Ralph Ellerbrock, Oliver Isermann

SAP® CRM praxisgerecht erweitern

Galileo Press

Bonn • Boston

Auf einen Blick

Inhalt

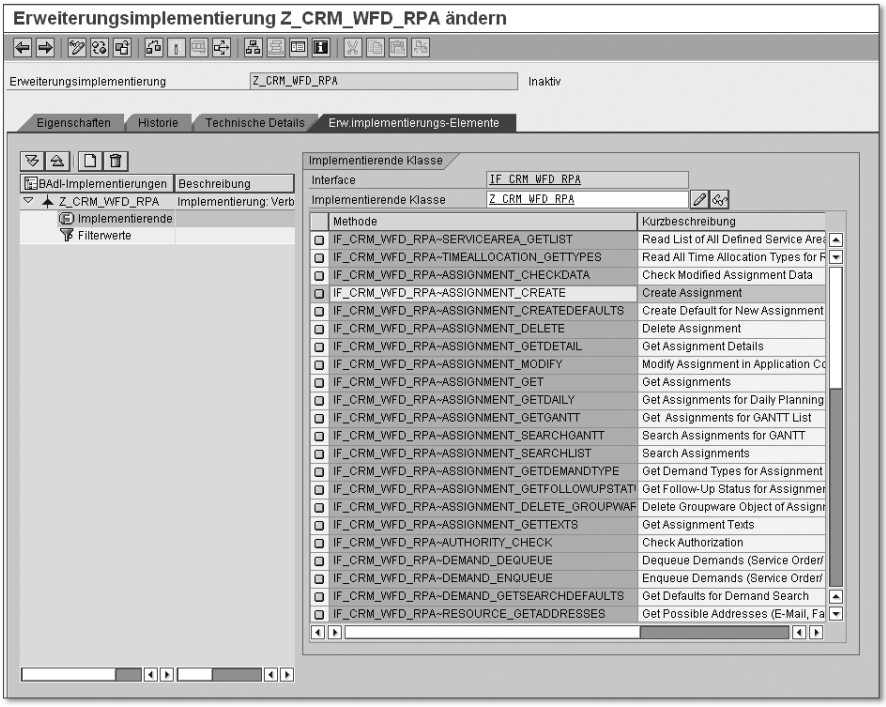


Abbildung 2.11 Erweiterungsspot-Methoden

Wie für klassische BAdIs gilt auch hier: Vergessen Sie nicht, sowohl Ihre Methode als auch die gesamte Implementierung zu aktivieren! Das Häkchen Implementierung ist aktiv auf dem Detailbild zeigt *nicht*, ob die Implementierung aktiviert ist. Wie Sie im oberen Bereich der Abbildung 2.9 sehen können, ist die Implementierung selbst im Beispiel noch nicht aktiviert!

2.2 Event Handler

Im Bereich der One-Order-Belegverarbeitung in SAP CRM gibt es die Möglichkeit, kundeneigene Logik an sogenannte *Events* zu koppeln, die zu bestimmten Systemzuständen und Zeitpunkten standardmäßig durch das CRM-System ausgelöst werden. Ein solcher Event ist beispielsweise der Zeitpunkt, zu dem ein Beleg im CRM-System gesichert wird.

Der CRM Event Handler ist ein sehr mächtiges Werkzeug und versteckt sich in der zunächst etwas unübersichtlich wirkenden Transaktion CRMV_EVENT. Wenn Sie diese Transaktion öffnen, finden Sie an oberster Stelle den Button Hinweis – Bitte lesen. Diese Bitte sollten Sie unbedingt ernst neh-

men, da die dort hinterlegte Hilfe bereits viele Fragen beantwortet. Grundsätzlich geht es darum, kundeneigene Verarbeitungslogik innerhalb der Standard-Programmabläufe unterzubringen. Die Verwendung eines Events bietet sich immer dann an, wenn es für Ihre gewünschte Logik kein passendes BAdI gibt. Ansonsten ist die Verwendung eines BAdIs vorzuziehen.

Betrachten wir zunächst den Bereich ZUORDNUNGEN innerhalb der Transaktion CRMV_EVENT (siehe Abbildung 2.12). Wählen Sie einen VORGANGSTYP (Business-Objekt-Typ, z. B. *Verkauf*), und starten Sie die Selektion über den Button CALLBACK ZU TYP/OBJ./EREIGNIS.

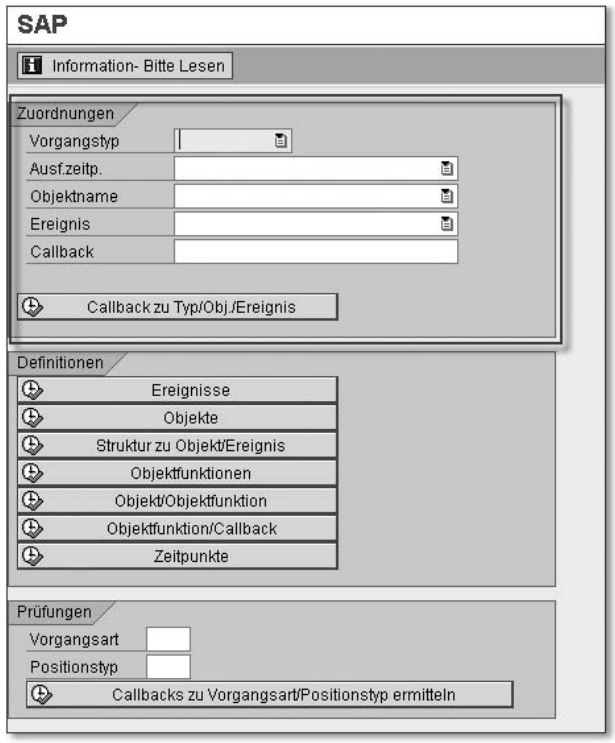


Abbildung 2.12 Transaktion CRMV_EVENT

Es öffnet sich eine Tabelle mit vielen hinterlegten Events bzw. Callbacks für den gewählten Business-Objekt-Typ. Unter *Callback* wird in diesem Zusammenhang ein Funktionsbaustein verstanden, der an definierten Stellen im Programmablauf aufgerufen wird. Die Stelle im Programmablauf wird im Wesentlichen durch folgende Parameter beschrieben:

- **Ausführungszeitpunkt (**Ausf.zeitp.**)**
  Der Ausführungszeitpunkt ist ein Zeitpunkt innerhalb der CRM-Vorgangsbearbeitung, beispielsweise Ende der Kopfbearbeitung, Ende der Positionsbearbeitung oder auch Sofort.
- **Objektname**
  Der Objektname definiert das technische Objekt, zu dem der Event aufgerufen wird, beispielsweise Partner für die Partnerfindung/-verarbeitung.
- **Ereignis**
  Das Ereignis bestimmt letztlich den eigentlichen Event, also den *Trigger*. Am häufigsten wird hier sicher das Ereignis After_Change verwendet, also der Zeitpunkt nach einer Änderung.

Ein Callback-Baustein mit der Schlüsselkombination Objekt Partner, Ereignis After_Change und Zeitpunkt Sofort würde also immer dann (sofort) ausgeführt, wenn in einem CRM-Geschäftsvorgang mindestens ein Partner geändert wird. Weitere Steuerungsmöglichkeiten sehen Sie, wenn Sie nun noch auf einen beliebigen Eintrag in der Event-Tabelle doppelklicken (siehe Abbildung 2.13):

- **Priorität**
  Die Priorität bestimmt die Reihenfolge der Abarbeitung bei ansonsten gleicher Schlüsselkombination.
- **Attribut**
  Ein Attribut definiert das Objekt genauer, z. B. der Partnerfunktionstyp 0001 beim Objekt Partner. Für den Ablauf würde dies bedeuten, dass nur Änderungen am Partnerfunktionstyp 0001 (Auftraggeber) den Event auslösen. Als Wildcard lässt sich hier »<*>« verwenden.
- **Funktion zum Belegkopf ausführen**
  Der Event wird auf Kopfebene ausgelöst.
- **Funktion zur Belegposition ausführen**
  Der Event wird auf Positionsebene ausgelöst.
- **Aufruf Callback**
  Diese Option steuert die Parameter, die an den Callback-Baustein übergeben werden. Sie können hier beispielsweise steuern, ob Sie alte und neue Daten des betreffenden Objekts in die Funktionsbausteinschnittstelle übergeben wollen. Bei Änderungen können Sie so sehen, welche Felder wie geändert wurden. Achtung: Die Funktionsbausteinschnittstelle muss zur Aufrufart passen!

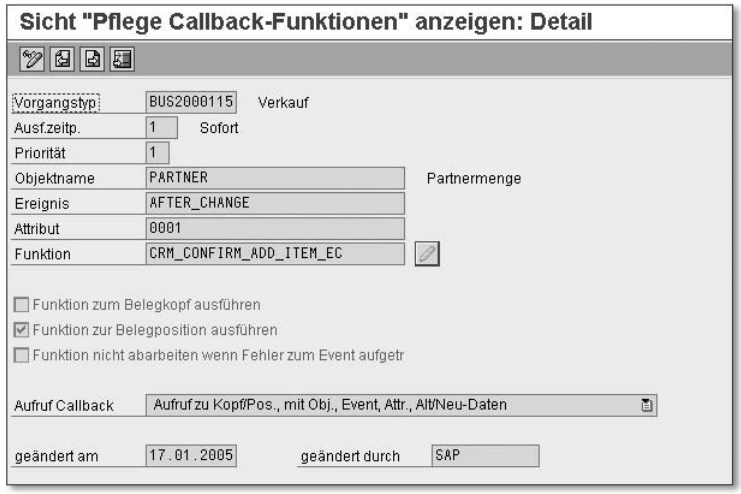


Abbildung 2.13 Detailansicht eines Callback-Events

Seit SAP CRM 5.0 gibt es als Ergänzung zur Transaktion CRMV_EVENT den View `CRMV_EVENT_CUST`, den Sie über die Transaktion SM30 oder über das Customizing erreichen (IMG-Pfad: Customer Relationship Management • Vorgänge • Grundeinstellungen • Event-Handler-Tabelle bearbeiten). Dort werden analog zur Transaktion CRMV_EVENT die *kundeneigenen* Event-Callbacks hinterlegt (der Aufbau der Tabellen unterscheidet sich dabei nicht). Diese Trennung sorgt dafür, dass die Verwendung kundeneigener Callback-Bausteine modifikationsfrei ist. Streng genommen, wäre zwar auch der Eintrag eigener Callbacks in der Standard-Tabelle keine Modifikation, dennoch können Sie nie sicher sein, ob beim nächsten Update nicht vielleicht ein SAP-eigener Event hinzugefügt wird, der die gleiche Schlüsselkombination benutzt wie Ihr Event. Wichtig ist also, dass Sie eigene Events immer im View `CRMV_EVENT_CUST` (bzw. in der Tabelle `CRMC_EVENT_CUST`) hinterlegen.

Tipp: Vorhandene Callback-Bausteine kopieren

Callback-Funktionsbausteine haben prinzipiell immer die gleiche Schnittstelle, die sich jedoch je nach Aufrufart im Event unterscheidet. Wenn Sie eigene Callback-Bausteine anlegen, müssen Sie also darauf achten, dass die Schnittstelle Ihres Funktionsbausteins zur Aufrufart passt (siehe Abbildung 2.14). Am einfachsten ist es, wenn Sie sich aus den Standard-Callbacks einen passenden Funktionsbaustein (mit gleicher Aufrufart) heraussuchen, diesen kopieren und ihn dann Ihren Anforderungen anpassen.

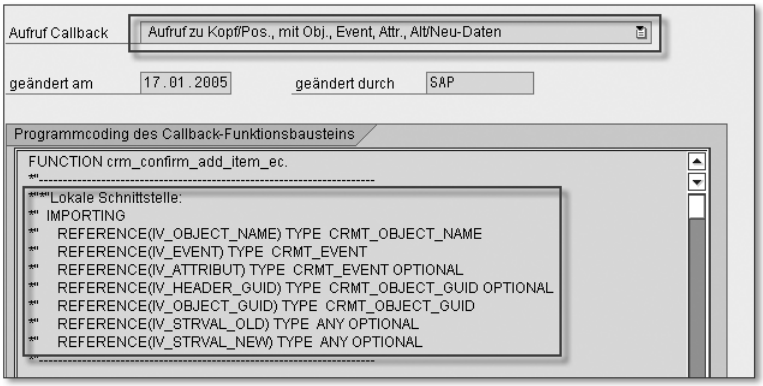


Abbildung 2.14 Schnittstelle eines Callback-Bausteins (passend zur Aufrufart)

Bevor Sie nun einen eigenen Callback anlegen können, müssen Sie noch einen weiteren Schritt durchführen: Ihr Funktionsbaustein muss als Callback *registriert* werden. Diese Registrierung führen Sie ebenfalls in der Transaktion CRMV_EVENT durch (auch für kundeneigene Bausteine!). Verwenden Sie dazu den Button Objektfunktionen/Callback aus dem Bereich Definitionen. Fügen Sie für Ihren Funktionsbaustein und das Objekt, für das Sie den Baustein einsetzen möchten, einen neuen Eintrag hinzu. Callback-Bausteine können nur für ein Objekt registriert werden. Wenn Sie Ihre Funktionalität für mehrere Objekte brauchen, können Sie ausprobieren, ob die bereits vorhandene Registrierung ausreicht (manchmal ist dies der Fall), oder Sie müssen den Baustein kopieren und unter einem anderen Namen erneut registrieren.

Sobald Sie Ihren Callback-Funktionsbaustein registriert haben, können Sie über den View `CRMV_EVENT_CUST` eigene Callbacks definieren. Dabei ist allerdings etwas Fingerspitzengefühl gefragt: Kundeneigene Events müssen immer ausgiebig getestet werden, weil es durchaus möglich ist, dass Sie mit Ihrem Event den Standard-Ablauf durcheinanderbringen (beispielsweise durch einen rekursiven `CRM_ORDER_MANTAIN`-Aufruf, eine Anweisung `COMMIT WORK` zum falschen Zeitpunkt o. Ä.). Zudem können Sie nicht beliebige Operationen zu jedem Zeitpunkt/Objekt durchführen. Hier hilft im Prinzip nur Ausprobieren: Setzen Sie als Erstes einen Breakpoint in Ihren Callback-Baustein, und testen Sie in der Vorgangsbearbeitung, ob er überhaupt »angesprungen« (durchlaufen) wird. Dann tasten Sie sich langsam weiter vor zu der Operation/Funktion, die Sie durchführen möchten.

```
      NUMBER                  = LV_NEW_NUMBER
    EXCEPTIONS
      INTERVAL_NOT_FOUND      = 1
      NUMBER_RANGE_NOT_INTERN = 2
      OBJECT_NOT_FOUND        = 3
      QUANTITY_IS_0           = 4
      QUANTITY_IS_NOT_1       = 5
      INTERVAL_OVERFLOW       = 6
      BUFFER_OVERFLOW         = 7
      OTHERS                  = 8.
  CHECK SY-SUBRC = 0.
* 5. Build and assign external campaign ID
  CONCATENATE LC_C
              LC_SEP
              LV_VKORG
              LC_SEP
              CS_ATTRIBUTES_NEW-CAMP_TYPE
              LC_SEP
              LV_NEW_NUMBER
              INTO LV_EXTERNAL_ID.
  CS_ATTRIBUTES_NEW-EXTERNAL_ID = LV_EXTERNAL_ID.
ENDMETHOD.
```

Listing 3.2 Methode SET_ATTRIBUTES_BEFORE

3.2.2 Erweiterung der Unvollständigkeitsprüfung für Kampagnen

Im Rahmen dieses Praxisbeispiels geht es um die Erweiterung der Unvollständigkeitsprüfung von Marketingelementen; speziell um die Verprobung von Kampagnen. Dies ist in der Praxis häufig ein Thema, da es standardmäßig möglich ist, auch sehr rudimentär gepflegte Kampagnen anzulegen und freizugeben, ohne dass der Erfasser auf initiale Felder aufmerksam gemacht wird. Dadurch kann es passieren, dass beispielsweise Kampagnen ohne elementare Angaben wie z. B. eine Gültigkeitsperiode angelegt und freigegeben werden, was in der Folge zu Problemen führen kann. Anders als beispielsweise im One-Order-Umfeld bietet das für den Marketingkontext zur Verfügung stehende Customizing leider keine Möglichkeiten, entsprechende Prüfungen im Standard abzubilden.

Grundsätzlich können Sie im Web UI per Konfiguration des entsprechenden Views entsprechende Feldeigenschaften festlegen. Sie können dort z. B. definieren, ob ein Feld eingabebereit sein darf und ob es sich um ein Mussfeld handeln soll. Abbildung 3.25 zeigt eine entsprechende Sicht für das Feld PLANSTART in einer Kampagne.

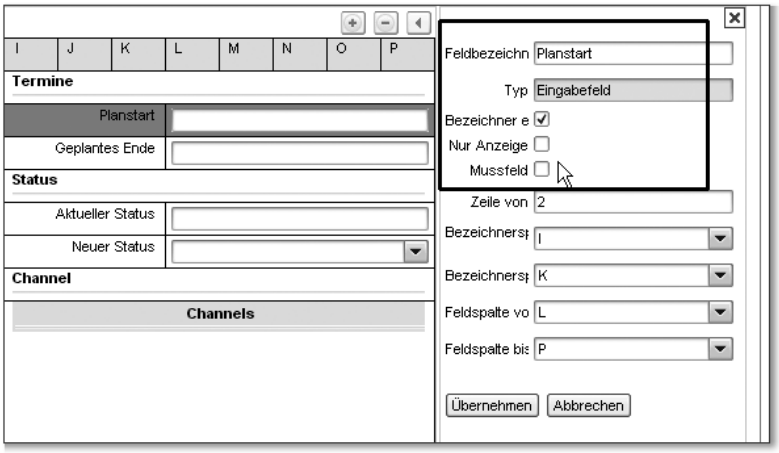


Abbildung 3.25 Festlegung von Feldeigenschaften in der Web-UI-Konfiguration

Aber: Diese Änderungen gelten dann *generell* für die entsprechende Sicht. Eine gezielte Steuerung (z. B. »Feld X soll nur für Kampagnenart Y oder nur für Marketingorganisation Z ein Mussfeld darstellen«) ist durch diese Implementierungsform nicht abgedeckt und damit in der Praxis oft nicht einsetzbar. Für unser Beispiel nehmen wir als zusätzliche Anforderung an, dass es möglich sein soll, bei der Unvollständigkeitsprüfung pro Feld zu definieren, ob ein initialer Wert eine Warnung oder einen Fehler hervorrufen soll.

Speziell für die hier beschriebene Anforderung bietet SAP CRM 7.0 für den Marketingkontext ein interessantes BAdI (`CRM_MKTPL_OL_OBJ`), das Sie bereits aus dem letzten Abschnitt kennen. Die BAdI-Methode `CHECK_ATTRIBUTES` verfügt über großzügig bemessene Importparameter, sodass ebenfalls sämtliche Feldwerte eines Marketingelements für eine parametrisierte Unvollständigkeitsprüfung verwendet werden können. Die Exportparameter `ET_MESSAGE_LOG` sowie `EV_HAS_ERRORS` erlauben die Ausgabe der gewünschten Meldungen.

Tipp: Methode CHANGE_FIELD_ATTRIBUTES

Wenn Ihnen eine Kennzeichnung als reines Mussfeld (ohne Unterscheidung zwischen Warn- und Fehlermeldung) reicht, können Sie auch die Methode `CHANGE_FIELD_ATTRIBUTES` verwenden. Sie funktioniert unabhängig vom verwendeten UI und bietet Ihnen noch zusätzlich die Möglichkeit, Felder gezielt für die Eingabe zu sperren (nur Anzeigefunktion) bzw. ganz auszublenden.

In Abbildung 3.26 ist der hier vorgestellte Lösungsansatz in der Übersicht dargestellt. Die Grundidee besteht darin, für bestimmte Felder in einer Kampagne zu prüfen, ob der Benutzer einen Wert gepflegt hat. Wenn ein solches Mussfeld noch keinen Inhalt aufweist, erhält der Benutzer eine entsprechende Meldung. Um einen in der Praxis möglichst hohen Nutzwert für diese Erweiterung zu erzielen, wird die Prüfung in zweierlei Hinsicht parametrisierbar gestaltet:

1. Je Kampagnenart kann definiert werden, welche Felder als Mussfelder geprüft werden sollen.
2. Die Systemreaktion auf ein leeres Mussfeld kann als Warn- oder Fehlermeldung erfolgen.

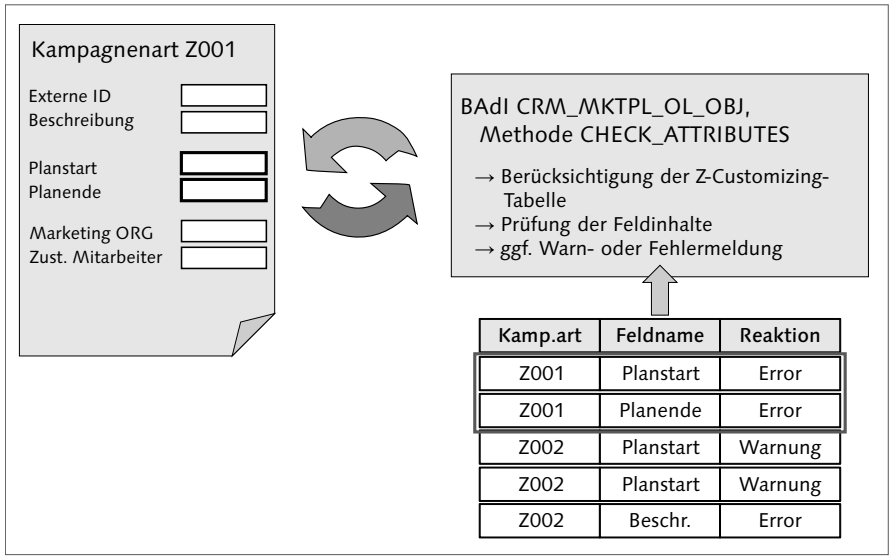


Abbildung 3.26 Lösungsansatz zur Unvollständigkeitsprüfung in Kampagnen

Parametrisierung der Unvollständigkeitsprüfung

Um die versprochene Parametrisierbarkeit der Unvollständigkeitsprüfung zu ermöglichen, bietet sich auch hier die Erstellung einer Z-Customizing-Tabelle an. Wie bereits erwähnt, soll je Kampagnenart auf Feldebene definierbar sein, wie das System auf fehlende Eingaben des Benutzers reagiert. Als mögliche Auswahl zu prüfender Felder stehen Ihnen hier die interne und externe Sicht auf die Attribute einer Marketingkampagne zur Verfügung. Wie Sie auch an den Importparametern `IS_ATTRIBUTES` und `IS_ATTRIBUTES_EXT` der Methode `CHECK_ATTRIBUTES` erkennen können, unterscheidet das CRM-

System zwischen der systeminternen und -externen Darstellung von Feldinhalten eines Marketingelements: Die Taktik (Feld TAKTIK) einer Kampagne beispielsweise wird systemintern im Feld TACTICS (Schlüssel des entsprechenden Customizing-Eintrags) gespeichert, dem Benutzer aber im Feld TACTICSTX (Bezeichnung der Taktik) dargestellt. Im Normalfall können Sie davon ausgehen, dass beide Felder gefüllt sind, wenn der Benutzer eine Taktik ausgewählt hat, und leer, wenn noch keine Taktik ausgewählt bzw. eine bereits eingegebene Taktik wieder entfernt wurde.

Nach unseren Erfahrungen funktioniert die Prüfung auf Felder der internen Sicht im Web UI zuverlässiger: Bei einigen Feldern (z. B. dem Feld PRIORITÄT) kam es vor, dass der Benutzer zwar schon einen zuvor gewählten Feldwert wieder entfernt hatte, vom System jedoch noch der alte Wert für den Importparameter IS_ATTRIBUTES_EXT ermittelt wurde. Im Zweifelsfall gilt auch hier: Ausprobieren!

Für unser Praxisbeispiel werden wir ausschließlich Felder der internen Sicht (IS_ATTRIBUTES) verwenden. In Abbildung 3.27 finden Sie einen Vorschlag dafür, wie die Z-Customizing-Tabelle definiert werden kann.

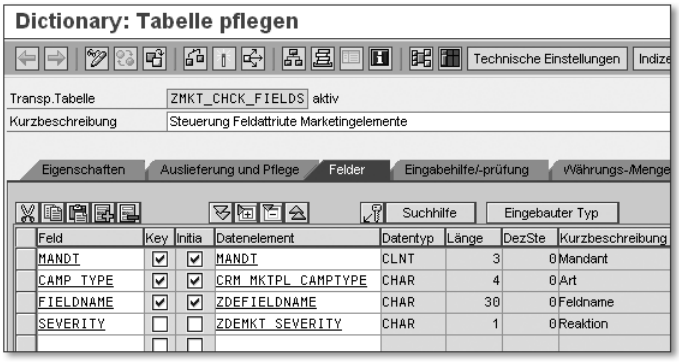


Abbildung 3.27 Definition der Z-Customizing-Tabelle zur Unvollständigkeitsprüfung in Marketingkampagnen

Um die Pflege von Tabelleneinträgen zu erleichtern, sollten Sie für alle relevanten Felder (CAMP_TYPE, FIELDNAME und SEVERITY) eine Eingabehilfe zur Verfügung stellen. Durch die Verwendung des Standard-Datenelements für die Kampagnenart ist dies bereits für das Feld CAMP_TYPE gegeben. Um dem Benutzer nur für den Kampagnenkontext relevante Felder im Feld FIELDNAME anzubieten, können Sie als pragmatische und einfache Möglichkeit ein eigenes Datenelement mit einer eigenen Domäne definieren, in der Sie ausge-

wählte Feldnamen des Importparameters `IS_ATTRIBUTES` als Domänenfestwerte zur Verfügung stellen (siehe Abbildung 3.28).

Abbildung 3.28 Domänenfestwerte als Auswahlhilfe für das Feld FIELDNAME

Für das Feld `SEVERITY`, das die Art der Systemnachricht (Warn- oder Fehlermeldung) festlegt, können Sie analog verfahren (siehe Abbildung 3.29).

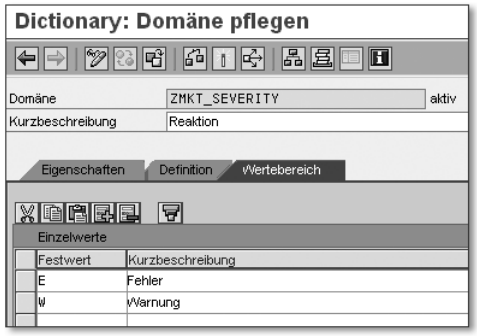


Abbildung 3.29 Domänenfestwerte als Auswahlhilfe für das Feld SEVERITY

Implementierung der BAdI-Methode CHECK_ATTRIBUTES

Bitte lesen Sie unbedingt Abschnitt 3.2.1, »Automatische Vergabe von Kampagnen-IDs«, bevor Sie die hier beschriebene Implementierung vornehmen! Dort ist ausführlich beschrieben, welche Schritte durchgeführt werden müssen, damit aus Ihrer Erweiterung keine Modifikation wird und auch keine anderen Standard-Prozesse bzw. -funktionalitäten beeinträchtigt werden. Sie finden dort auch eine Erläuterung dazu, welche Filtereinstellungen Sie auswählen müssen, damit ausschließlich Kampagnen auf Unvollständigkeit geprüft werden.

Im folgenden Listing 3.3 finden Sie einen Implementierungsvorschlag für die Methode `CHECK_ATTRIBUTES`. Da wir im Falle eines nicht gefüllten Mussfeldes eine Warn- oder Fehlermeldung ausgeben wollen, bietet sich als Ergänzung dazu die Anlage einer eigenen Systemmeldung in der Transaktion SE91 an (siehe Abbildung 3.30).

Abbildung 3.30 Erstellung einer eigenen Nachrichtenklasse und Nachricht

Hier noch einige Erläuterungen zu dem Implementierungsvorschlag aus Listing 3.3:

1. Die Unvollständigkeitsprüfung wird erst dann durchgeführt, wenn der Benutzer sich für eine Kampagnenart entschieden hat.

2.1 Erst mit dieser Information werden die zu prüfenden Felder aus der Z-Customizing-Tabelle einmalig ausgelesen und in einer globalen internen Tabelle gespeichert. Die zu prüfenden Felder sowie die Systemreaktion stehen damit fest und brauchen nicht nochmals von der Datenbank gelesen zu werden. Die Verwendung globaler interner Tabellen setzt die Definition und Typisierung geeigneter Klassenattribute in Ihrer Z-Klasse voraus. Die Abbildungen 3.31 und 3.32 zeigen entsprechende Beispiele.

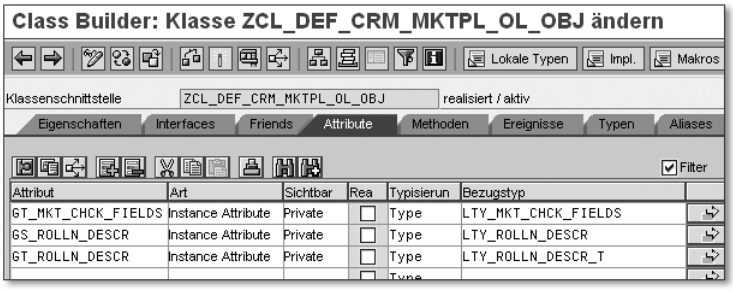


Abbildung 3.31 Definition globaler Variablen bzw. Tabellen

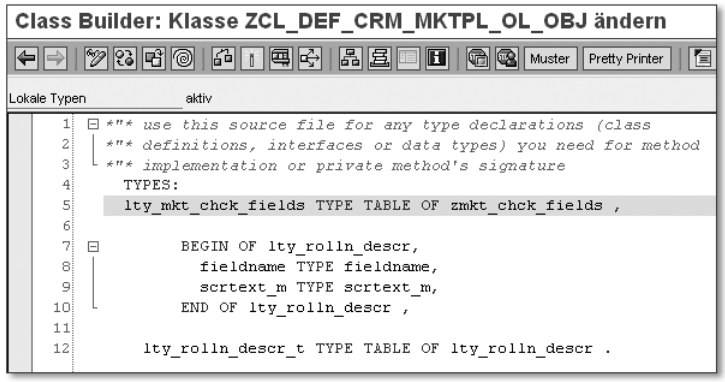


Abbildung 3.32 Klassenlokale Typdefinitionen

2.2 Um dem Benutzer eine möglichst aussagefähige Meldung zu liefern, wird zunächst einmalig für jedes zu prüfende Feld das zugehörige Datenelement anhand der Tabelle `DD03L` ermittelt und anschließend dessen Bezeichnung in der Anmeldesprache des Benutzers aus der Tabelle `DD04T` hinzugelesen. Dies ist zwar keine 100 %-Lösung, da diese Bezeichnung von derjenigen im Web UI leicht abweichen kann, in der Praxis hat sich diese Methode allerdings als ausreichend präzise bewährt. Alternativ wäre es auch möglich, für jedes geprüfte Feld die genaue Bezeichnung über die UI-Eigenschaften zu ermitteln. Um Performance bzw. Datenbankzugriffe zu sparen, werden alle so ermittelten Kombinationen aus Feldname und Bezeichnung des zugehörigen Datenelements ebenfalls in einer globalen internen Tabelle gespeichert.

3.1 Für die eigentliche Prüfung wird für jedes in der Z-Customizing-Tabelle angegebene Feld zunächst geprüft, ob ein initialer Wert zu einer Warn- oder Fehlermeldung führen soll.

3.2 Anschließend wird je Feld der aktuelle Wert der Kampagne gelesen. Dies erfolgt der Einfachheit halber durch die Verwendung eines Feldsymbols. Wichtig: In diesem Implementierungsvorschlag werden derzeit nur Felder aus dem Importparameter `is_attributes` geprüft. Dies können Sie natürlich ändern – erweitern Sie dann aber auch entsprechend die Feldauswahl bzw. die Festwerte in der Domäne `ZFDNAME` (siehe dazu Abbildung 3.28).

3.3 Wird ein leeres Mussfeld ermittelt, wird zunächst die Bezeichnung des verknüpften Datenelements aus der internen Tabelle `gt_rolln_descr` gelesen.

3.4 Anschließend wird eine entsprechende Warn- oder Fehlermeldung in den Exportparameter ET_MESSAGE_LOG eingestellt und der Parameter EV_HAS_ERRORS auf den Wert abap_true gesetzt.

```
METHOD if_ex_crm_mktpl_ol_obj~check_attributes.
*** 0. Declaration:
  CONSTANTS: lc_msgid TYPE c
             LENGTH 18
             VALUE 'Z_MKT_CHECK_FIELDS'.
  DATA: lt_mkt_chck_fields TYPE
                           STANDARD TABLE OF zmkt_chck_fields,
        ls_mkt_chck_fields TYPE zmkt_chck_fields,
        ls_badi_message    TYPE crms_mktgs_badi_msg,
        lt_rollname        TYPE STANDARD TABLE OF rollname,
        lv_rollname        TYPE rollname,
        lv_scrtext_m       TYPE scrtext_m,
        BEGIN OF ls_rolln_descr,
          fieldname TYPE fieldname,
          scrtext_m TYPE scrtext_m,
        END OF ls_rolln_descr.
  FIELD-SYMBOLS: <field_value> TYPE ANY.
*** 1. Check: User has decided for a campaign type:
  CHECK is_attributes-camp_type IS NOT INITIAL.
*** 2. Preparations
*** 2.1 Read only once own customizing table --> fields to
***     check and save them in global table:
  IF gt_mkt_chck_fields IS INITIAL.
    SELECT * FROM zmkt_chck_fields
             INTO TABLE gt_mkt_chck_fields
             WHERE camp_type = is_attributes-camp_type.
  ENDIF.
  IF gt_mkt_chck_fields IS NOT INITIAL.
*** 2.2  Read only once data element descriptions for every
***      field to check and save them in global table:
    IF gt_rolln_descr IS INITIAL.
      LOOP AT gt_mkt_chck_fields INTO ls_mkt_chck_fields.
        SELECT SINGLE rollname FROM dd03l INTO lv_rollname
                 WHERE tabname = 'CRMS_MKTPL_OL_ATTRIBUTES'
                 AND fieldname = ls_mkt_chck_fields-fieldname.
        SELECT SINGLE scrtext_m FROM dd04t INTO lv_scrtext_m
                 WHERE rollname = lv_rollname
                 AND ddlanguage = sy-langu.
        IF  ls_mkt_chck_fields-fieldname IS NOT INITIAL
        AND lv_scrtext_m IS NOT INITIAL.
      ls_rolln_descr-fieldname = ls_mkt_chck_fields-fieldname.
```

```
      ls_rolln_descr-scrtext_m = lv_scrtext_m.
      INSERT ls_rolln_descr INTO TABLE gt_rolln_descr.
        ENDIF.
        CLEAR: ls_mkt_chck_fields-fieldname,
               lv_scrtext_m.
      ENDLOOP.
    ENDIF.
    LOOP AT gt_mkt_chck_fields INTO ls_mkt_chck_fields.
*** 3. Perform check for every relevant field:
*** 3.1  Check: Either warning or error message must be customized:
      IF ls_mkt_chck_fields-severity IS NOT INITIAL.
*** 3.2  Assign curr. value of requested campaign
***      field to field symb.:
        ASSIGN COMPONENT ls_mkt_chck_fields-fieldname
        OF STRUCTURE is_attributes TO <field_value>.
        IF <field_value> IS ASSIGNED
        AND <field_value> IS INITIAL.
*** 3.3  Relevant field was initial --> Read descriptions
***      for relevant data elements to prepare warning or
***      error message:
         READ TABLE gt_rolln_descr
         INTO ls_rolln_descr
         WITH KEY fieldname = ls_mkt_chck_fields-fieldname.
*** 3.4  Construct warning or error message:
         ev_has_errors = abap_true.
          ls_badi_message-msgid = lc_msgid.
          ls_badi_message-msgty = ls_mkt_chck_fields-severity.
          ls_badi_message-msgno = '000'.
          ls_badi_message-msgv1 = ls_rolln_descr-scrtext_m.
          INSERT ls_badi_message INTO TABLE et_message_log.
        ENDIF.
      ENDIF.
      CLEAR: ls_rolln_descr.
    ENDLOOP.
  ENDIF.
ENDMETHOD.
```

Listing 3.3 Unvollständigkeitsprüfung in Kampagnen

Das Ergebnis Ihrer Unvollständigkeitsprüfung sollte dann so aussehen wie in Abbildung 3.33: Das leere Feld PLANSTART hat eine Fehler- und das leere Feld TAKTIK eine Warnmeldung hervorgerufen.

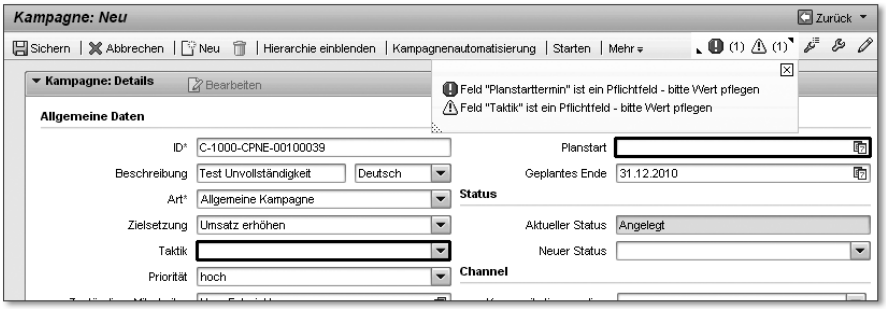


Abbildung 3.33 Ergebnis der eigenen Unvollständigkeitsprüfung

3.3 Marketingmerkmale

Marketingmerkmale sind im CRM-System das zentrale Instrument zur Klassifikation von Geschäftspartnern. Anhand dieser Klassifikationskriterien lassen sich im Segment Builder Zielgruppen aufbauen und anschließend zielgruppengerechte Kampagnen starten. Bei der Arbeit mit Marketingmerkmalen steht man jedoch immer wieder vor zwei Problemen: Zum einen werden Änderungen an den Marketingmerkmalen des Geschäftspartners im CRM-Standard leider nicht in den Änderungsbelegen fortgeschrieben, und zum anderen ist es sehr mühsam, alle zu klassifizierenden Geschäftspartner mit einem bestimmten Merkmal zu kennzeichnen. Für beide Probleme bietet dieser Abschnitt Lösungsansätze. Wir beschreiben, wie die Änderungsbelege des Geschäftspartners um die Marketingmerkmale ergänzt werden können, und stellen eine Methode vor, um Merkmalsbewertungen automatisiert aus CRM-Geschäftsvorgängen heraus vorzunehmen.

3.3.1 Änderungsbelege für Änderungen an Marketingmerkmalen

Um Änderungsbelege für Marketingmerkmale am Geschäftspartner anzuzeigen, bietet SAP den Standard-Report `CRM_MKTBP_ATTR_HISTORY`. Schöner wäre es jedoch, wenn die Änderungen direkt innerhalb der »normalen« Änderungshistorie eines Geschäftspartners sichtbar wären. Vor allem, wenn es sich um eine überschaubare Anzahl änderungsbelegrelevanter Merkmale handelt, ist die im Folgenden beschriebene Erweiterung gut geeignet, um dieses Ziel zu erreichen.

Als Beispiel nutzen wir ein Marketingmerkmal, das angibt, ob ein Kunde eine bestimmte Produktart gekauft hat, z. B. eine Pumpe. Es gibt zwei verschiedene Pumpenarten, A und B, somit haben wir es mit einem einstelligen Mar-

Abbildung 4.40 Pop-up-Fenster bei mehreren gültigen Kampagnen

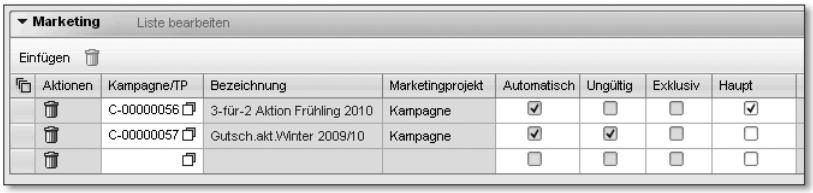


Abbildung 4.41 Ergebnis der automatischen Kampagnenfindung

4.7 Sales Order Management

In diesem Abschnitt möchten wir Ihnen drei praktische Features im Bereich des Sales Order Managements vorstellen, die nach unserer Erfahrung oft nachgefragt werden: Zunächst zeigen wir Ihnen, wie Sie in CRM-Vorgängen automatisch Fakturasperren setzen können. Anschließend stellen wir Ihnen einen Lösungsansatz vor, mit dessen Hilfe Sie eine eigene Unvollständigkeitsprüfung für Felder im CRM-Vorgang durchführen können, die nicht über die Standard-Funktionalität abgedeckt sind. Abschließend demonstrieren wir, wie Sie beim Upload von CRM-Aufträgen in das ERP-System dafür sorgen können, dass der Anwenderstatus auf Kopfebene synchron gehalten wird.

4.7.1 Automatisches Setzen von Fakturasperren

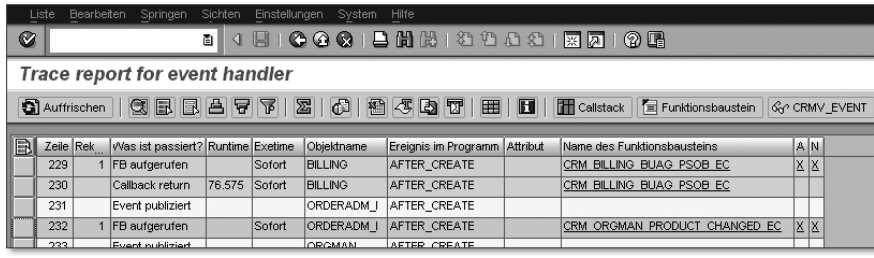
Jeder, der sich schon einmal mit der Implementierung von Retouren- bzw. Gutschriftprozessen auseinandersetzen durfte, kennt diese Anforderung: Wenn eine Gutschriftanforderung bzw. ein Retourenauftrag angelegt wird, sollen diese automatisch nach ihrer Erfassung mit einer Fakturasperre versehen werden. Das heißt, erst nach Herausnahme dieser Sperre kann der zugehörige Faktura- bzw. Gutschriftbeleg erzeugt werden. Bei zumeist auftragsbezogen fakturierten Gutschriftanforderungen ist dies häufig erwünscht, da normalerweise bereits nach dem Sichern des Belegs ein Fakturavorratseintrag vorhanden ist, der durch wenige Klicks – eventuell ungewollt bzw. ungeprüft –

als Gutschrift auf dem Konto des Kunden landet. Bei lieferbezogenen Retourengutschriften wird ein Fakturavorratseintrag zwar üblicherweise erst dann vom System gebildet, wenn der Wareneingang gebucht ist, sich die Ware also wieder im eigenen Hause befindet. Doch allein der physische Zugang sagt in der Praxis relativ wenig über den Zustand der retournierten Ware und damit auch über den Gutschriftbetrag aus, der dem Kunden erstattet werden soll.

Während es im ERP-System noch mittels Customizing der Verkaufsbelegart und des Positionstyps möglich war, neue Auftragspositionen automatisch mit einer Fakturasperre zu versehen, ist dies auf CRM-Seite im Standard nicht mehr möglich. Doch glücklicherweise bedeutet eine entsprechende Erweiterung keinen großen Aufwand:

- Durch eine Erweiterung der Eventsteuerung ist es möglich, direkt nach Erfassung einer Kundenauftragsposition einen eigenen Funktionsbaustein aufzurufen. Dieser Funktionsbaustein hinterlegt für die Position im zugehörigen Segment `Billing` die gewünschte Fakturasperre.
- Das Setzen der Fakturasperre soll nur für bestimmte Positionstypen durchgeführt werden, daher bietet sich eine Parametrisierung unserer Erweiterung durch eine eigene Steuerungstabelle an. In dieser kann dann auch hinterlegt werden, welche Fakturasperre jeweils für welchen Positionstyp ermittelt werden soll.

Nun können wir die in Abschnitt 2.6, »Tipps und Tricks«, beschriebene Tracing-Methode für die Suche nach geeigneten Callback-Events nutzen, um eine passende Stelle zu finden, an der wir kurz nach der Erzeugung einer Kundenauftragsposition »einhaken« können: Dazu aktivieren wir den Userparameter `CRM_EVENT_TRACE` und erfassen eine Position in einem Kundenauftrag. Anschließend werten wir das Trace-Protokoll aus, wie in Abbildung 4.42 dargestellt.



Zeile	Rek...	Was ist passiert?	Runtime	Exetime	Objektname	Ereignis im Programm	Attribut	Name des Funktionsbausteins	A	N
229	1	FB aufgerufen		Sofort	BILLING	AFTER_CREATE		CRM_BILLING_BUAG_PSOB_EC	X	X
230		Callback return	76.575	Sofort	BILLING	AFTER_CREATE		CRM_BILLING_BUAG_PSOB_EC		
231		Event publiziert			ORDERADM_I	AFTER_CREATE				
232	1	FB aufgerufen		Sofort	ORDERADM_I	AFTER_CREATE		CRM_ORGMAN_PRODUCT_CHANGED_EC	X	X
233		Event publiziert			ORGMAN	AFTER_CREATE				

Abbildung 4.42 Event-Trace beim Erfassen einer Kundenauftragsposition

Um eines vorwegzunehmen: *Die* richtige Stelle für die Einbettung eigener Callbacks gibt es selten. Oftmals hilft die Trace-Analyse sehr gut dabei, die möglichen Kandidaten einzukreisen, doch meistens ist vor allem auch Ausprobieren notwendig: Wir möchten ein Feld im Bereich der Fakturadaten der CRM-Geschäftsposition ändern. Unser Eingriff soll möglichst frühzeitig nach Erzeugung der Positionsdaten erfolgen, gleichwohl sollten etwaige im Standard enthaltene Schritte zur Initialisierung des Zielsegments zu diesem Zeitpunkt bereits abgeschlossen sein. Die in Abbildung 4.42 markierte Zeile stellt den ersten Callback im CRM-Standard für das Objekt `ORDERADM_I` (Administrationssicht der Geschäftsvorgangsposition) dar, der nach den ersten Callbacks des Objekts `Billing` stattfindet. Diese Stelle erscheint vielversprechend, daher wollen wir uns nun dort »einklinken«:

Zunächst sollten wir die Tauglichkeit der gefundenen Stelle überprüfen: Wir platzieren dazu einen eigenen, leeren Funktionsbaustein in der Eventsteuerung und setzen dort zunächst nur einen Breakpoint. Den Standard-Funktionsbaustein, der in der markierten Zeile in Abbildung 4.42 hinterlegt ist (`CRM_ORGMAN_PRODUCT_CHANGED_EC`) nutzen wir dazu als Kopiervorlage – damit haben wir bereits die passenden Import-/Exportparameter definiert. Danach registrieren wir unseren Funktionsbaustein `Z_SET_BILL_BLOCK` für die Eventsteuerung (Abbildung 4.43). Dies geschieht im View `CRMV_FUNC_ASSIGN`, die Sie mithilfe der Transaktion SM30 pflegen können. Dort hinterlegen Sie das Objekt `CRM_ORDERADM_I`, in dessen Kontext wir diesen Funktionsbaustein ausführen möchten.

Abbildung 4.43 Pflege des Views CRMV_FUNC_ASSIGN

Nun können Sie noch überprüfen, wie unsere »Kopiervorlage« in der Standard-Eventsteuerung verwendet wird. Auch dies hilft häufig dabei, geeignete Customizing-Einstellungen für eigene Erweiterungen des Event Handlers zu finden. Rufen Sie zu diesem Zweck die Transaktion CRMV_EVENT auf. Dort geben Sie im Feld Callback den Namen des gesuchten Funktionsbausteins ein und klicken auf den Button Callback zu Typ/Obj./Ereignis (siehe Abbildung 4.44).

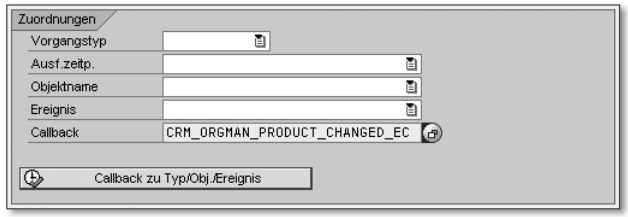


Abbildung 4.44 Suche nach den Standard-Callbacks für einen Funktionsbaustein

Die nun folgenden Treffer können Sie auf das Objekt `ORDERADM_I` eingrenzen. Darüber hinaus haben wir bei der Suche nach einem geeigneten Callback-Event lediglich eine Kundenauftragsposition erfasst – weder mit Bezug zu einem Vorgängerbeleg, noch haben wir die eingegebene Produktnummer anschließend geändert. Damit bleibt nur ein Referenzeintrag übrig, wie in Abbildung 4.45 gezeigt.

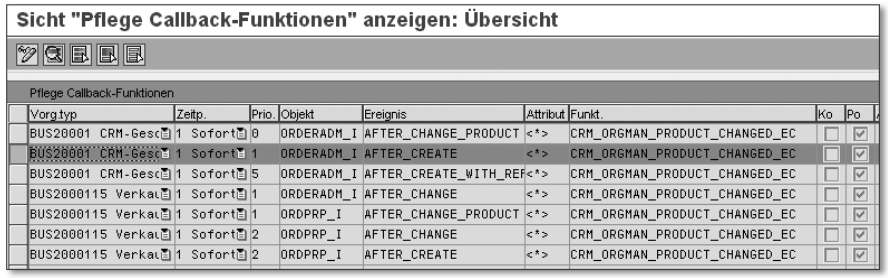


Abbildung 4.45 Standard-Eventsteuerung als Vorlage

Durch einen Doppelklick auf die in Abbildung 4.45 markierte Zeile gelangen Sie in die Detailsicht, anhand derer wir nun unseren eigenen Funktionsbaustein in die Eventsteuerung integrieren. Dazu rufen Sie den folgenden IMG-Pfad auf: Customer Relationship Management • Vorgänge • Grundeinstellungen • Event-Handler-Tabelle bearbeiten. Der nun anzulegende Eintrag sollte im Feld Priorität einen möglichst hohen Wert (= niedrige Priorität) erhalten. Damit kann das Risiko für Konflikte durch eventuell nachträglich in den Standard aufgenommene Callbacks (z. B. durch SAP-Hinweise) verringert werden. Abbildung 4.46 zeigt einen entsprechenden Eintrag.

Nach einem erfolgreichen Check, ob unser Funktionsbaustein auch wirklich »angesprungen« wird, können wir uns anschließend um die Anlage einer eigenen Customizing-Tabelle (Transaktion SE11) kümmern, anhand derer das Setzen einer Fakturasperre parametrisiert werden soll. Einen Vorschlag finden Sie in Abbildung 4.47.

Endgeräte verfügen, ist dies eine relativ häufige Anforderung, da im Außendienst relevante Informationen gern als Langtext abgelegt werden. Zudem lässt sich dieses Konzept auf beliebige andere Felder übertragen, die sich nicht über den Bedingungseditor der Aktionsverarbeitung abprüfen lassen. Als zweites Beispiel zeigen wir Ihnen, wie sich automatisiert Aktivitäten als Folgebeleg zu Serviceverträgen anlegen lassen, um z. B. vor Erreichen des Vertragsendes ein Nachfassen des Innendienstes beim Kunden zu veranlassen.

5.1.1 Vorhandensein eines Langtextes als Aktionsbedingung

Häufig kommt es vor, dass Servicetechniker mit mobilen Endgeräten (sei es mit SAP CRM Mobile Service oder einer anderen CRM-integrierten Softwarelösung) wichtige Informationen für den Innendienst in einem Langtext zum Servicebeleg festhalten. Sofern dieser Langtext nicht in einen Prozess integriert ist (z. B. Workflow oder gleichzeitiges Setzen eines bestimmten Status), kann es nun jedoch passieren, dass diese Information im Innendienst »untergeht« bzw. nicht wahrgenommen wird. Wünschenswert ist daher, dass der zuständige Innendienstmitarbeiter automatisch über das Vorhandensein eines solchen Textes informiert wird. Dies kann wiederum mithilfe der Aktionsverarbeitung realisiert werden: Sobald ein bestimmter Text im Beleg gefüllt wird, soll dieser Text (zusammen mit anderen Belegdaten) in einer E-Mail an den zuständigen Innendienstmitarbeiter gesendet werden.

Der Knackpunkt dieser Erweiterung ist, dass das Vorhandensein eines bestimmten Langtextes als Startbedingung für die Aktion dienen soll. Alles Weitere lässt sich durch den CRM-Standard abhandeln (Aktion mit Smart Form, E-Mail an Belegpartner, z. B. den *zuständigen Mitarbeiter*). In Abschnitt 2.4, »Aktionsverarbeitung«, haben Sie bereits die Grundlagen für diese Erweiterung kennengelernt. Zunächst müssen Sie eine passende Aktion einrichten (siehe Abbildung 5.1) und dann eine Aktionsbedingung (Startbedingung) erstellen. Die Aktion soll partnerabhängig sein, da die E-Mail an den zuständigen Mitarbeiter im Beleg (Partnerfunktion *00000014*) geschickt und automatisch eingeplant werden soll.

Für die Startbedingung legen Sie einen neuen Bedingungsparameter `TEXT_FOUND` vom Typ `BOOLEAN` an, wie in Abschnitt 2.4 beschrieben, und erstellen eine Startbedingung `TEXT_FOUND = X` (siehe Abbildung 5.2). Nun müssen Sie, wie ebenfalls bereits in Abschnitt 2.4 gezeigt, noch das BAdI `CONTAINER_PPF` implementieren und Ihren neu erstellten Parameter `TEXT_FOUND` dann setzen, wenn der gewünschte Langtext im Beleg vorhanden ist. Das Beispiel-Coding für die (fiktive) Text-ID Z001 finden Sie in Listing 5.1.

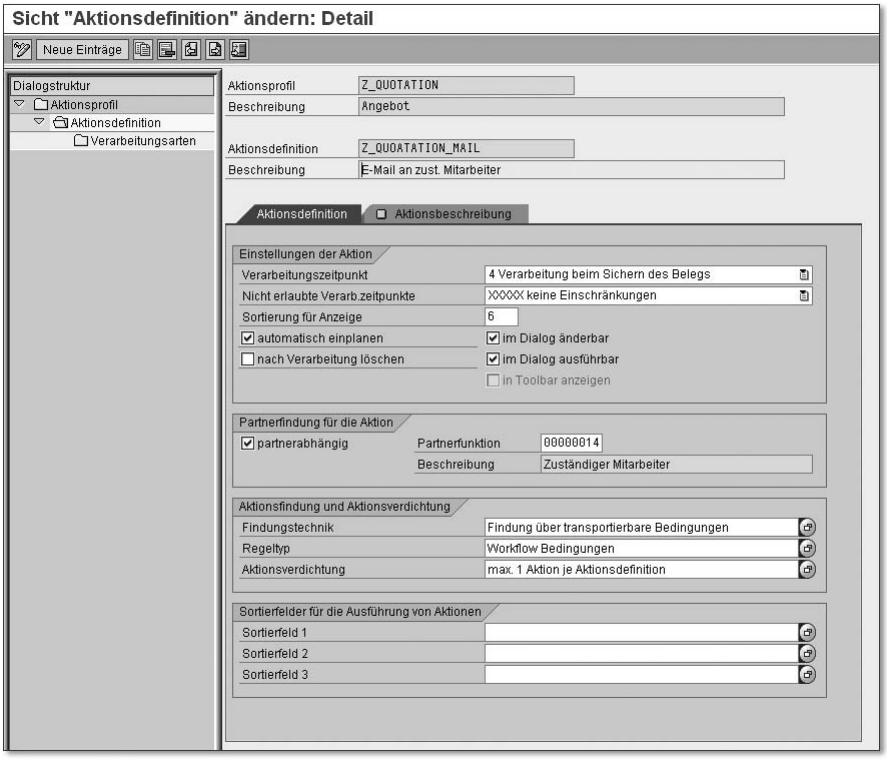


Abbildung 5.1 Aktionsdetails der E-Mail-Aktion

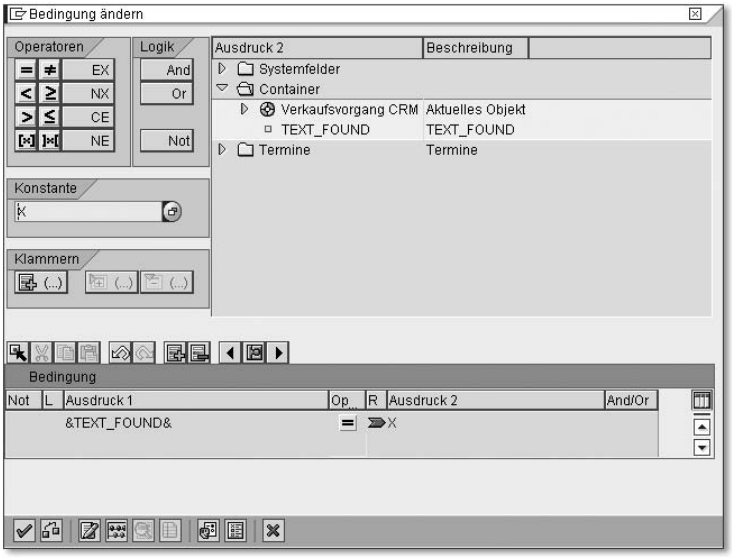


Abbildung 5.2 Startbedingung »Text vorhanden«

```
method IF_EX_CONTAINER_PPF~MODIFY_CONTAINER.
  INCLUDE: crm_direct.
  DATA: ls_object          TYPE sibflporb,
        lt_value           TYPE swconttab,
        ls_value           TYPE swcont,
        lv_return          TYPE sy-subrc,
        lv_guid            TYPE crmt_object_guid,
        lt_guid            TYPE crmt_object_guid_tab,
        lt_text            TYPE CRMT_TEXT_WRKT,
        ls_text            TYPE CRMT_TEXT_WRK,
        lt_req_objects     TYPE crmt_object_name_tab.
CHECK: ci_container IS BOUND,
ci_parameter IS BOUND.
* ------- Get the GUID ----------
  CALL METHOD ci_container->get_value
    EXPORTING
      element_name = 'BUSINESSOBJECT'
    IMPORTING
      data         = ls_object.
  lv_guid = ls_object-instid.
  CALL METHOD ci_parameter->get_values
    RECEIVING
      values       = lt_value.
  loop at lt_value into ls_value where ELEMENT = 'TEXT_FOUND'.
   clear ls_value-value.
  endloop.
  If sy-subrc ne 0.
    exit.
  endif.
  INSERT gc_object_name-texts    INTO TABLE lt_req_objects.
  refresh: lt_guid.
  insert lv_guid into table lt_guid.
  refresh: lt_text.
  If not lt_guid is initial.
    CALL FUNCTION 'CRM_ORDER_READ'
      EXPORTING
        IT_header_GUID       = lt_guid
        IT_REQUESTED_OBJECTS = lt_req_objects
      IMPORTING
        ET_TEXT              = lt_text
      EXCEPTIONS
        DOCUMENT_NOT_FOUND   = 1
        ERROR_OCCURRED       = 2
        DOCUMENT_LOCKED      = 3
        NO_CHANGE_AUTHORITY  = 4
```

```
        NO_DISPLAY_AUTHORITY = 5
        NO_CHANGE_ALLOWED    = 6
        OTHERS               = 7.
  endif.
  clear: ls_text.
  If not lt_text is initial.
    loop at lt_text into ls_text where STXH-TDID = 'Z001'.
    endloop.
    If sy-subrc = 0.
* --------- Set parameter TEXT_FOUND -----------
      CALL METHOD ci_parameter->set_value
        EXPORTING
          element_name = 'TEXT_FOUND'
          data         = 'X'
        RECEIVING
          retcode      = lv_return.
    else. "Parameter löschen
      CALL METHOD ci_parameter->set_value
        EXPORTING
          element_name = 'TEXT_FOUND'
          data         = ' '
        RECEIVING
          retcode      = lv_return.
    endif.
  endif.
endmethod.
```

Listing 5.1 BAdI CONTAINER_PPF, Methode MODIFY_CONTAINER

Sie haben nun erreicht, dass Ihre Aktion immer dann beim Sichern des Belegs ausgeführt wird, wenn der Text Z001 gefüllt ist. In unserem Beispiel wird die Aktion nur einmalig ausgeführt – Änderungen des Textes führen also nicht zu einer erneuten E-Mail an den zuständigen Mitarbeiter.

Nun müssen Sie noch ein Formular (z. B. Smart Forms) erstellen, das den entsprechenden Langtext (und gegebenenfalls weitere Daten aus dem Beleg) ausgibt. Für die Ausgabe des Langtextes können Sie im Formular einen Knoten vom Typ INCLUDE-TEXT verwenden, wie in Abbildung 5.3 gezeigt. Auf die weitere Smart-Forms-Bearbeitung gehen wir an dieser Stelle allerdings nicht ein (siehe dazu u. a. Werner Hertleif und Christoph Wachter: *SAP Smart Forms*, SAP PRESS 2003).

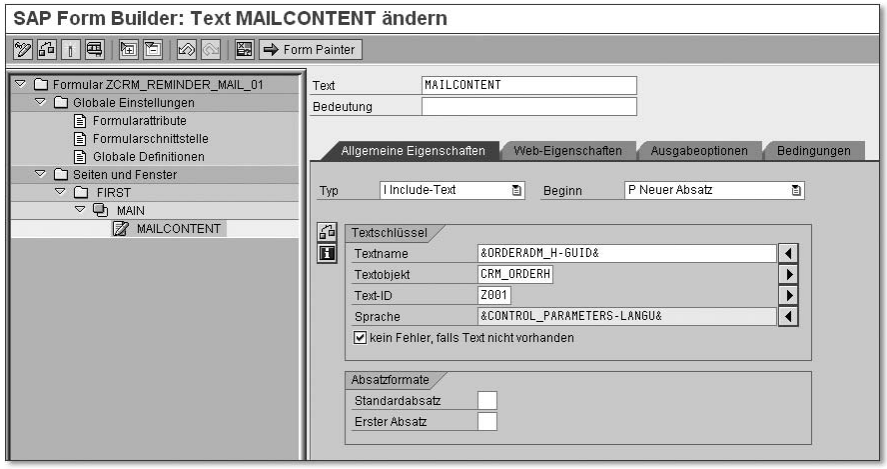


Abbildung 5.3 Include-Text in SAP Smart Forms einbinden

5.1.2 Automatisiertes Anlegen von Folgebelegen

Dieser Abschnitt zeigt Ihnen, wie Sie mithilfe der Aktionsverarbeitung automatisiert Folgebelege (hier Aktivitäten) zu bestehenden Belegen (hier Serviceverträgen) anlegen können. Dies ist hilfreich, um beispielsweise vor Erreichen des Vertragsendes ein Nachfassen des Innendienstes beim Kunden sicherzustellen.

Technisch baut die folgende Erweiterung auf dem in Abschnitt 2.4, »Aktionsverarbeitung«, gezeigten Methodenaufruf in Aktionen auf. Im Sales-Bereich (Abschnitt 4.1, »Aktionsverarbeitung«) haben wir ebenfalls bereits ein Beispiel für den Aufruf von Standard-Verarbeitungsmethoden gezeigt. Auch hier ist es möglich, das gewünschte Verhalten mithilfe einer im Standard bereits vorhandenen Verarbeitungsmethode zu erreichen (`COPY_DOCUMENT`). Falls Ihnen die vorhandene Funktionalität nicht ausreicht, kopieren Sie die Methode `COPY_DOCUMENT`, und passen Sie sie für Ihre Zwecke an.

In unserem Beispiel wollen wir die Bezeichnung der Zielaktivitäten beeinflussen. In der Standard-Methode `COPY_DOCUMENT` wird der Aktionstext als Bezeichnung der erzeugten Belege verwendet (Feld `ORDERADM_H-DESCRIPTION`). Wenn Ihre Aktion also Erinnerung Vertragsende heißt, erhalten alle erzeugten Aktivitäten diesen Text als Bezeichnung. Als einfaches Beispiel wollen wir nun die Kundennummer des Auftraggebers in diesen Text mit aufnehmen. Wir kopieren also die BAdI-Implementierung `COPY_DOCUMENT` nach `Z_COPY_DOCUMENT` und passen sie an, wie in Listing 5.2 gezeigt.

Index

C

D

E

F

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z