Jürgen Schwaninger

ABAP®-Programmierung für die SAP®-Materialwirtschaft – Kundeneigene Erweiterungen

Galileo Press

Bonn • Boston

Inhalt

Vorwort

Manche Dinge geschehen einfach überraschend. So auch die plötzliche Anfrage vor mittlerweile drei Jahren aus dem Lektorat von SAP PRESS, ob ich nicht ein Buch über User-Exits und BAdIs in der SAP-Materialwirtschaft verfassen möchte. Ich hatte zwar zu diesem Zeitpunkt bereits begonnen, ein kleines Online-Buch zum ABAP Debugger zu erstellen, aber ein richtiges Buch zu schreiben – das war schon etwas ganz anderes. Doch in den vielen Jahren, in denen ich bereits als Logistikberater und -entwickler tätig war, hatte ich wahrscheinlich Hunderte von Erweiterungen für Kunden ausprogrammiert, und so dachte ich mir, warum nicht?

Schwieriger als diese Entscheidung zu treffen, war die Frage, wie das Buch aufgebaut sein soll. Es war mir von vornherein klar, dass ich unmöglich auf alle Erweiterungsmöglichkeiten im Detail eingehen kann, dafür ist deren Anzahl in der Materialwirtschaft einfach zu groß. Auf der anderen Seite wollte ich auf keinen Fall eine unvollständige Übersicht erstellen, es reichen schon die Lücken, die selbst die offizielle Dokumentation in Teilen aufweist. Entstanden ist daher eine, so hoffe ich, gelungene Mischung: In den Hauptkapiteln dieses Buches werden einzelne ausgewählte Erweiterungen ausführlich erläutert – insbesondere die komplexeren und nicht ganz einfach zu verwendenden Erweiterungen stehen dabei im Vordergrund –, während Sie im Anhang eine strukturierte und vollständige Übersicht über alle User-Exits und BAdIs finden.

Sie halten nun bereits die zweite Auflage dieses Buches in der Hand. Neben vielen kleinen Verbesserungen habe ich vor allem ein neues Kapitel zu Erweiterungen im Materialstamm hinzugefügt.

Zu verdanken ist dieses Buch auch allen, die mich bei der Umsetzung dieses Projektes unterstützt haben, insbesondere seien hier Kristina Noe, Philipp Grimm und Timo Ellenberger genannt. Auch Stefan Proksch und Janina Schweitzer aus dem Lektorat SAP PRESS danke ich ganz herzlich für die hervorragende Unterstützung bei der Umsetzung dieses Projektes.

Ganz besonderer Dank geht an meine Familie, die mir den notwendigen Freiraum gegeben hat, um neben der normalen Arbeitszeit dieses Buch zu schreiben.

Jürgen Schwaninger
Senior Consultant Logistics & ABAP
conlutio UG

Einleitung

Die Komponente – früher auch: das Modul – für Materialwirtschaft gehört mit Sicherheit zu den größten ihrer Art in SAP ERP, gerade daher sind die Einstellungsmöglichkeiten im Customizing sehr umfassend. Doch auch die Prozesse bei Kunden sind in diesem Bereich hoch komplex und variantenreich, sodass man früher oder später an die Grenzen des Customizings stößt.

Zielsetzung

Um den geforderten Prozessen dennoch gerecht zu werden, bietet SAP eine große Anzahl an User-Exits und BAdIs in der Materialwirtschaft an, die es ermöglichen, höchst individuell Anforderungen und Prozesse zu realisieren. Dieses Buch zeigt Ihnen die vorhandenen Möglichkeiten und erklärt Ihnen zu einer Auswahl von Erweiterungen das genaue Vorgehen, sodass auch Sie diese Technik zur Optimierung Ihrer Prozesse einsetzen können.

Sie lernen zunächst allgemein den Umgang mit Erweiterungen, BAdIs und Erweiterungsspots kennen, sodass Sie die Beispiele aus diesem Buch problemlos nachvollziehen können. Zu ausgewählten Erweiterungen wird ihre Nutzung und Ausprogrammierung in ABAP anhand von Schritt-für-Schritt-Anleitungen erklärt. Alle verwendeten ABAP-Listings sind vollständig dargestellt und mit ausführlichen Kommentaren versehen, sodass Sie sie leicht verstehen und in Ihren eigenen Anwendungen einsetzen können.

Aufbau und Inhalt

Wenn Sie nur selten Erweiterungen programmieren, finden Sie gleich in **Kapitel 1** eine ausführliche Einführung in das Konzept von User-Exits, BAdIs und Erweiterungsspots. Mit kurzen Beispielen erläutere ich Ihnen, wie Sie diese Erweiterungsmöglichkeiten verwenden und aktivieren.

Es folgen Kapitel für jeden der großen Bereiche in der Materialwirtschaft, das heißt Einkauf (**Kapitel 2**), Dienstleistungsabwicklung (**Kapitel 3**), Bestandsführung (**Kapitel 4**), Bewertung und Kontierung (**Kapitel 5**), Logistik-Rechnungsprüfung (**Kapitel 6**) sowie Materialstamm (**Kapitel 7**). Die

wichtigsten und umfangreichsten Erweiterungsmöglichkeiten werden wiederum mit Beispielen erläutert, wobei die enthaltenen Beispiele dabei bewusst einfach gehalten sind, um unnötige Verwirrung zu vermeiden. Selten gibt es bei mehreren Unternehmen eine identische Problemstellung, daher vermitteln die Beispiele die grundsätzlichen Funktionen und Möglichkeiten. Mit diesem Wissen sind Sie jedoch in der Lage, Ihre individuelle Anforderung in die Erweiterung zu übertragen.

Ein Ausflug in die Validierung und Substitution von Buchhaltungsbelegen in **Kapitel 8** rundet das Buch ab. Bei der Buchung von Warenbewegungen und Eingangsrechnungen in MM werden als Folgebelege auch automatisch Buchhaltungsbelege in SAP ERP Financials erzeugt. Häufig besteht der Wunsch, diesen FI-Beleg mit zusätzlichen Daten aus der Materialwirtschaft anzureichern oder zusätzliche Prüfungen aus Buchhaltungssicht zu verwenden. Interessant ist diese Technik auch als Ersatz für eventuell vorhandene eigene Prüfungen in User-Exits oder BAdIs, da Sie so ein zentrales Regelwerk an einer Stelle aufbauen können, ungeachtet dessen, ob ein Beleg aus der Bestandsführung, der Logistik-Rechnungsprüfung oder dem FI-System selbst kommt.

In **Anhang A** finden Sie schließlich eine Übersicht über die User-Exits und BAdIs in der SAP-Materialwirtschaft. Auch der Anhang ist dabei in die genannten Bereiche unterteilt. Gibt es in einem Bereich eine große Anzahl von Erweiterungen, habe ich diesen Bereich weiter strukturiert, sodass Sie schnell alle vorhandenen Erweiterungen zu einer bestimmten Transaktion oder zu einem bestimmten Vorgang finden können.

Zielgruppe

Dieses Buch richtet sich vor allem an MM-Berater, die nur über grundlegende ABAP-Kenntnisse verfügen, aber auch an erfahrene ABAP-Programmierer, die keine oder nur wenige Kenntnisse in der Materialwirtschaft besitzen. Doch auch wenn Sie ein erfahrener MM-Berater und Programmierer sind, können Sie dieses Buch als Nachschlagewerk verwenden und vielleicht sogar noch den einen oder anderen Trick kennenlernen.

Zu welcher Gruppe Sie auch gehören, ich habe versucht, sowohl auf ABAP-Seite als auch aufseiten der Logistik nicht zu oberflächlich zu sein, gleichzeitig aber auch nicht zu sehr in die Grundlagen abzugleiten.

Voraussetzungen

Auch wenn sich dieses Buch unter anderem an Berater mit wenig Übung im Programmieren richtet, sollten Sie grundlegende ABAP-Kenntnisse auf dem Niveau der SAP-Schulung BC400 besitzen. Auf den Einsatz von ABAP Objects wird, wenn es nicht unbedingt erforderlich ist, verzichtet, dennoch gibt es einzelne BAdIs, die einen objektorientierten Ansatz haben, der in der Ausprogrammierung beibehalten werden muss.

Sie benötigen dennoch kein tief gehendes Wissen in objektorientierter Programmierung, die Kenntnis der wesentlichen Grundbegriffe der Objektorientierung kann in diesen Fällen jedoch nicht schaden. Auf einige Spezialitäten der Objektorientierung, wie zum Beispiel die Interfaces oder die Durchführung eines Upcasts, gehe ich in diesem Buch aber kurz ein.

Die Beispiele lassen sich prinzipiell auf jedem R/3-System ab Release 4.6C oder einem ECC-System ab Release 5.0 nachvollziehen. Manche Erweiterungen wurden erst zu einem späteren Release eingeführt und stehen somit nur auf neueren SAP-Systemen zur Verfügung. Welche Voraussetzung eine bestimmte Erweiterung hat, können Sie Anhang A entnehmen.

Hinweise zur Lektüre

In diesem Buch finden Sie mehrere Orientierungshilfen, die Ihnen die Arbeit erleichtern sollen.

In hervorgehobenen Informationskästen sind Inhalte zu finden, die wissenswert und hilfreich sind, aber etwas außerhalb der eigentlichen Erläuterung stehen. Damit Sie die Informationen in den Kästen sofort einordnen können, haben wir die Kästen mit Symbolen gekennzeichnet:

Die mit diesem Symbol gekennzeichneten *Tipps* geben Ihnen spezielle Empfehlungen, die Ihnen die Arbeit erleichtern können.

In Kästen, die mit diesem Symbol gekennzeichnet sind, finden Sie Informationen zu *weiterführenden Themen* oder wichtigen Inhalten, die Sie sich merken sollten. **[+]**

Dieses Symbol weist Sie auf *Besonderheiten*, *die Sie beachten sollten,* hin. Es warnt Sie außerdem vor häufig begangenen Fehlern oder Problemen, die auftreten können. **[!]**

6 User-Exits und BAdIs in der Logistik-Rechnungsprüfung

Auch in der Logistik-Rechnungsprüfung, also der Erfassung von Eingangsrechnungen in der SAP-Materialwirtschaft, existiert ein großes Repertoire an User-Exits und BAdIs. Dennoch war es lange Zeit nicht möglich, in Transaktion MIRO eigene Felder an einen Rechnungsbeleg anzubinden, ohne das System zu modifizieren. Mit der technischen Basis SAP ECC 6.0 hat sich dies geändert, endlich existiert eine saubere Lösung zur Einbindung eines eigenen Subscreens in Form eines BAdIs. Doch auch dieses hat seine Tücken, daher wird die Nutzung dieses BAdIs einen großen Teil dieses Kapitels beanspruchen. In Abschnitt 6.1 wird sowohl die Möglichkeit der Modifikation als auch die Nutzung des BAdIs beschrieben.

In Abschnitt 6.2 finden Sie zudem eine Erweiterung, mit der Sie die Einstellungen der Toleranzprüfung im Customizing dynamisch überschreiben können.

6.1 Kundeneigene Felder in Transaktion MIRO

Mit dem BAdI `MRM_ITEM_CUSTFIELDS` existiert ab Release SAP ECC 6.0 eine Möglichkeit, einen eigenen Subscreen auf Positionsebene in den Rechnungsbeleg zu integrieren. Wie Sie dieses BAdI einsetzen und wie die Kommunikation mit Ihrem Subscreen funktioniert, werden Sie im Folgenden Schritt für Schritt erfahren.

Eigene Felder per Modifikation [+]

Wenn Sie ein älteres Release als SAP ECC 6.0 verwenden, können Sie eigene Felder auch per Modifikation realisieren. Das Vorgehen hierzu ist in den SAP-Hinweisen 352701 (Belegposition) und 174413 (Sachkontenreiter) beschrieben. In diesen Hinweisen wird erklärt, wie man ein Feld durch die Modifikation der Standard-Dynpros und Dictionary-Strukturen hinzufügen kann.

Für Felder auf dem Sachkontenreiter müssen Sie zusätzlich beachten, dass das Feld mit identischem Namen als Append-Struktur an die Tabelle `BSEG` angehängt werden muss, da es anderenfalls nicht auf der Datenbank gesichert werden kann.

Auch muss das Feld in einige Steuerungstabellen aufgenommen werden. Dies erreichen Sie über Transaktion OXK3. Wählen Sie hier zunächst das Menü Kontierungsfelder • Experten-Modus. Anschließend klappen Sie den Bereich Kundendef. Kontierungen • Kunden Include Struktur • CI_COBL auf. Setzen Sie nun den Cursor auf das betreffende Feld, und wählen Sie den Menüpunkt Kontierungsfeld • St.Einträge aufnehm. • Testlauf bzw. Echtlauf.

6.1.1 Überblick über die Lösung per BAdI

Um eigene Felder im Rechnungsbeleg (siehe Abbildung 6.1) abzubilden, benötigen Sie wieder mehrere Elemente. Für einen besseren Überblick finden Sie hier zunächst eine grobe Betrachtung der Abläufe. Im folgenden Abschnitt werden aber alle Schritte noch einmal im Detail besprochen.

Abbildung 6.1 Eigene Felder in der Rechnungserfassung

1. Zunächst einmal müssen Sie das eigentliche Dynpro definieren, das als Subscreen im Rechnungsbeleg erscheinen soll. Hierbei ist zu beachten, dass dieser Subscreen alle vorhandenen Belegpositionen abbilden muss, was bedeutet, dass Sie hier in einem eigenen Dynpro in einem Z-Programm ein Table Control mitsamt der erforderlichen Ablauflogik erstellen müssen.
2. Als Nächstes muss der Subscreen dem Rechnungsbeleg bekannt gemacht werden. In der BAdI-Implementierung müssen Sie daher den Namen des

zuvor erstellten Programms und die Dynpro-Nummer eintragen. Außerdem müssen Sie dem Karteireiter einen Namen geben, dies erfolgt in der BAdI-Methode `TABPAGE_LABEL_SET`.

Wenn Sie die BAdI-Implementierung nun aktivieren würden, wären diese beiden Schritte bereits ausreichend, um den Subscreen auf dem Bildschirm erscheinen zu lassen, wenn zum Beispiel Transaktion MIRO gestartet wird. Darüber hinaus hätten die neuen Eingabefelder selbstverständlich noch keine weitere Funktion.

3. Die Felder, die aus Ihrem Subscreen in den Rechnungsbeleg übernommen werden sollen, müssen in der Dictionary-Struktur `CI_DRSEG_CUST` angelegt werden. Diese Struktur ist bereits Bestandteil der Struktur `DRSEG_CI`, die ihrerseits der Struktur `DRSEG` zugeordnet ist. `DRSEG` ist die zentrale Struktur, die alle Positionsfelder eines Rechnungsbelegs abbildet.

   Um die Felder auch in der Datenbank zu speichern, müssen die Felder mit gleichem Namen als `APPEND` an die entsprechende Datenbanktabelle angehängt werden.

4. Nun wird es Zeit, die BAdI-Methoden mit Leben zu füllen. Die Hauptaufgabe der Methoden ist es, die Kommunikation zwischen der Positionsstruktur `DRSEG` und Ihrem Dynpro herzustellen. Hierzu werden die in Tabelle 6.1 genannten Methoden in der aufgeführten Reihenfolge automatisch aufgerufen.

Schritt	Methode/Zeitpunkt	Beschreibung
1	`CUSTOMDATA_MODIFY`	Dies ist die erste aufgerufene Methode des BAdIs. Diese wird durchlaufen, sobald der eigene Karteireiter ausgewählt wurde. Sie können diese Methode verwenden, um die eigenen Felder mit Daten vorzubelegen.
2	`INVOICE_DATA_TRANSFER`	Als Nächstes wird diese Methode durchlaufen. Sie stellen hier die Daten, die auf dem Dynpro angezeigt werden sollen, in den Attributen der Klasse bereit.
3	Anzeige des Dynpros	Das Dynpro wird jetzt angezeigt und kann vom Anwender gepflegt werden.
4	`CUSTOM_DATA_GET`	Sobald der Anwender eine Aktion auslöst, wird diese Methode aufgerufen. Sie müssen hier die vom Dynpro bereitgestellten, eventuell geänderten Daten zurückholen und in das Datenmodell zurückschreiben.

Tabelle 6.1 BAdI-Aufruf in einem Dialogschritt

5. Zuletzt müssen Sie noch die Ablauflogik Ihres Dynpros anpassen, um die zuvor bereitgestellten Daten in das Table Control zu übertragen oder zurückzugeben. Hierzu rufen Sie aus der Ablauflogik heraus die zwei verbleibenden Methoden des BAdIs auf. Diese Methoden werden nicht automatisch an einer vorgegebenen Stelle gestartet, sondern dienen nur zum Datenaustausch mit Ihrem Dynpro. Das genaue Vorgehen hierzu finden Sie in Tabelle 6.2.

Zeitpunkt	Beschreibung
PROCESS BEFORE OUTPUT (PBO)	Sobald dieser Teil der Ablauflogik durchlaufen wird, müssen Sie die darzustellenden Daten abholen. Hierzu rufen Sie aus dem PBO die Methode INVOICE_DATA_GET auf. Diese Methode liefert Ihnen die Daten, die Sie zuvor per INVOICE_DATA_TRANSFER bereitgestellt haben.
PROCESS AFTER INPUT (PAI)	Nachdem der Anwender einen Dialogschritt ausgelöst hat, prüfen Sie, ob sich eine Änderung ergeben hat. Die neuen Daten schreiben Sie durch einen Aufruf der Methode CUSTOM_DATA_TRANSFER in das BAdI zurück (um die Daten später in CUSTOM_DATA_GET weiterzuverarbeiten).

Tabelle 6.2 Übersicht über Ablauflogik Ihres Dynpros

Sobald Sie diese Schritte ausgeführt haben, können Sie Ihre eigenen Felder verwenden und zusammen mit dem Beleg speichern. In den folgenden Abschnitten schauen wir uns dieses Vorgehen noch einmal im Detail an.

6.1.2 BAdI im Detail – Anpassungen im ABAP Dictionary

Im folgenden Beispiel sollen zwei Felder, ZZTEXT_1 und ZZTEXT_2, je Position erfasst und gesichert werden können. Felder im Kundennamensraum sollten immer mit einem Doppel-Z beginnen. Beide Felder sollen das Datenelement CHAR32 verwenden, das in jedem System vorhanden ist.

Wie bereits erläutert, müssen diese Felder zunächst in die Arbeitsstruktur DRSEG aufgenommen werden, die für alle Positionen im Rechnungsbeleg verwendet wird. Die Struktur DRSEG enthält bereits ein Include DRSEG_CI, das drei feststehende Felder sowie die Include-Struktur CI_DRSEG_CUST beinhaltet. In diese Struktur können Sie Ihre eigenen Felder aufnehmen.

1. Gehen Sie in das ABAP Dictionary (Transaktion SE11), und bearbeiten Sie die Struktur CI_DRSEG_CUST (Auswahl Datentyp). Normalerweise ist diese

Struktur noch nicht vorhanden, und Sie müssen sie anlegen. Wechseln Sie in den Bearbeitungsmodus.

2. Tragen Sie die beiden Felder ZZTEXT_1 und ZZTEXT_2 ein, und verwenden Sie jeweils das Datenelement CHAR32.
3. Über das Menü Zusätze • Erweiterungskategorie können Sie die Kategorie nicht erweiterbar setzen. Tun Sie dies nicht, ist dies nicht weiter schlimm, Sie werden jedoch bei der Aktivierung eine Warnmeldung erhalten.
4. Sichern und aktivieren Sie Ihre Struktur. Die Aktivierung kann etwas Zeit in Anspruch nehmen, da alle Objekte, die die Struktur DRSEG verwenden, ebenfalls neu generiert werden.
5. Betrachten Sie nun die Struktur DRSEG_CI. Diese sollte die Felder wie in Tabelle 6.3 enthalten. Wie Sie sehen, enthält die Struktur neben Ihren eigenen Feldern auch die Positionsnummer, zu der Ihre Felder gehören. Damit beschreibt diese Struktur genau eine Zeile des Belegs aus Sicht Ihrer eigenen Felder. Sie werden diese Struktur im weiteren Verlauf daher häufiger verwenden.

Feldname	Beschreibung
C_RBLGP	Positionsnummer des Belegs
C_KOART	Kontoart: ▸ A – Anlagen ▸ D – Debitoren ▸ K – Kreditoren ▸ M – Material ▸ S – Sachkonten
C_MATNR	Materialnummer bei Kontoart M
ZZTEXT_1	Ihr erstes eigenes Feld
ZZTEXT_2	Ihr zweites eigenes Feld

Tabelle 6.3 Struktur DRSEG_CI

Damit sind alle Voraussetzungen erfüllt, um später mit Ihren Feldern innerhalb des BAdIs und in Ihrem Dynpro zu arbeiten. Damit die Daten jedoch auch in der Datenbank gesichert werden, müssen Sie die Felder zusätzlich per APPEND an die entsprechende Tabelle anhängen. In Tabelle 6.4 sehen Sie alle Möglichkeiten, die Ihnen hierfür zur Verfügung stehen.

Tabelle/Struktur	Beschreibung
RSEG	Datenbanktabelle zur Speicherung der Belegpositionen
RBDRSEG	Datenbanktabelle zur temporären Speicherung der Belegpositionen in der Hintergrundprüfung
RBMA	Datenbanktabelle zur Speicherung von materialbezogenen Feldern
COBL_MRM	Struktur der Kontierungsfelder (Sachkontenreiter). Felder, die Sie dieser Struktur hinzufügen, werden in der Datenbanktabelle RBCO gesichert.

Tabelle 6.4 Tabellen zur Speicherung der eigenen Felder

Da die Felder ZZTEXT_1 und ZZTEXT_2 keine Kontierungsinformationen darstellen und nicht materialbezogen sind, sollen sie in Tabelle RSEG gesichert werden.

1. Lassen Sie sich im Dictionary (Transaktion SE11) die Datenbanktabelle RSEG anzeigen. Klicken Sie dann oben rechts auf die Schaltfläche Append-Struktur.
2. Im erscheinenden Auswahlfenster klicken Sie auf das Icon Append anlegen. Als Namen für die Append-Struktur können Sie zum Beispiel ZZRSEG verwenden.
3. Geben Sie nun wieder die beiden Felder ZZTEXT_1 und ZZTEXT_2 an. Achten Sie unbedingt auf die absolut identische Schreibweise mit den Feldnamen in der Struktur CI_DRSEG_CUST. Verwenden Sie außerdem unbedingt dasselbe Datenelement (CHAR32) für diese Felder (siehe Abbildung 6.2).

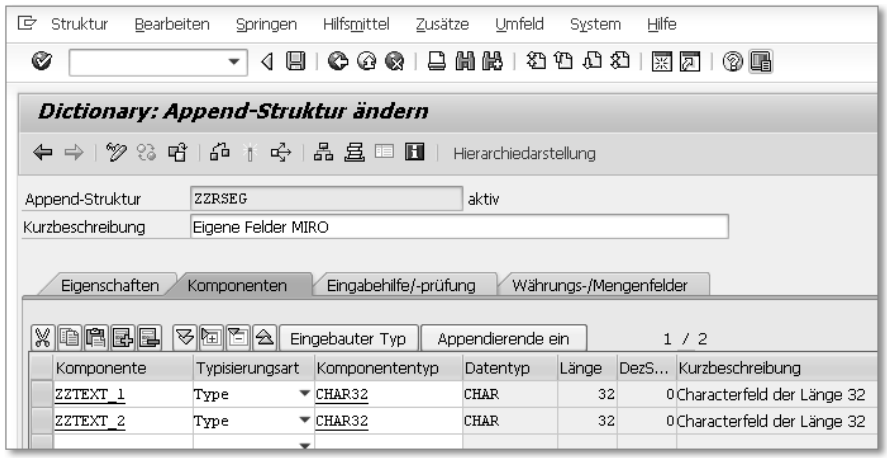


Abbildung 6.2 Append-Struktur ZZRSEG

4. Pflegen Sie auch hier wieder die Erweiterungskategorie (Menü Zusätze • Erweiterungskategorie) als nicht erweiterbar.
5. Sichern und aktivieren Sie die Append-Struktur.

Wenn Sie die Hintergrundprüfung verwenden und über Transaktion MIRA Belege erfassen, die Sie anschließend über Transaktion MIR6 oder MIR4 weiterbearbeiten und dort zur Hintergrundprüfung einplanen, werden die Positionsdaten temporär in der Tabelle RBDRSEG gespeichert. Erst wenn die Hintergrundprüfung (Transaktion MRBP) ohne Fehler durchlaufen wurde, werden die Sätze wieder aus Tabelle RBDRSEG gelöscht und in Tabelle RSEG gesichert.

Damit auch in diesem Verfahren zu jeder Zeit die eigenen Felder erfasst werden können, sollten Sie auch die Tabelle RBDRSEG identisch mit Tabelle RSEG um die eigenen Felder erweitern.

6.1.3 Eigenes Dynpro mit Table Control erstellen

Als Nächstes benötigen Sie natürlich auch das Dynpro, das als Subscreen in den Karteireiter eingebunden werden soll. Diesen packen Sie am besten in ein eigenes Programm in Form einer Funktionsgruppe oder in einen Report vom Typ Modulpool.

1. Starten Sie Transaktion SE80, und legen Sie ein neues Programm an. Für das Beispiel wurde der Name SAPMZMRM und als Programmtyp ein Modulpool verwendet (siehe Abbildung 6.3).

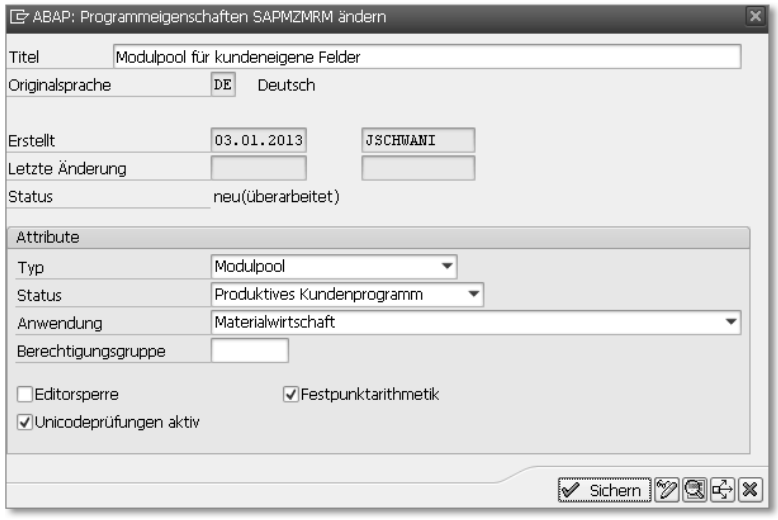


Abbildung 6.3 Modulpool zur Aufnahme des Dynpros

2. Bevor Sie das Dynpro mit dem Table Control erzeugen, sollten Sie überlegen, welche Felder im Table Control dargestellt werden sollen. Sie benötigen eine interne Tabelle und eine Arbeitsstruktur für diese Felder, um das Table Control daraus mit Daten zu versorgen.

   Sie können einfach die Struktur DRSEG_CI verwenden, diese enthält neben Ihren eigenen Feldern auch die Positionsnummer und eignet sich daher gut für eine Anzeige auf dem Bildschirm. Erstellen Sie deshalb im Programm SAPMZMRM zwei globale Datenobjekte wie in Listing 6.1.

```
PROGRAM  sapmzmrm.

* Globale Daten für Table Control
DATA: gs_drseg_ci TYPE drseg_ci,
      gt_drseg_ci TYPE TABLE OF drseg_ci.
```

Listing 6.1 Globale Daten für Table Control

3. Aktivieren Sie zunächst Ihr Programm. Klicken Sie dann in Transaktion SE80 links im Repository Browser mit der rechten Maustaste auf den Programmnamen. Im erscheinenden Kontextmenü wählen Sie Anlegen • Dynpro. Als Dynpro-Nummer geben Sie die 0100 an.
4. Vergeben Sie eine kurze Beschreibung für dieses Dynpro, und wählen Sie auf dem Karteireiter Eigenschaften im Abschnitt Dynprotyp unbedingt die Option Subscreen, da andere Dynpro-Typen nicht eingebunden werden können und Sie beim Starten von Transaktion MIRO einen Kurzdump erhalten würden.
5. Wechseln Sie dann über das Icon Layout in den Screen Painter. Hier erzeugen Sie als Erstes ein Table Control. Hierfür gibt es einen nützlichen Wizard, der Ihnen viel Arbeit abnimmt. Klicken Sie hierzu auf der linken Seite auf das Icon  , Table Control (mittels Wizard), und ziehen Sie anschließend einen Kasten. Beginnen Sie am besten ganz links oben, und zeichnen Sie Ihr Table Control ungefähr 100 Zeichen breit und 18 Zeilen hoch. Sie sehen rechts oben während der Erstellung einen Zähler mit Breite und Höhe des Table Controls (siehe Abbildung 6.4).
6. Sobald Sie das Table Control gezeichnet haben, erscheint nun der Wizard, wie in Abbildung 6.5 zu sehen ist. Überspringen Sie den Einführungsbildschirm durch einen Klick auf die Schaltfläche Weiter.

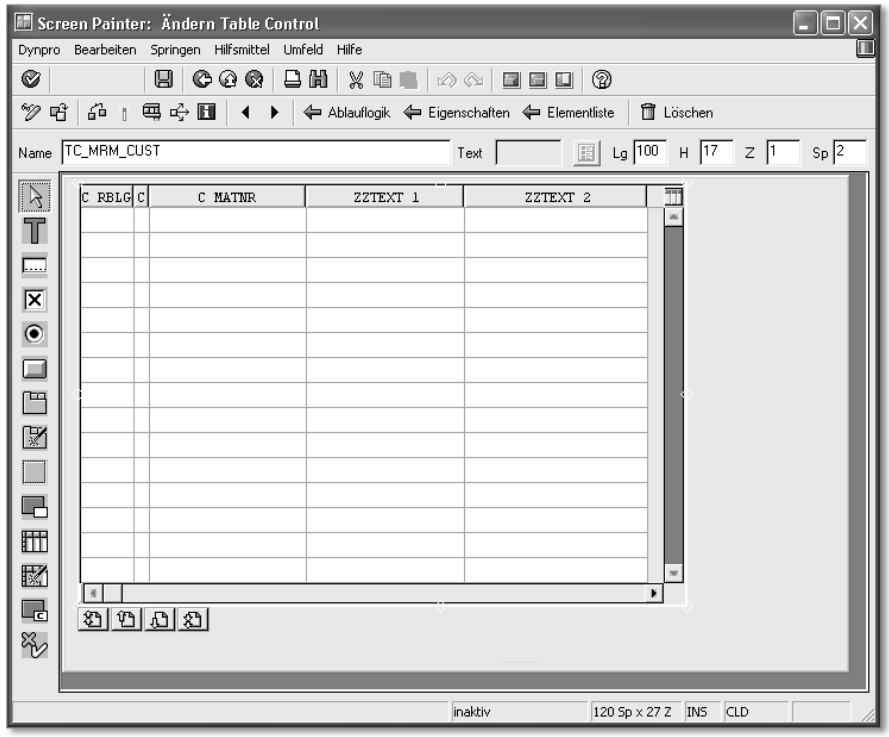


Abbildung 6.4 Fertiges Table Control im Screen Painter

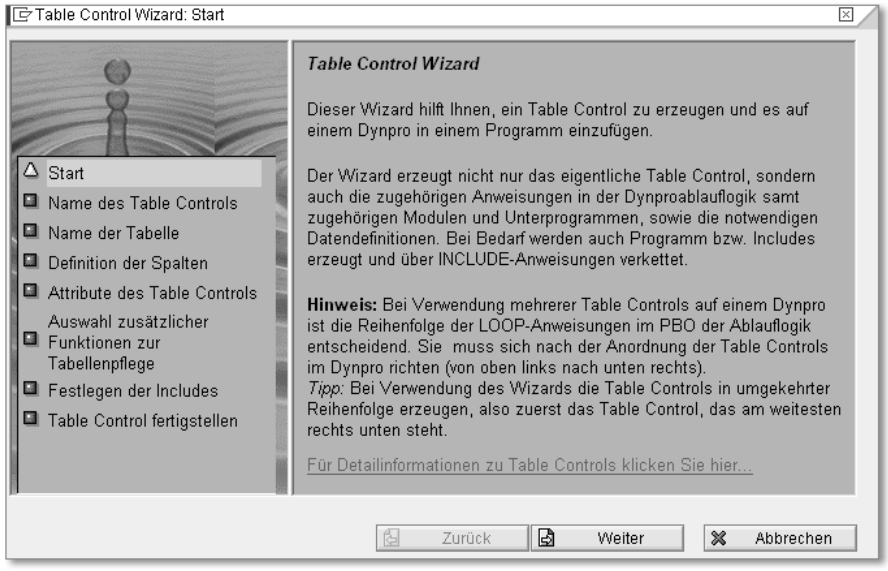


Abbildung 6.5 Table Control Wizard

7. Im Abschnitt Name des Table Controls geben Sie einen eindeutigen Namen an, im Beispiel wurde `TC_MRM_CUST` verwendet. Klicken Sie auf Weiter.
8. In Name der Tabelle wählen Sie als Typ die interne Programmtabelle und tragen dort die Tabelle `GT_DRSEG_CI` ein. Da diese interne Tabelle keine Kopfzeile besitzt, müssen Sie auch die Option Tabellen-Arbeitsbereich aktivieren und die Struktur `GS_DRSEG_CI` eintragen. Klicken Sie auf Weiter.
9. Im Schritt Definition der Spalten markieren Sie einfach alle angebotenen Spalten (die aus der Struktur `DRSEG_CI` übernommen wurden). Klicken Sie auf Weiter.
10. Im Schritt Attribute des Table Controls markieren Sie im Abschnitt Ein-/Ausgabe-Attribute die Option Eingabe, damit sind die Felder im Standardzustand eingabebereit. Alle weiteren Optionen lassen Sie unverändert. Klicken Sie auf Weiter.
11. Im Schritt Auswahl zusätzlicher Funktionen zur Tabellenpflege können Sie noch die Option Blättern aktivieren. Hierdurch werden unter das Table Control vier Schaltflächen gesetzt, mit denen der Anwender schnell durch die Zeilen des Controls blättern kann. Klicken Sie auf Weiter.
12. Im Schritt Festlegen der Includes werden Ihnen Bezeichnungen für verschiedene Includes vorgeschlagen, in denen die notwendigen Datendeklarationen und Ablaufmodule erstellt werden sollen. Übernehmen Sie den Vorschlag unverändert, und klicken Sie auf Weiter.
13. Im letzten Schritt müssen Sie nur noch auf Fertigstellen klicken, und alle für das Table Control notwendigen Objekte und Module werden erzeugt.
14. Sichern Sie Ihr Dynpro, und verlassen Sie den Screen Painter. Anschließend sollten Sie das Dynpro aktivieren, markieren Sie dabei auch alle anderen erzeugten Objekte, die Ihnen vorgeschlagen werden, und aktivieren Sie diese ebenfalls.

Damit steht das Grundgerüst Ihres Dynpros, noch fehlt aber die Kommunikation mit dem BAdI, um das Table Control mit Daten zu füllen und geänderte Werte zurückzuschreiben. Doch bevor Sie diesen Teil angehen, sollten Sie sich zunächst einmal dem BAdI widmen.

6.1.4 Vorbereitung der Daten im BAdI

Nachdem Sie diese Vorbereitungen getroffen haben, können Sie sich jetzt den ersten Methoden des BAdIs zuwenden. Betrachten Sie zunächst noch einmal Tabelle 6.1 und Tabelle 6.2. Bei der Bearbeitung von Eingangsrech-

nungen werden vom System die BAdI-Methoden der Tabelle 6.1 aufgerufen, hier müssen Sie die Daten gegebenenfalls aufbereiten und im BAdI zwischenspeichern. Dies geschieht, indem Sie die Daten in Attributen der zugehörigen Klasse ablegen. Wenn Sie die Daten dann später (siehe Tabelle 6.2) in der Ablauflogik Ihres Dynpros aus dem BAdI abholen, werden letztendlich die Daten aus diesen Attributen verwendet.

1. Legen Sie zunächst in Transaktion SE19 eine Implementierung zum BAdI `MRM_ITEM_CUSTFIELDS` an. Als Name für die Implementierung können Sie zum Beispiel `ZMRM_ITEM_CUSTFIELDS` verwenden. Sichern Sie die Implementierung.
2. Als Erstes sollten Sie Ihr zuvor erstelltes Dynpro in der Implementierung hinterlegen. Wechseln Sie hierzu auf den Karteireiter SUBSCREENS, und tragen Sie den Namen und die Dynpro-Nummer als gerufenes Programm ein (siehe Abbildung 6.6).

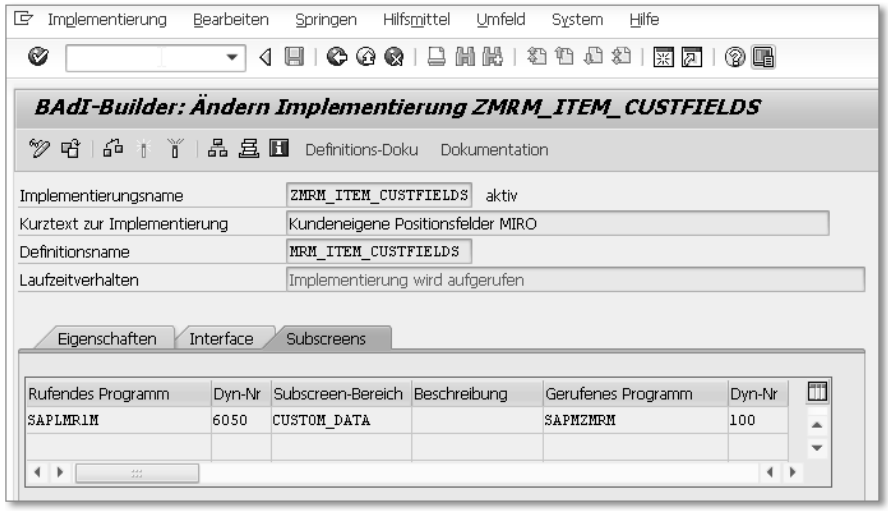


Abbildung 6.6 Karteireiter »Subscreens« der BAdI-Implementierung

3. Wechseln Sie nun auf den Karteireiter INTERFACE, und navigieren Sie durch einen Doppelklick in die implementierende Klasse (`ZCL_IM_MRM_ITEM_CUSTFIELDS`).
4. In der Klasse sollten Sie nun zunächst einmal einen Blick auf den Karteireiter ATTRIBUTE werfen. Sie werden hier sechs Attribute finden (siehe Tabelle 6.5), die automatisch angelegt wurden. Diese Attribute wurden bereits im Interface `IF_EX_MRM_ITEM_CUSTFIELDS` definiert, von dem das BAdI abgeleitet wurde.

Attribut	Beschreibung
IF_EX_MRM_ITEM_CUSTFIELDS~TRANSACTION_TYPE	Dieses Feld enthält den Transaktionstyp. Die wichtigsten möglichen Werte sind folgende: A – Anzeigen H – Hinzufügen (neuer Beleg) V – Verändern
IF_EX_MRM_ITEM_CUSTFIELDS~S_RBKPV	Diese Struktur enthält alle Daten des Belegkopfes.
IF_EX_MRM_ITEM_CUSTFIELDS~TAB_DRSEG	In dieser internen Tabelle befinden sich alle Daten der Belegpositionen, einschließlich Ihrer eigenen Felder.
IF_EX_MRM_ITEM_CUSTFIELDS~TAB_DRSEG_CUSTOM	Diese interne Tabelle enthält nur Ihre eigenen Felder (Typ DRSEG_CI, siehe Tabelle 6.3).
IF_EX_MRM_ITEM_CUSTFIELDS~H_SORT	Wenn die Reihenfolge der Datensätze in Ihrem Dynpro verändert (sortiert) wurde, können Sie dieses Kennzeichen setzen, um die Sortierung beizubehalten.
IF_EX_MRM_ITEM_CUSTFIELDS~H_CHANGE	Kennzeichen: Die eigenen Daten wurden durch den Anwender verändert.

Tabelle 6.5 Attribute aus Interface IF_EX_MRM_ITEM_CUSTFIELDS

[+] **Automatische Erzeugung der Attribute**

Eine wesentliche Grundeigenschaft von Interfaces ist folgende: Wenn eine Klasse (und nichts anderes versteckt sich hinter der Implementierung eines BAdIs) ein Interface implementiert, muss die Klasse alle im Interface definierten Methoden und Attribute übernehmen. Aus diesem Grund wurden die Attribute an dieser Stelle automatisch erzeugt. Für Sie bedeutet dies, dass Sie an dieser Stelle nichts weiter tun müssen. Alle Attribute, die Sie im weiteren Verlauf benötigen, sind schon da!

5. Wechseln Sie nun wieder auf den Karteireiter Methoden. Zunächst einmal sollen die Methoden implementiert werden, die automatisch aufgerufen werden (siehe Tabelle 6.1). Die Methode CUSTOMDATA_MODIFY können Sie verwenden, um die eigenen Felder vorzubelegen, für die grundlegende Funktion wird diese jedoch nicht benötigt.

   Stattdessen widmen Sie sich der Methode INVOICE_DATA_TRANSFER. Diese Methode besitzt vier Eingangsparameter (Transaktionstyp, Rechnungsbelegkopf, Rechnungsbelegpositionen und Ihre eigenen Felder). Sie müssen in dieser Methode nichts weiter tun, als die Parameter, die hier übergeben

werden, in die Attribute der Klasse zu kopieren. Das Coding hierzu finden Sie in Listing 6.2.

```
METHOD if_ex_mrm_item_custfields~invoice_data_transfer.
* Übernahme der Eingangsparameter in Attribute
  me->if_ex_mrm_item_custfields~transaction_type =
      i_transaction_type.
  me->if_ex_mrm_item_custfields~s_rbkpv          =
      is_rbkpv.
  me->if_ex_mrm_item_custfields~tab_drseg        =
      it_drseg.
  me->if_ex_mrm_item_custfields~tab_drseg_custom =
      it_drseg_custom.
ENDMETHOD.
```

Listing 6.2 Methode INVOICE_DATA_TRANSFER

6. Nach der Änderung der Daten durch den Anwender werden die Daten im Attribut `TAB_DRSEG_CUSTOM` den neuen Stand beinhalten (dieser Teil wird später realisiert). Auch wird das Attribut `H_CHANGE` ein Kennzeichen enthalten, ob die Daten geändert wurden. In der Methode `CUSTOM_DATA_GET` müssen Sie nun nichts weiter tun, als diese Informationen wieder an das Standardprogramm zurückzugeben. Das entsprechende Coding sehen Sie in Listing 6.3.

```
METHOD if_ex_mrm_item_custfields~custom_data_get.
* Rückgabe der geänderten Attribute an Aufrufer
  et_drseg_cust =
    me->if_ex_mrm_item_custfields~tab_drseg_custom.
  e_change =
    me->if_ex_mrm_item_custfields~h_change.
ENDMETHOD.
```

Listing 6.3 Methode CUSTOM_DATA_GET

7. Als Nächstes folgen die Methoden aus Tabelle 6.2 . Die Methode `INVOICE_DATA_GET` soll aus dem Zeitpunkt `PROCESS BEFORE OUTPUT` der Ablauflogik des Dynpros aufgerufen werden. Die Aufgabe der Methode ist es demnach, die aktuellen Inhalte der Attribute zu den Positionsdaten an das Dynpro zurückzuliefern. Hierzu versorgen Sie einfach die Exportparameter der Methode mit den passenden Attributen Ihrer Klasse. Das entsprechende Coding finden Sie in Listing 6.4 .

```
METHOD if_ex_mrm_item_custfields~invoice_data_get.
* Daten aus Attributen bereitstellen für Dynpro
  e_transaction_type  =
      me->if_ex_mrm_item_custfields~transaction_type.
```

```
  es_rbkpv            =
      me->if_ex_mrm_item_custfields~s_rbkpv.
  et_drseg            =
      me->if_ex_mrm_item_custfields~tab_drseg.
  et_drseg_cust       =
      me->if_ex_mrm_item_custfields~tab_drseg_custom.
ENDMETHOD.
```

Listing 6.4 Methode INVOICE_DATA_GET

8. Die Methode `CUSTOM_DATA_TRANSFER` wird aus dem Zeitpunkt `PROCESS AFTER INPUT` der Ablauflogik des Dynpros aufgerufen. Das Dynpro liefert an dieser Stelle die eventuell geänderten eigenen Felder sowie ein Kennzeichen zurück, ob eine Datenänderung erfolgt ist. Diese Daten legen Sie in den Attributen ab. Damit kann die zuvor implementierte Methode `CUSTOM_DATA_GET` genau diese Daten weiterverarbeiten (siehe Listing 6.5).

```
METHOD if_ex_mrm_item_custfields~custom_data_transfer.
  me->if_ex_mrm_item_custfields~tab_drseg_custom =
      it_drseg_cust.
  me->if_ex_mrm_item_custfields~h_change =
      i_change.
ENDMETHOD.
```

Listing 6.5 Methode CUSTOM_DATA_TRANSFER

9. Zu guter Letzt benötigen Sie noch die Methode `TABPAGE_LABEL_SET`, um dem Karteireiter eine Bezeichnung zu geben. Verwenden Sie ein Textsymbol, um die Bezeichnung übersetzbar zu halten. Auf dem Karteireiter werden maximal elf Zeichen dargestellt (siehe Listing 6.6).

```
METHOD if_ex_mrm_item_custfields~tabpage_label_set.
* Bezeichnung des Karteireiters, max. elf Zeichen
* text-001 enthält den Text 'Zusatz'
  e_screen_name = text-001.
ENDMETHOD.
```

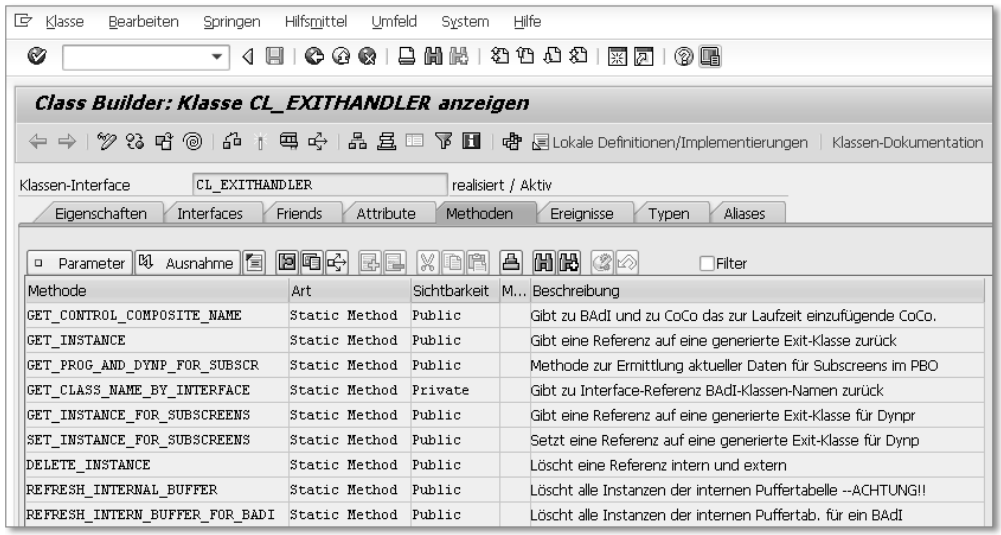
Listing 6.6 Methode TABPAGE_LABEL_SET

6.1.5 Zurück im Dynpro

Nachdem Sie nun auch im BAdI alle Vorbereitungen getroffen haben, müssen Sie noch die Datenkommunikation zwischen BAdI und Dynpro einrichten. Wie zuvor erwähnt, läuft die Kommunikation zwischen Dynpro und BAdI über die Methoden `INVOICE_DATA_GET` und `CUSTOM_DATA_TRANSFER`, die wiederum auf die Attribute der Klasse zugreifen. In der objektorientierten

Programmierung ist es jedoch so, dass der Inhalt von Attributen jeweils genau an eine Instanz der Klasse gebunden ist. Wenn Sie nun daher einfach eine neue Instanz zur implementierenden Klasse `ZCL_IM_MRM_ITEM_CUST-FIELDS` erzeugen würden, um dann die genannten Methoden aufzurufen, wäre dies erfolglos, denn die neue Instanz hätte auch eigene Attribute, die schlicht und ergreifend leer wären.

Daher müssen Sie für einen Zugriff auf genau dieselbe Instanz sorgen, die auch vom BAdI verwendet wurde. Dazu existiert die statische Klasse `CL_EXITHANDLER` (siehe Abbildung 6.7) mit der Methode `GET_INSTANCE_FOR_SUBSCREENS`. Dieser Methode müssen Sie nur eine passende Referenzvariable übergeben, die sich auf das Interface des BAdIs bezieht. Die Methode erkennt dann anhand des Interface die passende Instanz und gibt diese zurück.



Methode	Art	Sichtbarkeit	M...	Beschreibung
GET_CONTROL_COMPOSITE_NAME	Static Method	Public		Gibt zu BAdI und zu CoCo das zur Laufzeit einzufügende CoCo.
GET_INSTANCE	Static Method	Public		Gibt eine Referenz auf eine generierte Exit-Klasse zurück
GET_PROG_AND_DYNP_FOR_SUBSCR	Static Method	Public		Methode zur Ermittlung aktueller Daten für Subscreens im PBO
GET_CLASS_NAME_BY_INTERFACE	Static Method	Private		Gibt zu Interface-Referenz BAdI-Klassen-Namen zurück
GET_INSTANCE_FOR_SUBSCREENS	Static Method	Public		Gibt eine Referenz auf eine generierte Exit-Klasse für Dynpr
SET_INSTANCE_FOR_SUBSCREENS	Static Method	Public		Setzt eine Referenz auf eine generierte Exit-Klasse für Dynp
DELETE_INSTANCE	Static Method	Public		Löscht eine Referenz intern und extern
REFRESH_INTERNAL_BUFFER	Static Method	Public		Löscht alle Instanzen der internen Puffertabelle --ACHTUNG!!
REFRESH_INTERN_BUFFER_FOR_BADI	Static Method	Public		Löscht alle Instanzen der internen Puffertab. für ein BAdI

Abbildung 6.7 Methoden der Klasse CL_EXITHANDLER

1. Gehen Sie in Transaktion SE80 in Ihr Programm `SAPMZMRM`, und wechseln Sie in das Dynpro 0100. Fügen Sie in der Ablauflogik an erster Stelle im Zeitpunkt `PROCESS BEFORE OUTPUT` ein neues `MODULE` mit dem Namen `GET_DATA` ein. Doppelklicken Sie auf `GET_DATA`, um das Modul zu erzeugen. Übernehmen Sie das vorgeschlagene Include (`MZMRM_GET_DATAO01`).
2. Erstellen Sie im Modul zunächst eine neue Variable als Referenz auf das Interface des BAdIs. Sie finden den Namen des Interface in Transaktion SE18 oder SE19 auf dem Karteireiter Interface. In diesem Fall ist es das

Interface IF_EX_MRM_ITEM_CUSTFIELDS. Im Beispiel wurde diese Variable MRM_CUSTFIELDS genannt.

3. Rufen Sie nun die Methode GET_INSTANCE_FOR_SUBSCREENS der Klasse CL_EXITHANDLER auf, und übergeben Sie Ihre zuvor definierte Variable. Nach dem Aufruf zeigt Ihre Variable auf die Instanz des BAdIs.
4. Holen Sie nun über die Methode INVOICE_DATA_GET alle zur Verfügung stehenden Daten aus dem BAdI ab. Um die Daten aufnehmen zu können, müssen Sie noch passende Variablen in Ihrem Programm anlegen. Legen Sie diese Variablen im globalen Bereich ab, sodass Sie auch später noch Zugriff darauf haben.
5. Zuletzt müssen Sie die Daten in Ihr Table Control übertragen. Da das Table Control alle Felder der Struktur DRSEG_CI verwendet, können Sie den Inhalt der internen Tabelle ET_DRSEG_CUST, die Sie im vorangegangenen Schritt erhalten haben, einfach in die Tabelle GT_DRSEG_CI kopieren, die Ihr Table Control versorgt. Das entsprechende Coding für dieses Modul finden Sie in Listing 6.7 und Listing 6.8.

```
PROGRAM  sapmzmrm.
...
* Datendeklarationen für BAdI-Kommunikation
TYPE-POOLS: mrm, mmcr.

DATA: gv_trtyp LIKE t169-trtyp,
      gs_rbkpv TYPE mrm_rbkpv,
      gt_drseg TYPE mmcr_tdrseg,
      gt_drseg_custom TYPE tdrseg_cust.
```

Listing 6.7 Datendeklarationen in Programm SAPMZMRM

```
*---------------------------------------------------------------*
***INCLUDE MZMRM_GET_DATAO01 .
*---------------------------------------------------------------*
*&      Module  GET_DATA  OUTPUT
*&---------------------------------------------------------------
module GET_DATA output.

* Referenz auf das Interface
DATA: mrm_custfields TYPE REF TO if_ex_mrm_item_custfields.
* Instanz holen
  CALL METHOD cl_exithandler=>get_instance_for_subscreens
    CHANGING
      instance = mrm_custfields
    EXCEPTIONS
      OTHERS   = 6.
```

```
* Daten aus BAdI holen
  CALL METHOD mrm_custfields->invoice_data_get
    IMPORTING
      e_transaction_type = gv_trtyp
      es_rbkpv           = gs_rbkpv
      et_drseg           = gt_drseg
      et_drseg_cust      = gt_drseg_custom.

* Übertrag in Table Control
  gt_drseg_ci[] =  gt_drseg_custom[].

endmodule.                    " GET_DATA  OUTPUT
```

Listing 6.8 Modul GET_DATA

6. Wechseln Sie nun wieder zurück in die Ablauflogik Ihres Dynpros. Legen Sie ein neues Modul mit Namen `WRITE_DATA` als letztes Modul im Zeitpunkt `PROCESS AFTER INPUT` an. Klicken Sie doppelt auf den Modulnamen, um das Modul erzeugen zu lassen, und verwenden Sie wieder das vorgeschlagene Include (`MZMRM_WRITE_DATAI01`).
7. Sie müssen an dieser Stelle die Daten wieder zurück in das BAdI übertragen. Grundsätzlich kann keine Änderung erfolgen, wenn sich der Anwender nur im Anzeigemodus befindet. Daher müssen Sie die folgenden Prüfungen nur durchführen, wenn der Transaktionstyp, den Sie in `GV_TRTYP` zwischengespeichert haben, ungleich `'A'` ist.
8. Anschließend machen Sie einen `LOOP` über die aktuellen Daten Ihres Table Controls (`GT_DRSEG_CI`) und vergleichen jede Zeile mit dem ursprünglichen Zustand, der im PBO in der internen Tabelle `GT_DRSEG_CUSTOM` gespeichert wurde. Hat sich eine Zeile geändert, merken Sie sich dies, und schreiben Sie die Änderung in die Tabelle `GT_DRSEG_CUSTOM` zurück.
9. Danach können Sie die Methode `CUSTOM_DATA_TRANSFER` anlegen. Die im PBO definierte Referenzvariable `MRM_CUSTFIELDS` steht noch zur Verfügung und kann hierfür verwendet werden. Geben Sie die Tabelle `GT_DRSEG_CUSTOM` zurück. Falls Daten verändert wurden, setzen Sie das Kennzeichen `I_CHANGE` in der Parameterschnittstelle der aufgerufenen Methode. Den Parameter `I_SORT` verwenden wir im Beispiel nicht, Sie können hier einfach `ABAP_FALSE` zurückgeben.

Das vollständige Coding zum Modul `WRITE_DATA` finden Sie in Listing 6.9.

```
*---------------------------------------------------------------*
***INCLUDE MZMRM_WRITE_DATAI01 .
*---------------------------------------------------------------*
```

```
*&      Module  WRITE_DATA  INPUT
*&---------------------------------------------------------------------
MODULE write_data INPUT.

* Lokale Deklarationen
  DATA lv_change TYPE c.
  DATA ls_drseg_cust TYPE drseg_ci.

* Nur wenn nicht im Anzeigemodus
  CHECK gv_trtyp NE 'A'.

* Wurden Daten geändert?
  LOOP AT gt_drseg_ci INTO gs_drseg_ci.
    READ TABLE gt_drseg_custom
         INTO ls_drseg_cust INDEX sy-tabix.
    IF ls_drseg_cust NE gs_drseg_ci.
*     Daten wurden geändert
      lv_change = 'X'.
      MODIFY gt_drseg_custom FROM gs_drseg_ci
             INDEX sy-tabix.
    ENDIF.
  ENDLOOP.

* Daten zurückschreiben
  CALL METHOD mrm_custfields->custom_data_transfer
    EXPORTING
      i_sort        = abap_false
      it_drseg_cust = gt_drseg_custom
      i_change      = lv_change.

ENDMODULE.                 " WRITE_DATA  INPUT
```

Listing 6.9 Modul WRITE_DATA

Aktivieren Sie jetzt Ihre letzten Änderungen, und achten Sie auch darauf, dass die BAdI-Implementierung aktiv ist. Anschließend können Sie die neue Funktion in Transaktion MIRO testen, der zusätzliche Karteireiter erscheint nun, und auch die eingegebenen Daten werden dargestellt.

Es ist jedoch nicht ideal, dass alle Felder immer eingabebereit sind. Besser ist es, wenn die drei Standardfelder, zum Beispiel die Positionsnummer, grundsätzlich nicht eingabebereit sind. Außerdem sollte darauf geachtet werden, dass auch Ihre eigenen Felder deaktiviert werden, wenn eine reine Anzeigetransaktion darauf zugreift. Darüber hinaus ist Ihnen vielleicht aufgefallen, dass der Anwender auch in neue Positionen Werte eingeben kann. Da Ihre eigenen Felder jedoch positionsbezogen sind, darf die Eingabe nur möglich sein, wenn auch tatsächlich eine Position im Beleg vorhanden ist. Dies lässt

sich einfach über das Feld C_RBLGP prüfen – ist hier keine Positionsnummer vorhanden, darf auch keine Eingabe möglich sein.

1. Wechseln Sie nochmals in die Ablauflogik Ihres Dynpros. Der Table Control Wizard hat Ihnen bereits ein Modul vorgeschlagen, das Sie verwenden können, um die Attribute der Felder zu verändern (TC_MRM_CUST_CHANGE_FIELD_ATTR).
2. Entfernen Sie den Kommentar vor dieser Zeile, und klicken Sie doppelt auf den Namen, um das Modul anzulegen. Da es sich um ein weiteres PBO-Modul zum Table Control handelt, können Sie es im vorhandenen Include MZMRMO01 erstellen lassen.
3. Anschließend können Sie dort die Felder ändern, wie zuvor erläutert. Da dies keine spezielle Technik des BAdIs ist, sondern zur allgemeinen Programmierung von Table Controls gehört, wurde nicht detailliert darauf eingegangen. Sie finden das vollständige Coding jedoch in Listing 6.10. Die vollständige Ablauflogik zum Dynpro können Sie in Listing 6.11 überprüfen.

```
*&---------------------------------------------------------------------*
*&      Module  TC_MRM_CUST_CHANGE_FIELD_ATTR  OUTPUT
*&---------------------------------------------------------------------*
MODULE tc_mrm_cust_change_field_attr OUTPUT.
  DATA ls_col TYPE cxtab_column.

* Für alle Spalten der aktuellen Zeile
  LOOP AT tc_mrm_cust-cols INTO ls_col.

* Standardfelder: Eingabe verhindern
    IF ls_col-screen-name EQ 'GS_DRSEG_CI-C_RBLGP' OR
       ls_col-screen-name EQ 'GS_DRSEG_CI-C_KOART' OR
       ls_col-screen-name EQ 'GS_DRSEG_CI-C_MATNR'.
      ls_col-screen-input = 0.
    ENDIF.

* Im Anzeigemodus oder falls keine Positionsnummer
* zugeordnet ist - für eigene Felder Eingabe verhindern
    IF ls_col-screen-name EQ 'GS_DRSEG_CI-ZZTEXT_1' OR
       ls_col-screen-name EQ 'GS_DRSEG_CI-ZZTEXT_2'.
      IF gv_trtyp = 'A' OR
         gs_drseg_ci-c_rblgp IS INITIAL.
        ls_col-screen-input = 0.
      ELSE.
        ls_col-screen-input = 1.
      ENDIF.
    ENDIF.
```

```
    MODIFY tc_mrm_cust-cols FROM ls_col.
  ENDLOOP.

ENDMODULE.            " TC_MRM_CUST_CHANGE_FIELD_ATTR  OUTPUT
```

Listing 6.10 Modul TC_MRM_CUST_GET_LINES OUTPUT

```
PROCESS BEFORE OUTPUT.
  MODULE get_data.

*&SPWIZARD: PBO FLOW LOGIC FOR TABLECONTROL 'TC_MRM_CUST'
  MODULE tc_mrm_cust_change_tc_attr.
*&SPWIZARD: MODULE TC_MRM_CUST_CHANGE_COL_ATTR.
  LOOP AT   gt_drseg_ci
       INTO gs_drseg_ci
       WITH CONTROL tc_mrm_cust
       CURSOR tc_mrm_cust-current_line.
    MODULE tc_mrm_cust_get_lines.
    MODULE tc_mrm_cust_change_field_attr.
  ENDLOOP.

PROCESS AFTER INPUT.
*&SPWIZARD: PAI FLOW LOGIC FOR TABLECONTROL 'TC_MRM_CUST'
  LOOP AT gt_drseg_ci.
    CHAIN.
      FIELD gs_drseg_ci-c_rblgp.
      FIELD gs_drseg_ci-c_koart.
      FIELD gs_drseg_ci-c_matnr.
      FIELD gs_drseg_ci-zztext_1.
      FIELD gs_drseg_ci-zztext_2.
      MODULE tc_mrm_cust_modify ON CHAIN-REQUEST.
    ENDCHAIN.
  ENDLOOP.

  MODULE tc_mrm_cust_user_command.

  MODULE write_data.
```

Listing 6.11 Ablauflogik zu Dynpro 0100

6.2 Toleranzprüfungen übersteuern

Mit den Toleranzgrenzen im Customizing der Logistik-Rechnungsprüfung stehen Ihnen vielfältige Möglichkeiten zur Verfügung, um das Sperrverhalten von Eingangsrechnungen bei Preis- oder Mengenabweichungen zu beeinflussen. Dabei werden bei der Erfassung der Eingangsrechnung absolute oder

prozentuale Ober- und Untergrenzen geprüft. Weicht ein Wert (Menge/Preis) über diese Grenzen hinaus von der zugehörigen Bestellung ab, wird der Beleg gesperrt. Durch die Sperrung kann die Rechnung in der Finanzbuchhaltung nicht zur Zahlung angewiesen werden, solange die Sperre nicht beseitigt wurde. Mit der Erweiterung `MM08R002` können Sie zusätzlich diese Ober- und Untergrenzen individuell übersteuern.

6.2.1 Toleranzgrenzen im Customizing

Im Customizing (Transaktion SPRO) finden Sie die Einstellungen zu Toleranzgrenzen unter Materialwirtschaft • Logistik-Rechnungsprüfung • Rechnungssperre • Toleranzgrenzen festlegen. Abhängig vom Buchungskreis und dem Toleranzschlüssel, können Sie hier absolute und prozentuale Ober- und Untergrenzen für Differenzen pflegen (siehe Abbildung 6.8).

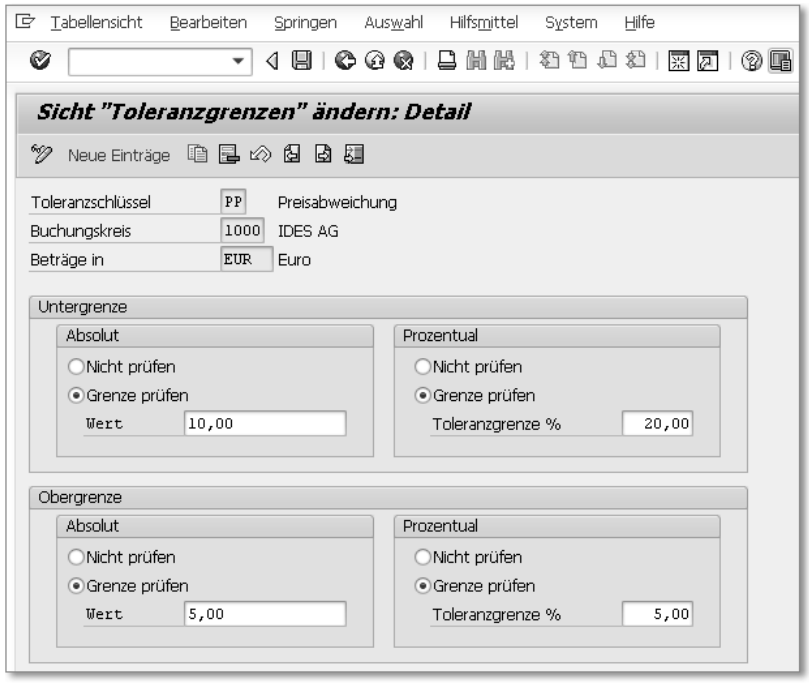


Abbildung 6.8 Toleranzgrenzen im Customizing

Setzen Sie bei einer Grenze die Option Nicht prüfen, kann die Rechnungsposition in dieser Richtung unbegrenzt abweichen. Werden diese Grenzen überschritten, kann die Rechnung zwar gesichert werden (in der Standard-

auslieferung ist die zugehörige Meldung als Warnung eingestellt), wird jedoch mit einem Sperrgrund versehen.

Der Toleranzschlüssel steht für eine SAP-seitig festgelegte Art der Abweichung zwischen der Eingangsrechnung und Bestellung oder dem Wareneingang. In Tabelle 6.6 sehen Sie eine Kurzübersicht über alle vom System geprüften Toleranzschlüssel. In der Dokumentation zum zuvor genannten Customizing-Punkt finden Sie zusätzlich ausführliche Erläuterungen zu den einzelnen Schlüsseln.

Toleranzschlüssel	Bedeutung
AN	Betragshöhe Position ohne Bestellbezug
AP	Betragshöhe Position mit Bestellbezug
BD	Kleindifferenzen automatisch ausbuchen
BR	prozentuale Bestellpreismengenabweichung (RE vor WE)
BW	prozentuale Bestellpreismengenabweichung (RE nach WE)
DQ	Überschreiten Betrag Mengenabweichung
DW	Mengenabweichung bei Wareneingangsmenge = null
KW	Preisabweichung Bezugsnebenkosten
LA	Betragshöhe Limitbestellung
LD	Zeitüberschreitung Limitbestellung
PP	Preisabweichung
PS	Preisabweichung Schätzpreis
ST	Terminabweichung
VP	V-Preisabweichung

Tabelle 6.6 Übersicht über Toleranzschlüssel

6.2.2 Nutzung der Erweiterung

Auch die Erweiterung `MM08R002` soll wieder anhand eines kleinen Beispiels erläutert werden. Nehmen wir an, Sie haben die Toleranzgrenzen für Preisabweichungen zur Bestellung (Toleranzschlüssel PP) so eingestellt, dass nur zwei Prozent Abweichung nach oben und unten erlaubt sind. Sie haben nun jedoch eine Warengruppe, bei der die Preise für die Waren starken Marktschwankungen unterliegen, daher soll in diesem Fall eine Abweichung von fünf Prozent erlaubt sein. Sie benötigen zur Umsetzung daher zunächst einmal die Information, welcher Warengruppe das Material einer Rechnungsposition angehört, und müssen dann abhängig davon die Grenzen neu defi-

nieren. Leider erhalten Sie diese Information jedoch in einem anderen User-Exit als dem User-Exit, der verwendet wird, um die Grenzwerte zu überschreiben.

Werfen Sie einen Blick auf Tabelle 6.7: Wenn Sie eine Position in der Eingangsrechnung erfassen und eine Abweichung auftritt, wird abhängig von der Art der Abweichung zunächst der `EXIT_SAPLMR1M_001` oder `EXIT_SAPLMRMP_001` aufgerufen. Sie müssen die Kopf- und Positionsdaten in die globalen Daten der Funktionsgruppe kopieren. Anschließend startet der `EXIT_SAPLMRMC_001`, in dem Sie die Toleranzgrenzen neu definieren können. Über die globalen Daten der Funktionsgruppe können Sie hier wieder auf die Kopf- und Positionsdaten zugreifen.

User-Exit	Beschreibung
`EXIT_SAPLMR1M_001`	Wird für jede Position bei Preis- und Mengenabweichungen durchlaufen und enthält Kopf- und Positionsdaten der Position.
`EXIT_SAPLMRMP_001`	Wird für jede Position bei Termin- und Betragsabweichungen durchlaufen und enthält Kopf- und Positionsdaten der Position.
`EXIT_SAPLMRMC_001`	Wird für jede Position und je relevanten Toleranzschlüssel einmal durchlaufen und bietet die Möglichkeit, die Toleranzgrenzen neu zu definieren.

Tabelle 6.7 User-Exits der Erweiterung MM08R002

Mit der folgenden Anleitung setzen Sie die Theorie in die Praxis um:

1. Legen Sie zunächst ein neues Projekt in Transaktion CMOD an, und nehmen Sie die Erweiterung `MM08R002` auf. Sichern Sie anschließend das Projekt, und wechseln Sie dann in die Komponenten.
2. Gehen Sie jetzt in den `EXIT_SAPLMR1M_001`, und wählen Sie dort den Menüpunkt Springen • Globale Daten. Sie können die globalen Daten eines User-Exits nicht direkt ändern, dies wäre eine Modifikation, aber Sie finden in den globalen Daten eine Zeile mit dem Inhalt `INCLUDE ZXM08TOP`. Dieses Include liegt im Kundennamensraum und kann so modifikationsfrei von Ihnen angepasst werden.

   Klicken Sie doppelt auf den Namen des Includes, um diesen anzulegen. Definieren Sie anschließend zwei Strukturen zur Aufnahme der Kopf- und Positionsdaten aus der Schnittstelle des Exits. Die Datentypen können Sie aus der Schnittstelle des User-Exits übernehmen (siehe Listing 6.12).

```
*&---------------------------------------------------------------*
*&  Include           ZXM08TOP
*&---------------------------------------------------------------*
* Globale Daten: Kopf- und Positionsdaten
DATA: zz_rbkpv TYPE MRM_RBKPV,
      zz_drseg TYPE MMCR_DRSEG.
```

Listing 6.12 Globale Daten der Erweiterung

3. Gehen Sie nun zurück in den User-Exit, und legen Sie dort das Include `ZXM08U04` durch einen Doppelklick an. Übertragen Sie nun einfach die Daten aus der Schnittstelle in die soeben definierten globalen Strukturen (siehe Listing 6.13).

```
*&---------------------------------------------------------------*
*&  Include           ZXM08U04 (EXIT_SAPLMR1M_001)
*&---------------------------------------------------------------*
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_RBKPV) TYPE  MRM_RBKPV
*"     VALUE(I_YDRSEG) TYPE  MMCR_DRSEG
*"  EXCEPTIONS
*"      CALL_FAILURE
*"----------------------------------------------------------------
zz_rbkpv = i_rbkpv.
zz_drseg = i_ydrseg.
```

Listing 6.13 User-Exit EXIT_SAPLMR1M_001

4. Wechseln Sie nun in den User-Exit `EXIT_SAPLMRMP_001`, und legen Sie das Include `ZXM08U17` an. Übertragen Sie auch hier die Daten der Schnittstelle in die globalen Daten. Das Coding ist identisch mit Listing 6.13.
5. Zuletzt gehen Sie in den `EXIT_SAPLMRMC_001` und legen dort das Include `ZXM08U19` an. In diesem User-Exit werden zwei Importparameter geliefert, der Buchungskreis und der Toleranzschlüssel. Sie sollten daher Ihre Neudefinition der Grenzen abhängig von diesen Parametern gestalten.

   Die neuen Toleranzgrenzen können Sie anhand von Tabelle 6.8 festlegen. Die Felder `E_XW1JA`, `E_XW2JA`, `E_XP1JA` und `E_XP2JA` bestimmen, ob die zugehörige Grenze geprüft werden soll, identisch mit den Optionen Nicht Prüfen und Grenze prüfen im Customizing (siehe Abbildung 6.8). Setzen Sie das Flag nicht, ist auch hier eine unbegrenzte Abweichung erlaubt.
6. Sie müssen das Flag `E_CHECK` setzen, wenn Ihre Änderungen angewendet werden sollen, anderenfalls werden weiterhin die Einstellungen aus dem Customizing verwendet.

Parameter	Bedeutung
E_WERT1	Untergrenze absolut
E_XW1JA	Untergrenze absolut prüfen
E_WERT2	Obergrenze absolut
E_XW2JA	Obergrenze absolut prüfen
E_PROZ1	Untergrenze prozentual
E_XP1JA	Untergrenze prozentual prüfen
E_PROZ2	Obergrenze prozentual
E_XP2JA	Obergrenze prozentual prüfen
E_CHECK	Kennzeichen: neue Grenzen aus User-Exit verwenden

Tabelle 6.8 Parameter des EXIT_SAPLMRMC_001

Standardwerte aus Customizing einbinden

Wenn Sie E_CHECK setzen, sonst jedoch kein Feld füllen, bedeutet dies, dass grundsätzlich unbegrenzte Abweichungen erlaubt sind. Da in unserem Beispiel nur die prozentuale Grenze angepasst werden soll, können Sie optional alle anderen Werte aus dem Customizing (Tabelle T169G) nachlesen. Das Beispiel hierzu finden Sie in Listing 6.14.

```
*&---------------------------------------------------------------------*
*&  Include          ZXM08U19 (EXIT_SAPLMRMC_001)
*&---------------------------------------------------------------------*
DATA ls_t169g TYPE t169g.

* Abweichende Logik für alle Buchungskreise
* und Preisabweichung (Toleranzschlüssel PP)
* für Warengruppe 007
IF i_tolsl = 'PP' AND
   zz_drseg-matkl = '007'.
* Standardwerte aus Customizing holen
  SELECT SINGLE * FROM t169g INTO ls_t169g
        WHERE bukrs = i_bukrs AND
              tolsl = i_tolsl.
* Absolute Grenzen aus Customizing zuordnen
  e_wert1 = ls_t169g-wert1.
  e_wert2 = ls_t169g-wert2.
  e_xw1ja = ls_t169g-xw1ja.
  e_xw2ja = ls_t169g-xw2ja.
* Prozentuale Abweichung erhöhen
  e_proz1 = 5.
  e_xp1ja = 'X'.
  e_proz2 = 5.
  e_xp2ja = 'X'.
```

```
* Änderung verwenden
  e_check = 'X'.
ENDIF.
```

Listing 6.14 User-Exit EXIT_SAPLMRMC_001

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z