege andere Werkzeuge zur Verfügung. Diese Variante der LOOP-Anweisung wird daher nur noch aus Kompatibilitätsgründen unterstützt.

```
MODIFY dictionary_table.
```

Dieses Kommando aktualisiert eine Datenbanktabelle innerhalb einer LOOP-Schleife über derselben. Außerdem behandelt es einige Aktionen, die bei der Standard-Tabellenpflege auftreten.

```
MODULE modul_name
  [ON INPUT]
  [ON CHAIN-INPUT]
  [ON *-INPUT]
  [ON REQUEST]
  [ON CHAIN-REQUEST]
  [AT CURSOR-SELECTION]
  [AT EXIT-COMMAND].
```

In der Ablauflogik eines Dynpros wird mit der Anweisung MODULE ein Modul des Modulpools aufgerufen. Die verschiedenen Zusätze, die nur zum Zeitpunkt PAI benutzt werden können, machen die Ausführung des Moduls von verschiedenen Vorbedingungen abhängig. Alle ON-Zusätze können darüber hinaus nur in Verbindung mit einer Feldzuordnung (Anweisungen FIELD und/oder CHAIN) genutzt werden.

Tabelle 9.13 erläutert die Funktion der verschiedenen Zusätze.

Zusatz	Modul wird ausgeführt, wenn
ON INPUT	im zugeordneten Eingabefeld ein Wert ungleich dem Initialwert steht.
ON CHAIN-INPUT	in mindestens einem Feld der CHAIN-Klammer ein Wert ungleich dem Initialwert steht.
ON *-INPUT	das Eingabefeld die Eigenschaft *-Eingabe hat und ein Stern eingegeben wurde.
ON REQUEST	im zugeordneten Eingabefeld eine Eingabe erfolgte.
ON CHAIN-REQUEST	in mindestens einem Feld der CHAIN-Klammer eine Eingabe erfolgte.
AT CURSOR-SELECTION	der Anwender durch einen Doppelklick mit der Maus oder die Funktionstaste F2 ein Objekt ausgewählt hat.
AT EXIT-COMMAND	im Dynpro ein Funktionscode ausgelöst wurde, der den Funktionstyp E (Exit-Code) hat. Das Modul wird vor allen anderen Modulen und vor der automatischen Wertprüfung ausgeführt.

Tabelle 9.13
Optionen beim Aufruf von Modulen

```
PROCESS (
  BEFORE OUTPUT   |
  AFTER INPUT     |
  ON HELP-REQUEST |
  ON VALUE-REQUEST ).
```

Diese Anweisung leitet jeweils einen eigenständigen Abschnitt in der Ablauflogik eines Dynpros ein. Es muss zwingend einer der Zusätze benutzt werden. Jeder Abschnitt entspricht einem bestimmten Ereignis bei der Verarbeitung eines Dynpros. Tabelle 9.14 beschreibt die verschiedenen Zeitpunkte.

Zeitpunkt	Abkürzung	Zeitpunkt der Ausführung
PROCESS BEFORE OUTPUT	PBO	Vor Anzeige des Dynpros.
PROCESS AFTER INPUT	PAI	Nach Eingabe von Daten im Dynpro.
PROCESS ON HELP-REQUEST	POH	Bei Betätigung der F1-Taste im Dynpro (allgemeine Hilfe).
PROCESS ON VALUE-REQUEST	POV	Bei Anforderung einer Werthilfe (F4-Taste).

Tabelle 9.14
Zeitpunkte der Ablauflogik

```
SELECT * FROM tabele
  WHERE keyfield = screen_field
    [ AND keyfield = screen_field ...]
  [ INTO field]
  [ WHENEVER ( NOT FOUND | FOUND )
      SEND ( ERRORMESSAGE | WARNING )
      [ message_number ]
      [ WITH field ... ]].
```

Dieses Kommando liest einen Datensatz aus einer Datenbanktabelle. Dies kann sowohl zum Zeitpunkt PBO erfolgen, um Dynpro-Felder zu initialisieren, oder aber zum Zeitpunkt PAI, um Werte in Dynpro-Feldern zu prüfen. Im ersten Fall erfolgt mit dem Zusatz `INTO` die Übertragung des Wertes in das Dynpro-Feld. Das Ergebnis der Selektion kann durch weitere Zusätze ausgewertet werden, um Fehlernachrichten zu erzeugen.

9.5 Systemfelder

Das System füllt die Systemfelder zur Laufzeit mit aktuellen Werten. Einige dieser Felder sind innerhalb der Anwendungen von erheblicher Bedeutung, da sie

Auskunft über den Systemstatus liefern. Andere dienen nur zur Kommunikation diverser Basisprogramme untereinander und sind für Anwendungsentwickler uninteressant. Dieser Abschnitt erläutert die Bedeutung der wichtigsten Systemfelder, geordnet nach Themenbereichen und Hinweisen zur Verwendung.

Beim Debuggen einer Anwendung können die Systemfelder entweder einzeln durch Eingabe ihres kompletten Namens oder zusammen als Liste angezeigt werden. Letztere Variante erfordert die Auswahl des Debugger-Modus zur Feldanzeige (SPRINGEN -> FELDER *Goto -> Fields*). Als Feldname wird `SY` oder `SYST` eingetragen. Nach Betätigen der Datenfreigabetaste erscheinen alle Systemfelder. Sie werden intern als Elemente der Feldleiste `SY` bzw. `SYST` verwaltet. Beide Namen sind synonym.

9.5.1 Allgemein

In diese Gruppe wurden Felder aufgenommen, die sich nicht eindeutig einem der anderen Themenbereiche zuordnen lassen. Das bedeutet nicht, dass diese Felder unwichtig sind. Bei einigen der Felder dieser Gruppe handelt es sich um die meistverwendeten Systemfelder überhaupt!

SY-ABCDE

Dieses Feld enthält das verwendete Alphabet in Großbuchstaben. Es kann zur Überprüfung von korrekten Eingaben benutzt werden, was in der Praxis aber selten auf diese Weise erfolgt.

SY-DATAR

Dieses Feld wird vor jeder Ausführung eines Dynpros initialisiert und nach der Bearbeitung, aber vor Ausführung des ersten PAI-Moduls, auf „X“ gesetzt, falls im Dynpro Daten geändert wurden. Auf diese Weise können in Exit-Modulen Datenänderungen erkannt und gegebenenfalls Sicherheitsabfragen ausgeführt werden.

SY-DBCNT

Dieses Feld enthält nach einer Datenbankoperation die Anzahl der bearbeiteten bzw. gefundenen Elemente. Dieses Feld wird beispielsweise oft benutzt, um in Verbindung mit einer `SELECT`-Anweisung festzustellen, ob zu einem bestimmten Suchbegriff 0, 1 oder mehrere Datensätze existieren. Dies kann auch mit dem Zusatz `COUNT` in der `SELECT`-Anweisung erfolgen, der sein Ergebnis ebenfalls in dieser Systemvariable stellt.

SY-FDPOS

Verschiedene Kommandos zur Zeichenkettenverarbeitung und Operatoren in logischen Anweisungen (z.B. `CP`, `CS` etc.) belegen dieses Feld mit der Position der gefundenen Zeichenkette. Der Feldinhalt von `SY-FDPOS` kann genutzt werden, um Ersetzungen in der Zeichenkette durchzuführen oder einen Teilstring herauszulösen.

SY-LANGU

Dieses Feld wird bei der Anmeldung eines Anwenders mit dem Kürzel für die aktuelle Sprache gesetzt. Anhand dieses Feldes ermitteln viele systeminterne Hilfsmittel die korrekten sprachabhängigen Texte und Nachrichten. Dieses Feld müssen Sie in Anwendungen auswerten, falls diese selbst sprachabhängige Tabellen anlegen, die z.B. verbale Beschreibungen zu einem Datensatz enthalten.

SY-MANDT

Dieses Feld wird bei der Anwendung mit der Nummer des aktuellen Mandanten gefüllt. Eine Auswertung ist selten erforderlich, da Mandantenfelder durch das System automatisch behandelt werden. Es kann aber Sinn machen, beispielsweise während der Testphase, spezielle Anwendungen durch Abfrage dieses Feldes in bestimmten Mandanten zu sperren oder freizugeben.

SY-SUBRC

Dies ist das am häufigsten benutzte Systemfeld. Es enthält nach Ausführung vieler Anweisungen einen Wert, der Auskunft über die korrekte Ausführung dieses Kommandos gibt. Beispiele dafür sind die OPEN-SQL-Anweisungen und der Aufruf von Funktionsbausteinen.

SY-UNAME

In dieses Feld wird bei der Anmeldung am System der Anmeldename des Anwenders eingetragen. Dieser Name wird beispielsweise von der Berechtigungsprüfung benutzt, um die aktuellen Berechtigungsprofile des Anwenders zu ermitteln. In Anwendungen wird das Feld oft benutzt, um den Namen des letzen Änderers des entsprechenden Datensatzes zu speichern.

9.5.2 Ablaufsteuerung

SY-DYNGR

In den Attributen eines Dynpros kann ein Wert für eine so genannte Dynpro-Gruppe gepflegt werden. Dieses Attribut ist optional. Es wird relativ selten be-

nutzt. Während der Abarbeitung eines Dynpros ist der Wert des Attributs Dynpro-Gruppe im Feld SY-DYNGR enthalten.

SY-DYNNR

Während der Abarbeitung eines Dynpros enthält dieses Feld die Nummer des aktuellen Dynpros. Dieser Wert wird relativ häufig benutzt, um in Modulen, die von mehreren Dynpros benutzt werden, die konkret auf ein spezielles Dynpro bezogenen Anweisungen auszuführen. Dies kann beispielsweise das Setzen eines Status oder das von der Dynpro-Nummer abhängige Deaktivieren von Funktionscodes sein.

SY-INDEX

In diesem Feld werden Schleifendurchläufe (LOOP, DO, SELECT...) gezählt. Der Wert wird mitunter zur Erzeugung einer laufenden Nummer oder zur einfachen Zählung von Aktionen in einer Schleife ausgeführt. Dieser Wert ist nur innerhalb der jeweiligen Schleife aktuell; außerhalb der Schleife ist er undefiniert.

SY-LOOPC

Dieses Feld enthält die Anzahl der auf dem Dynpro sichtbaren Zeilen eines Step Loop. Der Inhalt ist nur innerhalb des Geltungsbereiches einer LOOP-ENDLOOP-Schleife in der Ablauflogik (also auch innerhalb der von dort aus aufgerufenen Module und Unterprogramme) gültig.

SY-PFKEY

Dieses Feld enthält den Namen des aktuellen Status der Oberfläche. Eine Auswertung ist selten erforderlich.

SY-REPID

In diesem Feld wird der Name des aktuellen Programms aufbewahrt.

SY-STEPL

Innerhalb einer LOOP-ENDLOOP-Schleife der Ablauflogik wird die Nummer der momentan bearbeiteten Zeile des Step Loop in dieses Feld gestellt. Der Wert ist, ebenso wie SY-LOOPC, nur innerhalb der Schleife gültig.

SY-TCODE

Bei Ausführung einer Transaktion wird der Transaktionscode in dieses Feld gestellt. Da Anwendungen oft mit verschiedenen Transaktionen aufgerufen werden, dient dieses Feld sehr häufig zur Auswahl einer konkreten Aktion inner-

halb einer Anwendung (z.B. Anlegen, Ändern, Anzeigen) oder der Auswahl eines Wertebereichs für die Berechtigungsprüfung.

SY-UCOMM

Beim Auslösen eines Funktionscodes wird dieser in das Feld `SY-UCOMM` gestellt. Da der Funktionscode auch über andere Felder, beispielsweise das OK-Code-Feld eines Dynpros, ausgewertet wird, erfolgt ein lesender Zugriff auf `SY-UCOMM` nicht besonders oft. Es ist allerdings üblich, die programminternen Datenfelder, die einen Funktionscode aufnehmen sollen, per `LIKE` von `SY-UCOMM` abzuleiten.

9.5.3 Interne Tabellen

Auch bei der Bearbeitung von internen Tabellen werden einige Systemfelder gesetzt. Diese Felder geben vor allem Auskunft über Größe und Zustand der Tabelle. Zwei dieser Felder werden häufiger eingesetzt:

SY-TABIX

Dieses Feld enthält die Nummer des aktuellen Datensatzes der zuletzt bearbeiteten internen Tabelle. Es wird beispielsweise aktualisiert, wenn ein Datensatz mit der Anweisung `READ` gesucht wird. Soll dieser Datensatz nach einer Modifikation wieder in die Tabelle zurückgeschrieben werden, so erfordert das Kommando `MODIFY` die Angabe der Datensatznummer, die `SY-TABIX` entnommen werden kann.

SY-TFILL

Dieses Feld ist nur innerhalb einer `LOOP`-Schleife über eine interne Tabelle aktuell. Es enthält die Anzahl der Datensätze der internen Tabelle.

9.5.4 Texte und Nachrichten

Über spezielle Systemfelder kann auf Texte bzw. Eigenschaften von Nachrichten zugegriffen werden. Dies erfolgt in Anwendungen relativ selten. Allerdings kann die Verwendung dieser Felder beim Debuggen einer Anwendung mitunter recht nützlich sein.

SY-MSGID

Dieses Feld enthält die Message-ID der zuletzt ausgelösten Nachricht.

SY-MSGTY

Dieses Feld enthält den Typ (E, I, W, S, A) der zuletzt ausgelösten Nachricht.

SY-MSGNO

Dieses Feld enthält die Nummer der zuletzt ausgelösten Nachricht.

SY-MSGV1-4

Diese vier Felder enthalten die Zeichenketten, mit denen die Platzhalter eines Nachrichtentextes ersetzt wurden.

SY-ULINE

In diesem Feld legt das System das für Unterstreichungen benutzte Zeichen ab.

SY-VLINE

In diesem Feld wird das für vertikale Striche verwendete Zeichen gespeichert.

SY-TITLE

Dieses Feld enthält den Titel einer Anwendung.

9.5.5 Zeit und Datum

Datums- und Zeitangaben werden sehr häufig benötigt, beispielsweise um den Änderungszeitpunkt von Datensätzen protokollieren zu können. Dementsprechend häufig werden einige dieser Felder benutzt.

SY-TZONE

In diesem Feld wird die Differenz in Sekunden zwischen der aktuellen und der „Greenwich Mean Time“ (UTC) abgelegt.

SY-DAYST

Dieses Feld ist gesetzt, wenn die Sommerzeit aktiv ist.

SY-FDAYW

Dieses Feld enthält die Nummer des Wochentags gemäß Fabrikkalender.

SY-DATUM

Dieses Feld nimmt das aktuelle Datum auf.

SY-UZEIT

In diesem Feld wird vom System die aktuelle Uhrzeit gespeichert.

9.5.6 Bildschirm- und Listengestaltung

In der Regel reichen die durch das System bereitgestellten Hilfsmittel zur Gestaltung der Ausgabe aus. Damit diese Hilfsmittel korrekt arbeiten können, benötigen sie allerdings einige Informationen zum aktuellen Zustand der Liste oder des Dynpros. Diese Werte werden in Systemfeldern aufbewahrt und können in Ausnahmefällen auch für eigene Anwendungen interessant sein.

SY-COLNO

In diesem Feld wird bei der Ausgabe einer Liste die Spalte (Zeichenposition innerhalb einer Zeile) aufbewahrt.

SY-CPAGE

Aktuelle Seitennummer.

SY-CUCOL, SY-CUROW

Aktuelle Cursorposition.

SY-LINCT

Das Feld SY-LINCT enthält die Anzahl der Zeilen einer Liste.

SY-LINNO

Dieses Feld enthält während der Erstellung einer Ausgabeliste die aktuelle Zeilennummer.

SY-LINSZ

In diesem Feld bewahrt das System die nutzbare Zeilenbreite der Liste auf. Dieser Wert ergibt sich entweder aus der mit `LINE-SIZE` gesetzten Breite der Liste oder der aktuellen Fenstergröße, falls kein Wert für die Seitenbreite festgelegt wurde. Es ist denkbar, die Formatierung einer Liste an den zur Verfügung stehenden Platz anzupassen, also die Ausgabelänge sehr breiter Felder einzuschränken oder Felder ganz auszublenden. Dies ist aber nur sinnvoll, wenn die Liste vorrangig zur Ausgabe auf den Bildschirm vorgesehen ist, da die Formatierung bei Drucklisten sehr detailliert vorgegeben wird.

SY-MACOL, SY- MAROW

Mit der `SET MARGIN`-Anweisung kann auf der linken Seite und am oberen Rand ein freier Bereich definiert werden, der nicht bedruckt wird. Die Abmessungen dieses Bereiches (linker und oberer Rand) sind über die beiden Felder verfügbar.

SY-PAGNO

Während der Ausgabe einer Liste enthält dieses Feld die Nummer der aktuellen Seite.

SY-SCOLS, SY-SROWS

In diesen beiden Feldern bewahrt das System die Abmessungen des Bildschirms auf. Diese Werte werden aktualisiert, wenn der Anwender die Größe des Bildschirms ändert. Sie können benutzt werden, um den Aufbau einer Liste an die Bildschirmgröße anzupassen.

9.5.7 Window

Die Felder der Window-Gruppe liefern Informationen über Größe und Position eines Fensters sowie die Position des Cursors.

SY-WINX1, SY-WINY1

Linke obere Ecke des Windows, relativ zum übergeordneten Fenster.

SY-WINX2, SY- WINY2

Rechte untere Ecke des Windows, relativ zum übergeordneten Fenster.

SY-WINCO, SY-WINRO

Position des Cursors im Window.

9.5.8 Interaktives Reporting

Beim interaktiven Reporting erfolgt die Übermittlung von Informationen nicht nur über den verborgenen Datenbereich (siehe Kommando `HIDE`), sondern auch über einige Systemfelder. Allerdings liefern diese Felder lediglich ergänzende Informationen, die nur in Sonderfällen sinnvoll benutzt werden können.

SY-LILLI

Nach einer Zeilenselektion enthält dieses Feld die absolute Nummer der selektierten Zeile der Liste.

SY-LISEL

Nach einer Zeilenselektion steht in diesem Feld der Inhalt der ausgewählten Zeile. Da die Zuordnung zu den einzelnen Feldern verloren geht, kann der Inhalt dieses Feldes nur selten sinnvoll ausgewertet werden.

SY-LSIND

Dieses Feld ist von erheblicher Bedeutung. Es enthält die List-Stufe. Die Grundliste besitzt den Wert 0, die erste Verzweigungsliste den Wert 1 usw. Dieses Feld wird zu den diversen AT-Zeitpunkten benutzt, um den für die aufzubauende Liste erforderlichen Programmzweig auszuwählen.

SY-STACO, SY-STARO

Diese beiden Felder enthalten die Koordinaten des ersten angezeigten Zeichens der Liste – oder anders ausgedrückt: die Nummer der ersten Zeile im Fenster und die Nummer der ersten angezeigten Spalte. Diese Felder haben im Grundzustand den Wert 1. Der Wert ändert sich, wenn in der Liste gerollt wird. Wenn in weitere Listen verzweigt wird, dann enthalten diese Felder die entsprechenden Werte für die letzte angezeigte und nicht für die aktuelle Liste.

9.5.9 Systembezogene Felder

Diese Felder, die Informationen über das aktuelle R/3-System liefern, sind für die meisten Anwendungen uninteressant. Eine Auswertung ist nur in sehr speziellen Fällen erforderlich, z.B. beim Einsatz von Native-SQL-Anweisungen.

SY-DBSYS

Dieses Feld enthält die Bezeichnung des Datenbanksystems.

SY-HOST

Dieses Feld enthält den Namen des Rechners, auf dem das R/3-System läuft. Der Name ist identisch mit dem Rechnernamen, der in der Statuszeile angezeigt wird.

SY-OPSYS

In diesem Feld wird der Name des Betriebssystems abgelegt.

SY-SAPRL

Das Feld SY-SAPRL enthält die Bezeichnung des Releasestands des Systems.

SY-SYSID

Dieses Feld liefert den Namen des R/3-Systems. Auch dieser Wert wird in der Statuszeile angezeigt.

Listings

10

Nachfolgend finden Sie die in Kapitel 7 erwähnten Listings zur zweiten Beispielanwendung. Im zweiten Dynpro dient ein Table View zur Pflege der Kurse. Dies erspart im Vergleich zu Step Loops einiges an Programmieraufwand. Der Einfachheit halber sind auch in diesem Programm die Menüs relativ einfach gehalten. Im ersten Dynpro stehen nur die Standard-Funktionen zum Abbrechen, Beenden und Zurückspringen zur Verfügung. All diese Funktionen sind Exit-Funktionen. Im zweiten Dynpro kommt dann nur noch eine Funktion zum Speichern hinzu.

10.1 Ablauflogik Grundbild

```
PROCESS BEFORE OUTPUT.
  MODULE status_9100.
*
PROCESS AFTER INPUT.
  MODULE exit_9100 AT EXIT-COMMAND.
  MODULE read_data_9100.
  MODULE user_command_9100.
```

10.2 Ablauflogik Bearbeitungsbild

```
PROCESS BEFORE OUTPUT.
  MODULE status_9200.
  LOOP AT i_price INTO f_price CURSOR cursor WITH CONTROL tc_price.
  ENDLOOP.
*
PROCESS AFTER INPUT.
  MODULE exit_9200 AT EXIT-COMMAND.
  LOOP AT i_price.
    FIELD f_price-price
```

```
      MODULE data_changed_9200 ON REQUEST.
    FIELD f_price-price
      MODULE modify_itab_9200.
  ENDLOOP.
  MODULE user_command_9200.
```

10.3 Globale Deklarationen

```
PROGRAM  sapmyz3p MESSAGE-ID y3.
TABLES:
  yz3stock,
  yz3price
.                                       " tables

CONSTANTS:
  c_true           VALUE 'X',
  c_false          VALUE ' ',
  c_yes            VALUE 'J',
  c_no             VALUE 'N',
  c_cancel         VALUE 'A',
  c_on      TYPE i VALUE 1,
  c_off     TYPE i VALUE 0
.                                       " constants

DATA:
  fcode         LIKE sy-ucomm,
  g_fcode       LIKE fcode,
  g_date        LIKE yz3price-aktdate,
  cursor        TYPE i,
  g_changed     LIKE c_false VALUE c_false,
  g_save        LIKE c_false VALUE c_false,
  g_exit        LIKE c_false VALUE c_false,
  g_exit_module LIKE c_false VALUE c_false,

  i_stock       LIKE yz3stock OCCURS 20,
  f_stock       LIKE LINE OF i_stock,

  i_price       LIKE yz3price OCCURS 20,
  f_price       LIKE LINE OF i_price
.                                       " data

CONTROLS:
  tc_price TYPE TABLEVIEW USING SCREEN 9200
.                                       "controls
```

10.4 PBO-Module

```
MODULE status_9100 OUTPUT.
  SET PF-STATUS 'STAT_G'.
  yz3price-aktdat = sy-datum.
ENDMODULE.                              " STATUS_9100  OUTPUT

MODULE status_9200 OUTPUT.
  SET PF-STATUS 'STAT_A'.

*...globale Felder initialisieren
  g_exit        = c_false.
  g_exit_module = c_false.
ENDMODULE.                              " STATUS_9200  OUTPUT
```

10.5 PAI-Module

```
MODULE exit_9100 INPUT.
  LEAVE TO SCREEN 0.
ENDMODULE.                              " EXIT_9100  INPUT

MODULE read_data_9100 INPUT.
*...Itab für Kurse füllen
  PERFORM read_data_9100.
ENDMODULE.                              " READ_DATA_9100
INPUT

MODULE user_command_9100 INPUT.
  g_fcode = fcode.
  CLEAR fcode.

*...wenn ENTER-Taste gedrückt, dann Daten bearbeiten
  IF g_fcode = ' '.
    LEAVE TO SCREEN 9200.
  ENDIF.
ENDMODULE.                              " USER_COMMAND_9100
INPUT

MODULE exit_9200 INPUT.
  g_fcode = fcode.
  CLEAR fcode.
  g_exit_module = c_true.

*...wenn Daten geändert, dann Sicherheitsabfrage
  IF g_changed = c_true OR
     sy-datar  = c_true.
```

```
*...G_CHANGED setzen, damit auch im allerersten
*...Durchlauf korrekte Auswertung möglich
    g_changed = c_true.

*.....ggf. Sicherheitsabfragen durchführen
*.....und EXIT-Flag setzen
    PERFORM check_save_9200.

*.....wenn Beenden ohne Sichern gewünscht,
*.....dann Rücksprung direkt im EXIT-Modul
    IF g_exit = c_true AND
       g_save = c_false.
      g_changed = c_false.

*.......Satz entsperren, dann Rücksprung
      PERFORM dequeue_9200.
      PERFORM navigate_9200.
    ENDIF.
  ELSE.

*.....wenn keine Daten geändert
*.....G_EXIT setzen wegen Auswertung in NAVIGATE_9200
    g_exit = c_true.

*.....Satz entsperren, dann Rücksprung je nach Funktionscode
    PERFORM dequeue_9200.
    PERFORM navigate_9200.
  ENDIF.
ENDMODULE.                              " EXIT_9200  INPUT

MODULE D9200_DATA_CHANGED INPUT.
*...SY-DATAR wird immer neu initialisiert,
*...muß daher gesichert werden
  IF SY-DATAR = TRUE.
    G_CHANGED = TRUE.
  ENDIF.
ENDMODULE.

MODULE data_changed_9200 INPUT.
*...SY-DATAR wird immer neu initialisiert,
*...muß daher gesichert werden
  IF sy-datar = c_true.
    g_changed = c_true.
  ENDIF.
ENDMODULE.                              " DATA_CHANGED_9200
INPUT
```

```
MODULE modify_itab_9200 INPUT.
  IF f_price-price < 0.
    MESSAGE e010.                        " Korrekten Kurs
eingeben!
  ENDIF.

*...Itab aktualisieren
  MODIFY  i_price INDEX cursor FROM f_price.
ENDMODULE.                               " MODIFY_ITAB_9200
INPUT

MODULE user_command_9200 INPUT.
*...Bei FT03 und FT15 wird FCODE schon
*...im EXIT-Modul zurückgesetzt
  IF g_exit_module = c_false.
    g_fcode = fcode.
    CLEAR fcode.

*.....feststellen, ob Daten gesichert werden müssen
*.....und Dynpro verlassen werden darf. Abfrage nur
*.....erforderlich, wenn EXIT-Modul nicht wirksam war
    PERFORM check_save_9200.
  ENDIF.

  PERFORM save_9200.
  PERFORM navigate_9200.
ENDMODULE.                               " USER_COMMAND_9200
INPUT
```

10.6 Unterprogramme

```
FORM read_data_9100.
  RANGES r_wkz FOR yz3price-wkz.

*...Datum aus Dynpro 9100 sichern, da Kopfzeile
*...verändert wird
  g_date = yz3price-aktdate.
  CLEAR: i_stock, f_stock.
  REFRESH i_stock.

*...alle Aktien lesen und zwischenspeichern
  SELECT * FROM yz3stock
  INTO TABLE i_stock.

*...RANGES-Tabelle aufbauen
  r_wkz-sign = 'I'.
```

```
  r_wkz-option = 'EQ'.
  LOOP AT i_stock INTO f_stock.
    r_wkz-low = f_stock-wkz.
    APPEND r_wkz.
  ENDLOOP.

  CLEAR: i_price, f_price.
  REFRESH i_price.

*...Kurse für alle gültigen Aktien lesen
  SELECT * FROM yz3price
    INTO TABLE i_price
    WHERE aktdate = g_date AND
          wkz IN r_wkz.

*...Tabelle für Kurse sperren
  CALL FUNCTION 'ENQUEUE_EY3PRICE'
    EXPORTING
*     MODE_YZ3PRICE  = 'X'
*     MANDT          = SY-MANDT
      wkz            = ' '
      aktdate        = g_date
*     X_WKZ          = ' '
*     X_AKTDATE      = ' '
*     _SCOPE         = '2'
*     _WAIT          = ' '
*     _COLLECT       = ' '
    EXCEPTIONS
      foreign_lock   = 1
      system_failure = 2
      OTHERS         = 3.

  IF sy-subrc <> 0.
    MESSAGE e005.    " Satz kann nicht gesperrt werden!
  ENDIF.

  LOOP AT i_stock INTO f_stock.

*.....testen, ob für alle Sätze aus YZ3STOCK auch Einträge
*.....in YZ3PRICE existieren, wenn nein,
*.....dann einen Satz anfügen
    READ TABLE i_price INTO f_price
      WITH KEY wkz = f_stock-wkz.

    IF sy-subrc <> 0.
      CLEAR f_price.
      MOVE-CORRESPONDING f_stock TO f_price.
```

```
      f_price-aktdate = g_date.
      APPEND f_price TO i_price.

    ELSE.
      f_price-aktdate = g_date.
      MODIFY i_price INDEX sy-tabix FROM f_price.
    ENDIF.

  ENDLOOP.
ENDFORM.                               " READ_DATA_9100

FORM check_save_9200.
  CASE g_fcode.

*.....ENTER-Taste -> nur Datenprüfung
    WHEN ' '.
      g_save = c_false.
      g_exit = c_false.

*.....F3 = ZURÜCK
*.....-> Möglichkeit zum Sichern bei Datenänderung
    WHEN 'FT03'.
      PERFORM save_question_9200.

*.....F11 = SICHERN
*.....-> unbedingtes Sichern und Beenden des Dynpros
    WHEN 'FT11'.
      g_save = c_true.
      g_exit = c_true.

*.....F12 = ABBRECHEN
*.....-> nur Hinweis auf möglichen Datenverlust
    WHEN 'FT12'.
      PERFORM cancel_question_9200.

*.....F15 = BEENDEN
*.....-> Möglichkeit zum Sichern bei Datenänderung
    WHEN 'FT15'.
      PERFORM save_question_9200.

  ENDCASE.
ENDFORM.                               " CHECK_SAVE_9200

FORM dequeue_9200.
  CALL FUNCTION 'DEQUEUE_EY3PRICE'
    EXPORTING
*     MODE_YZ3PRICE = 'X'
*     MANDT         = SY-MANDT
```

```
      wkz           = ' '
      aktdate       = g_date
*     X_WKZ         = ' '
*     X_AKTDATE     = ' '
*     _SCOPE        = '3'
*     _SYNCHRON     = ' '
*     _COLLECT      = ' '
  .
ENDFORM.                                " DEQUEUE_9200

FORM navigate_9200.
  IF g_exit = c_true.
    CASE g_fcode.

*.......F3 = ZURÜCK
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT03'.
        LEAVE TO SCREEN 9100.

*.......F11 = SICHERN
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT11'.
        LEAVE TO SCREEN 9100.

*.......F12 = ABBRECHEN
*.......-> Rückkehr zum vorangegangenen Dynpro
      WHEN 'FT12'.
        LEAVE TO SCREEN 9100.

*.......F15 = BEENDEN
*.......-> Beenden der gesamten Anwendung
      WHEN 'FT15'.
        LEAVE PROGRAM.

    ENDCASE.
  ENDIF.
ENDFORM.                                " NAVIGATE_9200

FORM save_9200.
  IF g_save = c_true.
    MODIFY yz3price FROM TABLE i_price.

*.....Status der Datenbankoperation testen
    IF sy-subrc <> 0.
      ROLLBACK WORK.

*.......Text: Fehler beim Speichern!
      MESSAGE e013.
```

```
    ELSE.

*.......Text: Daten für & gesichert.
      MESSAGE s004 WITH g_date.
    ENDIF.

*.....Datensatzsperre aufheben
    PERFORM dequeue_9200.

*.....Änderungen festschreiben
    COMMIT WORK.

*.....Datenänderungsflag zurücksetzen
    g_changed = c_false.
  ENDIF.
ENDFORM.                                " SAVE_9200

FORM save_question_9200.
  DATA: l_answer.

*...Abfrage nur erforderlich, wenn Daten geändert wurden
  IF g_changed = c_true.
    CALL FUNCTION 'POPUP_TO_CONFIRM_STEP'
      EXPORTING
*         DEFAULTOPTION = 'Y'

*...........Text: Daten werden verloren gehen!
          textline1     = text-012

*...........Text: Vorher sichern?
          textline2     = text-013
          titel         = text-010
*         START_COLUMN  = 25
*         START_ROW     = 6
      IMPORTING
          answer        = l_answer
      EXCEPTIONS
          OTHERS        = 1.
    CASE l_answer.

*.......Sichern und beenden
      WHEN c_yes.
        g_save = c_true.
        g_exit = c_true.

*.......Beenden, ohne zu sichern
      WHEN c_no.
```

```
        g_save = c_false.
        g_exit = c_true.

*.......Abbrechen der Funktion -> Dynpro wird nicht beendet
      WHEN c_cancel.
        g_save = c_false.
        g_exit = c_false.
    ENDCASE.

*...wenn Daten nicht geändert, dann Beenden immer möglich
  ELSE.
    g_save = c_false.
    g_exit = c_true.
  ENDIF.
ENDFORM.                                " SAVE_QUESTION_9200

FORM cancel_question_9200.
  DATA: l_answer.

  CALL FUNCTION 'POPUP_TO_CONFIRM_LOSS_OF_DATA'
    EXPORTING

*...........Text: Wirklich abbrechen?
          textline1    = text-011
*         TEXTLINE2    = ' '

*...........Text: Achtung
          titel        = text-010
*         START_COLUMN = 25
*         START_ROW    = 6
     IMPORTING
          answer       = l_answer
     EXCEPTIONS
          OTHERS       = 1.

*...wenn wirklich abbrechen, dann EXIT-Flag setzen
  IF l_answer = c_yes.
    g_exit = c_true.
    g_save = c_false.

*...sonst EXIT-Flag zurücksetzen
  ELSE.
    g_exit = c_false.
  ENDIF.
ENDFORM.                                " CANCEL_QUESTION_9200
```

Stichwortverzeichnis

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z