Kapitel 5
Anwendung der Consumer-API

Willkommen im BOPF-Programmierparadies! Hier finden Sie Dutzende von Programmierbeispielen, die Ihnen zeigen, wie Sie mit der Consumer-API auf bestehende Geschäftsobjekte zugreifen.

In diesem Kapitel zeigen wir Ihnen, wie Sie die Methoden der in Kapitel 4, »Architektur der Consumer-API«, vorgestellten Consumer-API auf bereits bestehende Geschäftsobjekte anwenden können. Dazu zählen in erster Linie die *CRUDQ*-Grundoperationen zum Bearbeiten von Knoteninstanzen eines Geschäftsobjekts:

- Anlegen (Create)
- Lesen (Read)
- Aktualisieren (Update)
- Löschen (Delete)
- Abfragen (Query)

Darüber hinaus zeigen wir Ihnen aber auch alle weiteren wichtigen Funktionen, die Sie bei Ihrer alltäglichen Arbeit mit dem BOPF und seinen Geschäftsobjekten benötigen werden. Zu diesen Funktionen zählen:

- das Auflösen von Assoziationen zwischen zwei Knoten
- das Sperren und Entsperren von Geschäftsobjektinstanzen ohne Modifikation (Änderung)
- das Ermitteln eines Schlüssels zu einem alternativen Schlüssel
- das Auslesen von Standardwerten für die Anlage von Knoten und für das Ausführen von Aktionen und Abfragen
- das Ermitteln des Oberknotens zu einem gegebenen Knoten
- das Wissen über das Ausführen von Aktionen, Validierungen und Ermittlungen eines Geschäftsobjekts

Wichtig ist es, zu wissen, dass in den Listings dieses Kapitels immer wieder die Variable *LO_SERV_MGR* vorkommt, die eine Referenz auf den Service-Manager hält. Dieser wird aus Platzgründen nicht immer wieder erneut beschafft. Die Listings bauen daher immer auf dem folgenden Listing 5.1 auf:

```
DATA: lo_serv_mgr  TYPE REF TO /bobf/if_tra_service_manager.
lo_serv_mgr = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(
  iv_bo_key = /bobf/if_demo_customer_c=>sc_bo_key
).
```

Listing 5.1 Beschaffung des Service-Managers

Das vorliegende Kapitel können Sie auch durcharbeiten, wenn Sie noch kein eigenes Geschäftsobjekt angelegt haben, da Sie in jedem Abschnitt ein auf den Demo-Geschäftsobjekten des SAP-Systems beschriebenes Beispiel vorfinden werden.

Wir empfehlen Ihnen, die hier vorgestellten Programmierbeispiele direkt im ABAP-Editor auszuprobieren. Den Erfolg können Sie anhand der in Kapitel 3, »Testen von Geschäftsobjekten«, beschriebenen Test-UI, der Transaktion BOBT, durch Vergleich der Ergebnisse überprüfen.

[!]

Consumer-API nur für externe Aufrufe

Alle in diesem Kapitel beschriebenen Methoden sollten Sie nicht innerhalb eines Geschäftsobjekts, d. h. in den Implementierungen von Aktionen, Validierungen, Ermittlungen usw., verwenden. Die interne Verwendung wird in Kapitel 6, »Die Geschäftsobjekt-API«, beschrieben. Dort geschieht die Implementierung durch die in den Methoden bereitgestellten Importparameter.

5.1 Lesen von Knoteninstanzen

Das Lesen von Instanzen eines Geschäftsobjekts gehört zu den absoluten Grundaufgaben des Frameworks. In diesem Zusammenhang möchten wir noch einmal darauf hinweisen, dass immer nur Instanzen eines einzelnen Knotens ausgelesen werden können und nicht die Instanzen aller Knoten eines Geschäftsobjekts. In der Regel werden Sie daher erst den Hauptknoten (*ROOT*) auslesen und anschließend, falls weitere Knoten benötigt werden, von diesem aus über Assoziationen zu Ihrem gewünschten Zielknoten navigieren. Beim Lesen von Knoteninstanzen eines Geschäftsobjekts müssen zwei Szenarien unterschieden werden:

1. **Abfrage von Instanzen anhand von Kriterien**
   Sie haben keinen Schlüssel, aber eventuell einen oder mehrere alternative Schlüssel (beispielsweise die ursprünglichen Primärschlüssel) oder Filterkriterien, zu denen Sie die dazugehörigen Instanzen bzw. die Schlüssel auslesen möchten. Dann können Sie die Methode *QUERY* des Service-Managers verwenden.

2. **Abfrage von Instanzen über einen Schlüssel**
   Sie haben bereits einen oder mehrere Schlüssel und wollen dazu die dazugehörigen Knoteninstanzen eines Geschäftsobjekts auslesen. Dann können Sie die Methode *RETRIEVE* oder *QUERY* des Service-Managers verwenden.

5.1.1 Die Methode QUERY

Zur Abfrage von Knoteninstanzen eines Geschäftsobjekts anhand von bestimmten Kriterien können Sie die Methode *QUERY* des Service-Managers verwenden.

[«]

Nur Rückgabe von persistenten Daten

Abfragen geben nur die Schlüssel von Instanzen zurück, die bereits auf der Datenbank persistiert wurden (und nicht solche, die gerade im Speicher in Bearbeitung liegen). Abfragen sollten daher nicht innerhalb der transaktionalen Verarbeitung (z. B. in einer Aktion, Ermittlung oder Validierung) verwendet werden. Als Alternative stehen Ihnen hier Assoziationen zur Verfügung.

Konkret können Sie mit dieser Methode dieselben Funktionen wie mit einer *SELECT*-Anweisung (von Joins und Aggregationen abgesehen) abbilden. Zu diesen Funktionen zählen:

- vordefinierte Abfragen aus dem Geschäftsobjektausführen
- nach Schlüsseln und Werten filtern
- bestimmen, ob nur die ermittelten Schlüssel oder auch die Daten der Knoteninstanzen zurückgegeben werden sollen
- sich die Anzahl der gefunden Einträge zurückgeben lassen
- die maximale Ergebnismenge einschränken
- die Ergebnismenge sortieren
- Paging verwenden, d. h. sich nur eine Teilmenge zurückgeben lassen

Nachfolgend erläutern wir alle diese Funktionen getrennt. Das bedeutet aber nicht, dass Sie diese nicht auch kombiniert verwenden können.

Grundaufbau

Zur Demonstration der Funktionsweise der Methode *QUERY* wird in diesem Abschnitt die grundlegende Abfrage *SELECT_ALL* auf den Hauptknoten (hier *ROOT*) verwendet, die in jedem Geschäftsobjekt als Standard definiert werden kann. Diese gibt, wie der Name es aussagt, alle Instanzen eines Knotens zu dem jeweiligen Geschäftsobjekt zurück.

Beim Aufruf der Methode *QUERY* müssen Sie unter dem Parameter *IV_QUERY_KEY* über das Konstanten-Interface des Geschäftsobjekts unter Konstante *SC_QUERY_ROOT* den Schlüssel dieser Abfrage entsprechend auswählen. Für das Geschäftsobjekt */BOBF/DEMO_CUSTOMER* wäre der Pfad zu dem Schlüssel der Abfrage der folgende:

/bobf/if_demo_customer_c=>sc_query-root-select_all

Listing 5.2 zeigt einen beispielhaften Aufruf der Methode *QUERY* für den Knoten *ROOT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER*.

```
DATA: lt_data TYPE /bobf/t_demo_customer_hdr_k,
      lt_keys TYPE /bobf/t_frw_key.
"Beschaffen des Service-Managers wie im Listing 5.1
lo_serv_mgr->query(
  EXPORTING
    iv_query_key = /bobf/if_demo_customer_c=>
                   sc_query-root-select_all
    iv_fill_data = abap_true
  IMPORTING
    et_data      = lt_data
    et_key       = lt_keys
).
```

Listing 5.2 Grundaufbau einer Abfrage

Sind Sie nicht nur an den Schlüsseln (Rückgabetabelle *ET_KEY*), sondern auch an den Daten der Knoteninstanzen interessiert, können Sie diese über den Parameter *IV_FILL_DATA = 'X'* abfragen. Dadurch wird auch die Rückgabetabelle *ET_DATA* mit den Daten der Knoteninstanzen gefüllt. Wichtig dabei ist, dass Sie für diesen Parameter eine Tabelle mit der richtigen, zu dem Knoten passenden Struktur bereitstellen.

Eine solche Abfrage über die Methode *QUERY* können Sie für alle vordefinierten Abfragen des Geschäftsobjekts durchführen. Wählen Sie dazu über das Konstanten-Interface die entsprechende Abfrage aus. Nachfolgend ist dies über die zweite Standardabfrage *SELECT_BY_ATTRIBUTES* (bzw. *SELECT_BY_ELEMENTS*), die ebenfalls für jeden Knoten zusätzlich angelegt werden kann, dargestellt.

Einschränken über Attribute

Zum Einschränken der Abfrageergebnisse nach Attributwerten bietet der Parameter *IT_SELECTION_PARAMETERS* den Grundaufbau einer Range-Tabelle mit den klassischen Feldern *SIGN*, *OPTION*, *LOW* und *HIGH* an.

Zusätzlich zu diesen Grundfeldern müssen Sie in der Struktur noch den Namen des einzuschränkenden Attributs im Feld *ATTRIBUTE_NAME* angeben. Durch dieses zusätzliche Feld ist es möglich, dass Sie auf mehrere Attribute gleichzeitig filtern können oder für ein Attribut mehrere Einschränkungen treffen können, z. B. pro Zeile der Selektions-Parametertabelle eine Einschränkung. Listing 5.3 zeigt Ihnen eine beispielhafte Implementierung einer Abfrage mit *SELECT_BY_ATTRIBUTES* für den Knoten *ROOT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER*. Im Beispiel werden alle Kunden gesucht, die »EUR« als Kundenwährung haben.


```
DATA: lt_sel_param TYPE /bobf/t_frw_query_selparam,
      ls_sel_param TYPE /bobf/s_frw_query_selparam.

ls_sel_param-attribute_name = /bobf/if_demo_customer_c=>sc_
   query_attribute-root-select_by_attributes-cust_curr.
"Oder auch direkte Angabe des  Attributs CUST_CURR möglich
ls_sel_param-low    = 'EUR'.
ls_sel_param-option = 'EQ'.
ls_sel_param-sign   = 'I'.
APPEND ls_sel_param TO lt_sel_param.
lo_serv_mgr->query(
  EXPORTING
    iv_query_key  = /bobf/if_demo_customer_c=>sc_query-root-
                    select_by_attributes
    iv_fill_data  = abap_true
    it_selection_parameters = lt_sel_param
  IMPORTING
    et_data       = lt_data
    et_key        = lt_keys
).
```

Listing 5.3 Abfrage nach Attributen

Einschränken über Schlüssel

Wenn Sie bereits die Schlüssel einer Knoteninstanz eines Geschäftsobjekts haben und sich dazu die Daten der Knoteninstanz auslesen lassen wollen, müssen Sie diesen Schlüssel nur im Feld *KEY* der Tabelle des Parameters *IT_FILTER_KEY* angeben und zusätzlich den Parameter *IV_FILL_DATA* auf *abap_true* setzen. Listing 5.4 zeigt eine solche Abfrage auf Knoteninstanzen über einen Schlüssel beispielhaft.

```
DATA: lt_filter_key TYPE /bobf/t_frw_key,
      ls_filter_key TYPE /bobf/s_frw_key.

ls_filter_key-key = '000C294A1BC91ED78FA9BE1AD6C520E0'.
APPEND ls_filter_key TO lt_filter_key.
```

```
lo_serv_mgr->query(
  EXPORTING
    iv_query_key  = /bobf/if_demo_customer_c=>sc_query-root-
                    select_all
    iv_fill_data  = abap_true
    it_filter_key = lt_filter_key
  IMPORTING
    et_data       = lt_data
    et_key        = lt_keys
).
```

Listing 5.4 Abfrage nach Schlüsseln

5.1.2 Die Methode RETRIEVE

Anstelle bei der Methode *QUERY* im Parameter *IT_FILTER_KEY* die Schlüssel mitzugeben und über den Parameter *IV_FILL_DATA* die Daten anzufragen, können Sie dies auch über die Methode *RETRIEVE* des Service-Managers erreichen.

Vorteil dieser Methode ist, dass Sie über den Parameter *IV_INVALIDATE_CACHE* die Möglichkeit haben, den Puffer zurückzusetzen und, falls Sie dies benötigen, Sperren auf die Knoteninstanzen zu setzen. Letzteres finden Sie in Abschnitt 5.4, »Sperren/Entsperren«, beschrieben.

Listing 5.5 zeigt Ihnen einen solchen Aufruf der Methode *RETRIEVE*. Achten Sie darauf, wirklich nur Schlüssel und keine alternativen Schlüssel mitzugeben. Im Beispiel ist zu demonstrativen Zwecken ein richtiger und ein falscher Schlüssel mitgegeben worden.

```
DATA: lt_keys        TYPE /bobf/t_frw_key,
      ls_key         TYPE /bobf/s_frw_key,
      lt_failed_keys TYPE /bobf/t_frw_key.

"beispielhafter Schlüssel
ls_key-key = '0050568F72901ED79EA326CDC37C60FF'.
APPEND ls_key TO lt_keys.

ls_key-key = '123456'. "falsch
APPEND ls_key TO lt_keys.

lo_serv_mgr->retrieve(
  EXPORTING
    iv_node_key         = /bobf/if_demo_customer_c=>sc_node-
                          root
    it_key              = lt_keys
```



```
    iv_fill_data        = abap_true
    iv_invalidate_cache = abap_true
  IMPORTING
    et_data             = lt_data
    et_failed_key       = lt_failed_keys
).
```

Listing 5.5 Auslesen von Daten mit RETRIEVE

Für die Schlüssel, für die die Abfrage erfolgreich war, können Sie im Rückgabeparameter *ET_DATA* die Daten der Knoteninstanzen finden. Alle fehlerhaften Schlüssel würden dagegen im Rückgabeparameter *ET_FAILED_KEY* zurückgegeben werden.

5.1.3 Auslesen einer Verknüpfung mit RETRIEVE_BY_ASSOCIATION

In Abschnitt 2.5, »Assoziationen«, haben wir Ihnen die Möglichkeiten der Verknüpfung von Knoten über Assoziationen vorgestellt. Die Methode *RETRIEVE_BY_ASSOCIATION* ist letztlich die Stelle, an der Sie mit diesen definierten Verknüpfungen arbeiten können und von einer Instanz eines Knotens A zu einem oder mehreren Instanzen eines Knotens B navigieren können.

Zum Aufruf der Methode müssen Sie im Parameter *IV_ASSOCIATION* den Namen der Assoziation angeben. Diesen können Sie im Konstanten-Interface des Geschäftsobjekts unter der Konstante *SC_ASSOCIATION*, gefolgt von dem gewünschten Knoten, finden. Im Parameter *IV_NODE_KEY* müssen Sie zusätzlich die ID des gewünschten Knotens angeben, von der aus Sie navigieren wollen, während Sie im Parameter *IT_KEY* die Schlüssel der zu diesem Knoten gehörenden Knoteninstanzen hinterlegen. Über den Parameter *IV_FILL_DATA* können Sie direkt die Daten der ermittelten Knoteninstanzen anfragen, die im Rückgabeparameter *ET_DATA* bereitgestellt werden.

Listing 5.6 zeigt beispielhaft die Navigation vom Knoten *ROOT* zu dem Unterknoten *ROOT_TEXT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER* über die gleichnamige Assoziation *ROOT_TEXT*.

```
DATA: lt_data TYPE /bobf/t_demo_short_text_k,
      lt_key  TYPE /bobf/t_frw_key,
      ls_key  TYPE /bobf/s_frw_key.

"beispielhafter Schlüssel
ls_key-key = '0050568F72901ED79EA63B400C91A0FF'.
APPEND ls_key TO lt_key.

lo_serv_mgr->retrieve_by_association(
  EXPORTING
```

```
    iv_node_key    = /bobf/if_demo_customer_c=>sc_node-root
    it_key         = lt_key
    iv_association = /bobf/if_demo_customer_c=>sc_
                     association-root-root_text
    iv_fill_data   = abap_true
  IMPORTING
    et_data        = lt_data ).
```

Listing 5.6 Auslesen von Daten über eine Assoziation

Bei Erfolg werden Ihnen in der Rückgabetabelle *ET_DATA* die ausgelesenen Knoteninstanzen zurückgegeben.

5.2 Konvertieren von alternativen Schlüsseln

Häufig werden Sie bereits einen alternativen Schlüssel eines Geschäftsobjektsknotens zur Hand haben (beispielsweise die Kundennummer 123456) und möchten dann zu diesem alternativem Schlüsselden technischen Schlüssel des dazugehörigen Knotens bestimmen, damit Sie mit diesem technischen Schlüssel eine Sperrung, eine Modifikation oder eine Assoziation über die Methoden des Frameworks durchführen können. In einem solchen Fall haben Sie zwei Möglichkeiten:

- Sie führen mit dem alternativen Schlüssel eine Abfrage über *SELECT_BY_ATTRIBUTES* aus (siehe Abschnitt 5.1.1).
- Sie verwenden die Methode *CONVERT_ALTERN_KEY* des Service-Managers und übergeben dort den alternativen Schlüssel.

Die Verwendung der Methode *CONVERT_ALTERN_KEY* ist denkbar einfach: Sie müssen in der internen Tabelle des Parameters *IT_KEY* die alternativen Schlüssel, im Parameter *IV_NODE_KEY* den entsprechenden Knoten und im Parameter *IV_ALTKEY_KEY* den Namen des alternativen Schlüssels mitgeben.

Auch wenn sich der Parameter *IV_ALTKEY_KEY* über Literale (durch die direkte Angabe von *'CUSTOMER_ID'*) befüllen lässt, empfiehlt sich die Benutzung des Konstanten-Interface des Geschäftsobjekts. Die alternativen Schlüssel finden Sie im Konstanten-Interface unter der Konstante *SC_ALTERNATIVE_KEY*, gefolgt vom Namen des Knotens.

In Listing 5.7 sehen Sie eine beispielhafte Implementierung der Methode, mit der für den alternativen Schlüssel *CUSTOMER_ID* der Schlüssel der dazugehörigen Knoteninstanz bestimmt wird.

```
DATA: lt_alt_keys TYPE TABLE OF /bobf/demo_customer_id,
      ls_alt_keys TYPE /bobf/demo_customer_id,
      lt_keys     TYPE /bobf/t_frw_key.

ls_alt_keys = '123456'.
APPEND ls_alt_keys TO lt_alt_keys.

lo_mgr->convert_altern_key(
  EXPORTING
    iv_node_key   = /bobf/if_demo_customer_c=>sc_node-root
    iv_altkey_key = /bobf/if_demo_customer_c=>sc_
                    alternative_key-root-customer_id
    it_key        = lt_alt_keys
  IMPORTING
    et_key        = lt_keys
).
```

Listing 5.7 Konvertierung von alternativen Schlüsseln

Bei erfolgreicher Ausführung erhalten Sie die zugehörigen Schlüssel in der Rückgabetabelle *ET_KEY*.

5.3 Modifizieren von Objekten

Das Modifizieren von Objekten geschieht im BOPF mit der Methode *MODIFY* des Service-Managers. Mit dieser Methode können Sie Geschäftsobjekte anlegen, ändern und löschen.

Zusätzlich wird zum Abschluss für all diese Modifikationen die Methode *SAVE* des Transaktions-Managers benötigt. Den Transaktions-Manager brauchen Sie in diesem Fall, da Datenbankmodifikationen nicht direkt ausgeführt, sondern wie bei klassischen transaktionalen Funktionsbausteinen (BAPI-Funktionsbausteinen) in einer logischen Datenbankeinheit (*LUW*) gesammelt werden. Die Methode *SAVE* stößt zum Abschließen dieser Verarbeitung die Anweisung *COMMIT WORK* an und verarbeitet die in der *LUW* gesammelten Datenbankoperationen.

Datenbanksperren

Sperren werden durch den Aufruf der Methode *MODIFY* für die übergebenen Schlüssel automatisch gesetzt. Die Sperren gelten für das gesamte Geschäftsobjekt und werden durch die Methode *SAVE* wieder freigegeben. Falls das Speichern fehlschlägt, kann die Freigabe mit der Methode *CLEANUP* erfolgen. Darüber können Sie auch manuelle Sperren ohne eine Änderung durch die Methode *MODIFY* setzen. Diese Vorgehensweise wird in Abschnitt 5.4 genauer beschrieben.

Die entscheidende Tabelle beim Modifizieren mit der Methode *MODIFY* ist die Tabelle des Importparameters *IT_MODIFCATION*. Eine Zeile dieser Tabelle entspricht genau einer Modifikation. Für jede Zeile kann über das Feld *CHANGE_MODE* bestimmt werden, welche Modifikation ausgeführt wird (also ob angelegt, geändert oder gelöscht wird). Die Methode *MODIFY* ermöglicht daher die Ausführung mehrerer Modifikationsoperationen gleichzeitig. Listing 5.8 zeigt die grundlegende Aufrufreihenfolge der Methode *MODIFY*.

```
"Beschaffen des Service-Managers
DATA: lo_serv_mgr TYPE REF TO /bobf/if_tra_service_manager.
lo_mgr = /bobf/cl_tra_serv_mgr_factory=>get_service_manager(
  iv_bo_key = /bobf/if_demo_customer_c=>sc_bo_key
).
"Beschaffen des Transaktions-Managers
DATA: lo_trans_mgr TYPE REF TO /bobf/if_tra_transaction_mgr.
lo_trans_mgr = /bobf/cl_tra_trans_mgr_factory=>
                 get_transaction_manager( ).
"Deklaration der benötigten Variablen
DATA: lt_modi          TYPE /bobf/t_frw_modification,
      ls_modi          TYPE /bobf/s_frw_modification,
      lo_message       TYPE REF TO /bobf/if_frw_message,
      lt_messages      TYPE /bobf/t_frw_message_k,
      lv_rejected      TYPE flag,
      lt_rejected_keys TYPE /bobf/t_frw_key2.
FIELD-SYMBOLS: <s_message> TYPE /bobf/s_frw_message_k.

"Hier unterschiedliches Befüllen der Tabelle LT_MODI je nachdem,
"ob Sie die das Geschäftsobjekt Anlegen, Ändern oder
"Aktualisieren wollen
ls_modi-...
APPEND ls_modi TO lt_modi.
"Aufruf der Methode MODIFY und Übergabe der Modifikations-Tabelle
lo_serv_mgr->modify(
  EXPORTING
    it_modification = lt_modi
  IMPORTING
    eo_message      = lo_message
).
"Überprüfung ob die Verarbeitung erfolgreich war
IF lo_message IS BOUND.
  IF lo_message->check( ) EQ abap_true.
    "Ausgabe der Nachrichten im Fehlerfall
    lo_message->get_messages(
```



```
      IMPORTING et_message = lt_messages
    ).
    LOOP AT lt_messages ASSIGNING <s_message>.
      cl_demo_output=>write_data(
        <s_message>-message->get_text( )
      ).
    ENDLOOP.
    cl_demo_output=>display( ).
  ELSE. "bei erfolgreicher Ausführung speichern
    lo_trans_mgr->save(
      IMPORTING
        ev_rejected = lv_rejected
    ).
    "Falls Speichern erfolgreich, Ausgabe sonst Rollback
    IF lv_rejected IS INITIAL.
      cl_demo_output=>display_text( 'Erfolg!' ).
    ELSE.
      lo_trans_mgr->cleanup( ).
    ENDIF.
  ENDIF.
ENDIF.
```

Listing 5.8 Grundaufbau für die Methode MODIFY

Das Feld *CHANGE_MODE* können Sie mit folgenden Konstanten des Interface */BOBF/IF_FRW_C* belegen:

- */BOBF/IF_FRW_C=>SC_MODIFY_CREATE* für Anlegen
- */BOBF/IF_FRW_C=>SC_MODIFY_UPDATE* für Aktualisieren
- */BOBF/IF_FRW_C=>SC_MODIFY_DELETE* für Löschen

Je nachdem, ob Sie anlegen, aktualisieren oder löschen wollen, müssen Sie die Zeilenfelder der Tabelle *IT_MODIFICATION* unterschiedlich belegen. Einen Überblick über die einzelnen Felder der Modifikationstabelle finden Sie in Tabelle 5.1.

Feldname	Beschreibung
NODE	Name des zu modifizierenden Knotens
CHANGE_MODE	Art der Änderung
NODE_CAT	Beim Anlegen: Knotenkategorie; wenn keine mitgegeben wird, wird die Standardkategorie verwendet (obsolet, immer leer lassen).
KEY	Beim Anlegen: neu erzeugter Schlüssel des Knotens (mit *GET_NEW_KEY*)

Tabelle 5.1 Die Felder der Modifikationsstruktur

Feldname	Beschreibung
DATA	Beim Anlegen und Aktualisieren: Daten des Knotens
CHANGED_FIELDS	Beim Ändern: die geänderten Felder
ASSOCIATION	ID der Verknüpfung; zum Anlegen von Unterknoten
SOURCE_NODE	eventuell referenzierter Oberknoten; zum Anlegen von Unterknoten
SOURCE_KEY	eventuell referenziertes Geschäftsobjekt; zum Anlegen von Unterknoten
ROOT_KEY	Optional beim Anlegen und Aktualisieren: Zu welchen *ROOT*-Knoten soll die Instanz gehören?

Tabelle 5.1 Die Felder der Modifikationsstruktur (Forts.)

In den folgenden Abschnitten beschreiben wir diese Aspekte:

- Anlegen von Hauptknoten
- Anlegen von Unterknoten
- Aktualisieren von Knoteninstanzen
- Löschen von Knoteninstanzen

5.3.1 Anlegen von Hauptknoten

Beim Anlegen von Hauptknoten müssen Sie die Felder *KEY*, *DATA*, *NODE* und natürlich *CHANGE_MODE* befüllen. Wichtig ist, dass Sie in der Struktur des Parameters *DATA* und im Parameter *KEY* einen neu generierten Schlüssel mitgeben, den Sie von der Methode *GET_NEW_KEY* des Service-Managers erhalten.

Zusätzlich müssen Sie dafür sorgen, dass Sie im Parameter *DATA* die richtige Struktur des gewünschten Knotens mitgeben. Das Feld ist hierbei vom Typ *TYPE REF TO data*. Daher müssen Sie die Struktur vorher ebenfalls über *TYPE REF TO strukturname* deklarieren und über die Anweisung *CREATE DATA* erzeugen. Den Strukturnamen finden Sie in den Einstellungen des Knotens. Anschließend können Sie die erzeugte Struktur über den Komponentenselektor -> befüllen. Listing 5.9 zeigt dies beispielhaft für das Geschäftsobjekt */BOBF/DEMO_CUSTOMER* für den Knoten *ROOT*.

```
"Deklaration der Variablen und Beschaffung des Service-Managers
"und Transaktions-Managers wie in Listing 5.8
DATA: lr_cust_root TYPE REF TO /bobf/s_demo_customer_hdr_k.
"Erzeugen der neuen Knoten-Struktur des Kunden-Kopfes
CREATE DATA lr_cust_root.
```



```
"Erzeugung eines neuen technischen Schlüssels
lr_cust_root->key = lo_serv_mgr->get_new_key( ).
"oder über /bobf/cl_frw_factory=>get_new_key( ).

lr_cust_root->customer_id    = '123459'.
lr_cust_root->sales_org      = '200'.
lr_cust_root->cust_curr      = 'US'.
lr_cust_root->address_contry = 'US'.

ls_modi-data = lr_cust_root. "Befüllte Struktur
ls_modi-node = /bobf/if_demo_customer_c=>sc_node-root.
ls_modi-change_mode = /bobf/if_frw_c=>sc_modify_create.
ls_modi-key         = lr_cust_root->key.
APPEND ls_modi TO lt_modi.

lo_serv_mgr->modify(
  EXPORTING
    it_modification = lt_modi
  IMPORTING
    eo_message      = lo_message
).
```

Listing 5.9 Anlegen eines Hauptknotens

Ob das Anlegen erfolgreich war, können Sie wie in Listing 5.8 über den Rückgabeparameter *LO_MESSAGE* ermitteln. Zusätzlich müssen Sie wie in Listing 5.8 die Methode *SAVE* aufrufen, um die Verbuchung anzustoßen. Erst dadurch werden die Daten auf der Datenbanktabelle erzeugt.

> **Standardwerte**
>
> Für einen Knoten können über eine optionale Knotenklasse Standardwerte für die Anlage hinterlegt werden. Wie Sie die dazugehörigen Standardwerte auslesen, können Sie in Abschnitt 5.5, »Standardwerte für Knoten auslesen«, nachlesen.

5.3.2 Anlegen von Unterknoten

Das Anlegen für Unterknoten funktioniert analog zum Anlegen eines Hauptknotens, indem Sie im Feld *NODE* den Namen des gewünschten anzulegenden Knotens angeben. Zusätzlich müssen Sie noch im Feld *SOURCE_NODE* die ID des Oberknotens, im Feld *SOURCE_KEY* den Schlüssel des Oberknotens und im Feld *ASSOCIATION* die dazugehörige Assoziation vom Ober- zum Unterknoten angeben. Dadurch legen Sie letztlich auch immer eine Assoziationsverbindung mit an.

Diese Assoziationsverbindung können Sie im Konstanten-Interface des Geschäftsobjekts unter der Konstante *SC_ASSOCIATION*, gefolgt von dem gewünschten Knoten, finden. Das in diesem Abschnitt gezeigte Beispiel legt hierzu eine neue Instanz des Knotens *ROOT_TEXT* zum Oberknoten *ROOT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER* an und ist als Fortsetzung des im vorherigen Abschnitt 5.3.1 gezeigten Anlegens eines Hauptknotens zu verstehen. Abbildung 5.1 zeigt nochmals die Knotenhierarchie des Geschäftsobjekts.

Abbildung 5.1 Knotenstruktur von /BOBF/DEMO_CUSTOMER

Da der Knoten *ROOT_TEXT* unter dem Knoten *ROOT* hängt, wird als *SOURCE_KEY* derselbe Schlüssel aus dem in Abschnitt 5.3.1 angelegten Knoten *ROOT* benutzt. Das bedeutet, dass der hier neu angelegte Text dem Kunden mit der Kundennummer 123456 zugewiesen wird, wie es Listing 5.10 zeigt.

```
"Deklaration der Variablen und Beschaffung des Service-Managers und
"Transaktions-Managers wie in Listing 5.8
DATA lr_cust_root_txt TYPE REF TO /bobf/s_demo_short_text_k.
"Erzeugen der neuen Knoten-Struktur eines Kurztextes CREATE DATA lr_cust_root_txt.
lr_cust_root_txt->key      = lo_serv_mgr->get_new_key( ).
lr_cust_root_txt->text     = 'Beispiel-Text'.
lr_cust_root_txt->language = sy-langu.

ls_modi-node = /bobf/if_demo_customer_c=>sc_node-root_text.
ls_modi-change_mode = /bobf/if_frw_c=>sc_modify_create.
ls_modi-source_node = /bobf/if_demo_customer_c=>sc_
                      node-root.
ls_modi-association = /bobf/if_demo_customer_c=>sc_
                      association-root-root_text.
ls_modi-source_key  = lr_cust_root->key. "aus
ls_modi-key         = lr_cust_root_txt->key.
ls_modi-data        = lr_cust_root_txt.
APPEND ls_modi TO lt_modi.
lo_serv_mgr->modify(
  EXPORTING
    it_modification = lt_modi
```



```
  IMPORTING
    eo_message      = lo_message
).
```

Listing 5.10 Anlegen eines Unterknotens

Ob das Anlegen erfolgreich war, können Sie wie in Listing 5.8 mit dem Rückgabeparameter *LO_MESSAGE* ermitteln. Zusätzlich müssen Sie wie in Listing 5.8 die Methode *SAVE* aufrufen, um die transaktionale Verarbeitung abzuschließen. Erst dadurch werden die Daten auf der Datenbanktabelle angelegt.

5.3.3 Aktualisieren von Knoteninstanzen

Das Aktualisieren einer Knoteninstanz funktioniert vom Prinzip her so wie das Anlegen derselben – mit dem Unterschied, dass Sie in der internen Tabelle des Feldes *CHANGED_FIELDS* der Modifikationsstruktur angeben, welche Felder aus den im Feld *DATA* mitgegebenen Daten geändert werden sollen. Ohne diese Angabe können keine Attribute der Knoteninstanz aktualisiert werden.

Listing 5.11 zeigt hier beispielhaft das Aktualisieren der Attribute *ADDRESS* (Adresse) und *CUST_CURR* (Kundenwährung) des Knotens *ROOT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER*. Die Adresse wird auf eine neue Straße geändert, während die Kundenwährung auf initial gesetzt wird. Im Feld *KEY* ist ein beispielhafter Schlüssel einer Knoteninstanz des Knotens *ROOT* angegeben.

```
"Deklaration der Variablen und Beschaffung des Service-
 Managers und Transaktions-Managers wie in Listing 5.8
"Für die Angabe der geänderten Attribute
DATA: lt_changed_fields TYPE /bobf/t_frw_name.

DATA: lr_cust_root TYPE REF TO /bobf/s_demo_customer_hdr_k.
"Erzeugen der neuen Knoten-Struktur als Datenobjekt
CREATE DATA lr_cust_root.

"Spezifizieren welche Felder geändert werden sollen
APPEND 'ADDRESS'   TO lt_changed_fields.
APPEND 'CUST_CURR' TO lt_changed_fields.
"Auch Angabe der Attribute über das Konstanten-Interface möglich:
"/bobf/if_demo_customer_c=>sc_query_attribute-root-
"select_by_attributes-cust_curr.

"Neue Werte setzen
lr_cust_root->address = 'Mühlenstraße 5'.
```

```
CLEAR lr_cust_root->cust_curr.

ls_modi-data = lr_cust_root.
ls_modi-node = /bobf/if_demo_customer_c=>sc_node-root.
ls_modi-change_mode = /bobf/if_frw_c=>sc_modify_update.
"Beispielhafter Schlüssel
ls_modi-key            = '0050568F7290...1ED79EA63B400C91A0FF'.
ls_modi-changed_fields = lt_changed_fields.
APPEND ls_modi TO lt_modi.

lo_serv_mgr->modify(
  EXPORTING
    it_modification = lt_modi
  IMPORTING
    eo_message      = lo_message
).
```

Listing 5.11 Aktualisieren von Attributen einer Knoteninstanz

Ob das Aktualisieren erfolgreich war, können Sie wie in Listing 5.8 über den Rückgabeparameter *LO_MESSAGE* ermitteln. Zusätzlich müssen Sie wie in Listing 5.8 die Methode *SAVE* aufrufen, um die transaktionale Verarbeitung abzuschließen. Erst dadurch werden die Daten auf der Datenbanktabelle aktualisiert.

5.3.4 Löschen von Knoteninstanzen

Zum Löschen von Knoteninstanzen müssen Sie bei der Methode *MODIFY* lediglich den Knotennamen im Feld *NODE*, den Ändern-Modus *SC_MODIFY_DELETE* im Feld *CHANGE_MODE* und den Schlüssel des zu löschenden Knotens im Feld *KEY* der Änderungsstruktur angeben. Unterknoten können Sie auf dieselbe Art und Weise löschen. Geben Sie dazu bei *NODE* den Namen des Unterknotens und im Feld *KEY* den Schlüssel des zu löschenden Unterknotens an.

Löschen eines gesamten Geschäftsobjekts

Wenn Sie die Instanz eines Hauptknotens (*ROOT*) eines Geschäftsobjekts löschen, werden alle dazugehörigen Unterknoten mit gelöscht.

In Listing 5.12 können Sie das Löschen einer Knoteninstanz des Knotens *ROOT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER* sehen. Im Feld *KEY* ist ein beispielhafter Schlüssel einer Knoteninstanz des Knotens *ROOT* angegeben.


```
"Deklaration der Variablen und Beschaffung des Service-Managers
 und Transaktions-Managers wie in Listing 5.8
"Befüllen der Modifikations-Tabelle
ls_modi-node = /bobf/if_demo_customer_c=>sc_node-root.
ls_modi-change_mode = /bobf/if_frw_c=>sc_modify_delete.
ls_modi-key = '0050568F7290...1ED79EA31FECBD852OFF'. "Beispiel
APPEND ls_modi TO lt_modi.

lo_serv_mgr->modify(
  EXPORTING
    it_modification = lt_modi
  IMPORTING
    eo_message      = lo_message
).
```

Listing 5.12 Löschen einer Knoteninstanz

Ob das Löschen erfolgreich war, können Sie wie in Listing 5.8 über den Rückgabeparameter *LO_MESSAGE* ermitteln. Zusätzlich müssen Sie wie in Listing 5.8 die Methode *SAVE* aufrufen, um die transaktionale Verarbeitung abzuschließen. Erst dadurch werden die Daten auf der Datenbanktabelle gelöscht.

5.4 Sperren/Entsperren

Sperren werden beim Bearbeiten der Geschäftsobjekte über die Methode *MODIFY* wie in Abschnitt 5.3 beschrieben automatisch gesetzt. Dabei handelt es sich prinzipiell um exklusive Sperren. Das bedeutet, dass nur genau ein Benutzer das gesperrte Geschäftsobjekt (bzw. dessen gesperrte Knoteninstanzen) bearbeiten darf und alle anderen Änderungsanfragen abgewiesen werden. Beim Speichern (Methode *SAVE*) oder Zurücksetzen der Daten (Methode *CLEANUP*) mit dem Transaktionsmanager werden die so gesetzten Sperren automatisch wieder entfernt. Für Sie bedeutet das, dass Sie keine Sperren manuell setzen müssen, da dies das Framework vollautomatisch übernimmt.

In bestimmten Fällen kann es dennoch vorkommen, dass Sie eine Sperre setzen wollen, ohne direkt eine Änderung mit der Methode *MODIFY* durchzuführen. Beispielsweise wollen Sie ein Geschäftsobjekt bereits sperren, wenn der Benutzer Ihr Programm (bzw. die dazugehörige Transaktion) im Bearbeitungsmodus aufruft. Vielleicht wollen Sie aber auch mehrere Instanzen eines Geschäftsobjekts in einer Liste anbieten, und erst der Benutzer entscheidet nach der Auswahl, welche Instanz er letztlich bearbeiten möchte. Um ein solches Geschäftsobjekt für zukünftige Änderungen zu sperren, können Sie die Methode *RETRIEVE* benutzen. Hierfür benötigen

Sie nur den Schlüssel der Instanz des zu sperrenden Geschäftsobjekts. Über den Importparameter *IV_EDIT_MODE* derselben Methode können Sie anschließend angeben, welche Art der Sperre verwendet werden soll.

Dafür stehen Ihnen die folgenden Sperrarten zur Verfügung (es gibt im Framework weitere [obsolete] Sperrarten, aber für alle anderen Sperroptionen als die hier aufgeführten wirft die Methode eine Ausnahme):

- für die *exklusive* Sperre die Konstante */BOBF/IF_CONF_C=>SC_EDIT_EXCLUSIVE*
  In diesem Standardfall darf nur der Benutzer, der die Sperre setzt, das Geschäftsobjekt bearbeiten. Alle anderen Sperranfragen werden abgewiesen.
- für die *optimistische* Sperre die Konstante */BOBF/IF_CONF_C=>SC_EDIT_OPTIMISTIC*
  Bei dieser Variante kann eine Sperre von mehreren Benutzern gleichzeitig gesetzt werden. Sobald einer der Benutzer tatsächlich eine Änderung (durch Speichern) am Geschäftsobjekt vornimmt, werden alle anderen optimistischen Sperren gelöscht, und seine Sperre wird in eine exklusive Sperre umgewandelt. Möchte ein anderer Benutzer nun das Geschäftsobjekt ändern, wird diese Anfrage abgewiesen. Eine exklusive Sperre gewinnt also immer gegen eine optimistische Sperre. Wird die Änderung am Geschäftsobjekt gespeichert, wird die exklusive Sperre entfernt, und die vorherigen optimistischen Sperren werden reaktiviert.

Listing 5.13 zeigt beispielhaft, wie eine Instanz des Geschäftsobjekts */BOBF/DEMO_CUSTOMER* exklusiv gesperrt wird. Wir empfehlen Ihnen, den Parameter *IV_FILL_DATA* aus Performancegründen mit *abap_false* zu belegen, damit Sie hier nicht unnötig die Daten der Instanz zurückbekommen.

```
DATA: lt_keys TYPE /bobf/t_frw_key,
      ls_key  TYPE /bobf/s_frw_key.
"beispielhafter Schlüssel
ls_key-key = '0050568F72901ED79EA326CDC37C60FF'.
APPEND ls_key TO lt_keys.

lo_serv_mgr->retrieve(
    iv_node_key   = /bobf/if_demo_customer_c=>sc_node-root
    it_key        = lt_keys
    iv_edit_mode  = /bobf/if_conf_c=>sc_edit_exclusive
    iv_fill_data  = abap_false ).
```

Listing 5.13 Sperren einer Knoteninstanz mit RETRIEVE

Sie können die Instanzen mit der Methode *RETRIEVE* nicht entsperren (auch dann nicht, wenn Sie im Parameter *IV_EDIT_MODE = /BOBF/IF_CONF_C=>SC_EDIT_READ_ONLY* angeben).

Die Instanzen werden immer nur dann entsperrt, wenn Sie die Methoden *SAVE* oder *CLEANUP* des Transaktions-Managers aufrufen. So wird sichergestellt, dass es zu keinen schiefen Datenbeständen kommen kann. Listing 5.14 zeigt dies noch einmal in Kurzfassung.

```
"Entsperren
lo_trans_mgr->save( ).
"Oder (in der Regel bei Misserfolg von SAVE):
lo_trans_mgr->cleanup( ).
```

Listing 5.14 Speichern und Aufräumen mit dem Transaktions-Manager

5.5 Standardwerte für Knoten auslesen

Zu jedem Knoten kann eine Knotenklasse angelegt werden, die das Interface */BOBF/IF_FRW_NODE* mit der Methode *RETRIEVE_DEFAULT_VALUES* implementiert.

Anlegen einer Knotenklasse

Die Anlage und Implementierung einer solchen Klasse über das Interface */BOBF/IF_FRW_NODE* haben wir in Abschnitt 6.2, »Knoten implementieren«, beschrieben.

Im Folgenden zeigen wir Ihnen, wie Sie diese Werte programmatisch für die Anlage eines neuen Haupt- und Unterknotens ermitteln können.

5.5.1 Standardwerte für Oberknoten auslesen

Sind für einen Hauptknoten Standardwerte definiert worden, können Sie diese über die Methode *RETRIEVE_DEFAULT_NODE_VALUES* des Service-Managers wie hier beschrieben auslesen. Dazu müssen Sie im Changing-Parameter *CT_DATA* der Methode die erzeugte leere Knotenstruktur des gewünschten *ROOT*-Knotens als interne Tabelle übergeben. Die Methode wird diese so übergebenen Zeilen mit den in der Knotenklasse ermittelten Standardwerten anreichern. Listing 5.15 zeigt dies beispielhaft für den *ROOT*-Knoten des Geschäftsobjekts */BOBF/DEMO_CUSTOMER*.

```
"Erzeugen der neuen Knoten-Struktur als Datenobjekt
DATA: lr_cust_root TYPE REF TO /bobf/s_demo_customer_hdr_k.
CREATE DATA lr_cust_root.

"Erzeugen eines neuen Schlüssels
lr_cust_root->key = lo_serv_mgr->get_new_key( ).
```

```
"Ab hier: Beschaffung der Standard-Werte
DATA: lt_cust_root TYPE TABLE OF /bobf/s_demo_customer_hdr_k.
FIELD-SYMBOLS: <s_cust_root> TYPE /bobf/s_demo_customer_hdr_k.

"Hinzufügen der neuen Knotenstruktur zur Übergabetabelle
ASSIGN lr_cust_root->* TO <s_cust_root>.
APPEND <s_cust_root> TO lt_cust_root.

"Aufruf der Anreicherung
lo_serv_mgr->retrieve_default_node_values(
  EXPORTING
    iv_node_key = /bobf/if_demo_customer_c=>sc_node-root
  CHANGING
    ct_data     = lt_cust_root
).
```

Listing 5.15 Auslesen von Standardwerten für einen Oberknoten

5.5.2 Standardwerte für Unterknoten auslesen

Auch für Unterknoten können Standardwerte implementiert und später durch die Anwendung ausgelesen werden. Das Auslesen funktioniert wie bei Oberknoten mit der Methode *RETRIEVE_DEFAULT_NODE_VALUES*. So müssen Sie auch hier die vorbereiteten leeren Knotenstrukturen als interne Tabelle im Changing-Parameter *CT_DATA* mitgeben.

Anders als beim Auslesen für den Hauptknoten reicht es allerdings nicht mehr aus, nur den Importparameter *IV_NODE_KEY* mit dem gewünschten Knoten zu belegen. Das liegt daran, dass das Framework versucht, bei einem Unterknoten die dazugehörigen Standardwerte kontextabhängig (d. h. abhängig von den Attributen der dazugehörigen Instanz des Oberknotens) zu bestimmen. Daher müssen Sie zusätzlich folgende Einstellungen vornehmen:

- im Parameter *IV_SOURCE_KEY* den Schlüssel der Instanz des zum Unterknoten gehörigen Oberknotens mitgeben
- im Parameter *IV_ASSOC_KEY* den Schlüssel (über das Konstanten-Interface) der dazugehörigen Assoziation vom Ober- zum Unterknoten mitgeben

Listing 5.16 zeigt dies für den Knoten *ROOT_TEXT* als Unterknoten von *ROOT* für das Geschäftsobjekt */BOBF/DEMO_SALES_ORDER*, der zu dem Kurztext die Sprache des Textes in Abhängigkeit zum Verkaufsauftragskopf bestimmt.

```
DATA: lt_root_text TYPE TABLE OF /bobf/s_demo_short_text_k,
      lr_root_text TYPE REF TO   /bobf/s_demo_short_text_k.
```



```
FIELD-SYMBOLS: <s_root_text> TYPE /bobf/s_demo_short_text_k.

CREATE DATA lr_root_text.
lr_root_text->key = /bobf/cl_frw_factory=>get_new_key( ).

ASSIGN lr_root_text->* TO <s_root_text>.
APPEND <s_root_text> TO lt_root_text.

lo_mgr->retrieve_default_node_values(
  EXPORTING
    iv_node_key = /bobf/if_demo_sales_order_c=>sc_node-root_text
    iv_source_key = '0050568F72901ED7A5949F1AA5B940FF'
 "Schlüssel der ROOT-Instanz!
    iv_assoc_key = /bobf/if_demo_sales_order_c=>sc_association-
                   root-root_text
  CHANGING
    ct_data = lt_root_text
).
```

Listing 5.16 Auslesen von Standardwerten für einen Unterknoten

Abbildung 5.2 zeigt die mit der Sprache (Spalte **LANGUAGE**) angereicherte Struktur des Kurztextes.

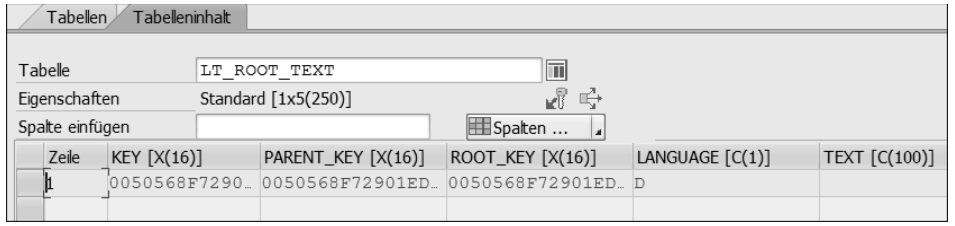

Abbildung 5.2 Angereicherte Struktur des Knotens ROOT_TEXT

5.6 Hauptknoten auslesen

Um zu einem Unterknoten den dazugehörigen Hauptknoten (*ROOT*) auszulesen, können Sie die Methode *GET_ROOT_KEY* verwenden. Diese wird allerdings nicht wie die anderen hier vorgestellten Methoden vom Service-Manager, sondern vom BOPF-Service-Layer bereitgestellt, den Sie mit der Methode *GET_BOPF* der Klasse */BOBF/CL_FRW_FACTORY* über den Schlüssel des Geschäftsobjekts auslesen können. Zum Aufruf der Methode *GET_ROOT_KEY* müssen Sie lediglich im Importparameter *IV_NODE_KEY* die ID des Knotens und im Importparameter *IT_KEY* die Schlüssel zum angegebenen Knoten mitgeben.

Listing 5.17 zeigt beispielhaft, wie für den Unterknoten *ROOT_TEXT* des Geschäftsobjekts */BOBF/DEMO_CUSTOMER* die dazugehörige Instanz des Hauptknotens ermittelt wird.

```
DATA: lt_key_link   TYPE /bobf/t_frw_key_link,
      lt_failed_key TYPE /bobf/t_frw_key,
      lt_target_key TYPE /bobf/t_frw_key.

DATA: lt_data TYPE /bobf/t_demo_short_text_k,
      lt_key  TYPE /bobf/t_frw_key,
      ls_key  TYPE /bobf/s_frw_key.

DATA: lo_customer TYPE REF TO /bobf/if_frw_service_layer.
lo_customer = /bobf/cl_frw_factory=>get_bopf(
  EXPORTING
    iv_bo_key    = /bobf/if_demo_customer_c=>sc_bo_key
).

ls_key-key = '0050568F7290ED79EB7C7DA5BFA00FF'.
APPEND ls_key TO lt_key.

lo_customer->get_root_key(
  EXPORTING
    iv_node_key     = /bobf/if_demo_customer_c=>sc_node-root_text
    it_key          = lt_key
  IMPORTING
    et_key_link     = lt_key_link
    et_failed_key   = lt_failed_key
    et_target_key   = lt_target_key
).
```

Listing 5.17 Beschaffung des Schlüssels des Hauptknotens

Die ermittelten Hauptknoten können Sie in der Rückgabetabelle *ET_TARGET_KEY* auslesen. Dann können Sie auf diese Knoten weitere Methoden, z. B. ein *RETRIEVE* oder *MODIFY*, anwenden. Konnten für bestimmte Schlüssel keine Hauptknoten bestimmt werden, werden diese Schlüssel in der Rückgabetabelle *ET_FAILED_KEY* zurückgegeben. Zusätzlich gibt Ihnen die Methode *GET_ROOT_KEY* in den Rückgabetabellen *LT_KEY_LINK* die komplette Verknüpfungskette vom angegebenen Knoten zum Hauptknoten zurück.

5.7 Abfragen

In Abschnitt 5.1.1, »Die Methode QUERY«, haben wir Ihnen bereits die grundlegende Implementierung des Aufrufs einer Abfrage erläutert. In diesem Abschnitt möchten wir Ihnen darüber hinaus zeigen, wie Sie über den optionalen Parameter *IS_QUERY_OPTIONS* noch weitere Abfrageoptionen implementieren können. Zusätzlich demonstrieren wir Ihnen, wie Sie über die Methode *RETRIEVE_DEFAULT_QUERY_PARAM* des Service-Managers Standard-Abfrageparameter für eine Abfrage auslesen können.

5.7.1 Weitere Abfrageoptionen

Mit dem optionalen Parameter *IS_QUERY_OPTIONS* der Methode *QUERY* können Sie zusätzlich Folgendes ausführen:

- die maximale Ergebnismenge einschränken
- die Ergebnismenge sortieren
- den Paging-Mechanismus aktivieren

Den grundlegenden Aufruf für alle drei Möglichkeiten finden Sie in Listing 5.18 (beispielhaft an der Abfrage *SELECT_ALL*).

```
lo_serv_mgr->query(
  EXPORTING
    iv_query_key     = /bobf/if_demo_customer_c=>sc_query-
                       root-select_all
    iv_fill_data     = abap_true
    is_query_options = ls_query_options
  IMPORTING
    et_data          = lt_data
    et_key           = lt_keys
).
```

Listing 5.18 Angabe von weiteren Abfrageoptionen

Die maximale Ergebnismenge angeben

Um das klassische *UP TO ... ROWS* einer SQL-Anweisung durchzuführen, müssen Sie die gewünschte maximale Anzahl der Ergebnismenge im Feld *MAXIMUM_ROWS* des Parameters *IS_QUERY_OPTIONS* angeben:

```
DATA: ls_query_options TYPE /bobf/s_frw_query_options.
ls_query_options-maximum_rows = 1.
```

Die Ergebnismenge sortieren

Um das klassische *ORDER BY* einer SQL-Anweisung durchzuführen, müssen Sie lediglich eine neue Zeile zum Feld *SORTING_OPTIONS* des Parameters *IS_QUERY_OPTIONS* hinzufügen. In dieser neuen Zeile muss im Feld *ATTRIBUTE_NAME* das zu sortierende Attribut und im Feld *ASCENDING* die Sortierrichtung (*abap_false* für absteigend, *abap_true* für aufsteigend) angegeben werden. Listing 5.19 zeigt beispielhaft für das Attribut *CUST_CURR* (Währung des Kunden) eine absteigende Sortierung.

```
DATA: ls_sort_options  TYPE /bobf/s_frw_query_sorting,
      ls_query_options TYPE /bobf/s_frw_query_options.

ls_sort_options-attribute_name = /bobf/if_demo_customer_c=>sc_query_attribute-
                                 root-select_by_attributes-cust_curr.
"Oder direkte Angabe des Namen des Attributs CUST_CURR
ls_sort_options-ascending = ''.
"oder X für aufsteigende Sortierung
APPEND ls_sort_options TO ls_query_options-sorting_options.
```

Listing 5.19 Sortieren der Ergebnismenge

Den Paging-Mechanismus aktivieren

Der *Paging*-Mechanismus dient dazu, nur eine gewisse Teilmenge von Knoteninstanzen zu bestimmen, und ist vor allem für Tabellen auf der Oberfläche gedacht, auf denen geblättert werden kann. Abbildung 5.3 zeigt eine solche Tabelle aus einer SAPUI5-Anwendung.

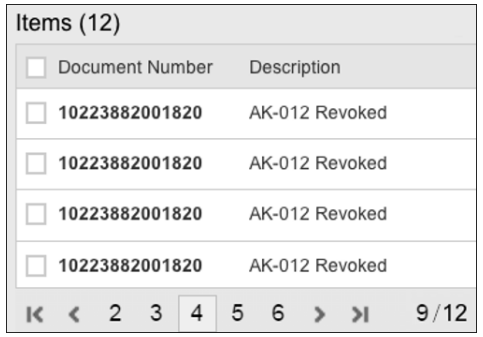

Abbildung 5.3 Tabelle mit Paging-Funktion

Da die Performance auf diesen webbasierten Anwendungen kritisch ist, ist es wichtig, dass nur so wenige Daten wie möglich geladen und auf dem Endgerät (z. B. Handy) vorgehalten werden. Konkret bedeutet dies für die Tabelle aus Abbildung 5.3: Erst wenn die vierte Seite angeklickt wird, werden die Einträge 13 bis 16 geladen. Für Sie bedeutet das bei der Implementierung der Datenbeschaffung: Sie müssen nur die

Knoteninstanzen 13 bis 16 selektieren und sollten die ersten zwölf Knoteninstanzen überspringen.

> **OpenSQL und der Zusatz »OFFSET«**
>
> In OpenSQL konnte man dieses Problem bis Version 7.51 nicht performant lösen. Erst durch das Hinzufügen des Zusatzes *OFFSET* (Angabe der zu überspringenden Zeilen) konnte dies performant implementiert werden. Vorher mussten im genannten Beispiel alle 16 Einträge selektiert und anschließend die ersten zwölf Instanzen mithilfe von ABAP wieder aus der internen Tabelle gelöscht werden.

Im BOPF übernimmt das Framework diese Aufgabe für Sie.

Implementieren können Sie eine solche Funktion wie folgt: Aktivieren Sie die Paging-Option, indem Sie in der Struktur *PAGING_OPTIONS* des Parameters *IS_QUERY_OPTIONS* im Feld *PAGING_ACTIVE* ein X mitgeben. Geben Sie zusätzlich im Feld *START_ROW* die Anzahl der zu überspringenden Zeilen und im Feld *MAXIMUM_ROWS* die maximal zu selektierenden Zeilen an. Listing 5.20 zeigt dies beispielhaft um die Zeile 13 bis 16 zu laden.

```
DATA: ls_query_options TYPE /bobf/s_frw_query_options.
ls_query_options-maximum_rows = 4.
ls_query_options-paging_options-paging_active = 'X'.
ls_query_options-paging_options-start_row = 13.
```

Listing 5.20 Umsetzung der Paging-Funktion

5.7.2 Standard-Abfrageparameter

Bei Abfragen, die nicht durch das Framework als Standard vorgegeben sind, sondern durch eigene Abfrageklassen über das Interface */BOBF/IF_FRW_QUERY* implementiert werden, gibt es mit der Methode *RETRIEVE_DEFAULT_PARAM* die Möglichkeit, für die implementierte Abfrage Standardwerte vorzugeben.

> **Anlegen einer Abfrage**
>
> Wie Sie eine solche Abfrage anlegen und das dazugehörige Interface */BOBF/IF_FRW_QUERY* implementieren, können Sie in Abschnitt 6.6, »Abfragen implementieren«, nachlesen.

Um zu sehen, ob eine Abfrage die Vorgabemethode implementiert hat oder nicht, können Sie über die Abfrage in die dazugehörige Abfrageklasse der Methode *RETRIEVE_DEFAULT_PARAM* springen. Ist diese beispielsweise wie in Abbildung 5.4 nicht leer, sondern mit Quellcode (für das Geschäftsobjekt */IAM/ISSUE*, Knoten *ROOT*,

Abfrage *MOC_WORKLIST*) gefüllt, können Sie aus Ihrer Anwendung heraus diese Standardwerte übernehmen.

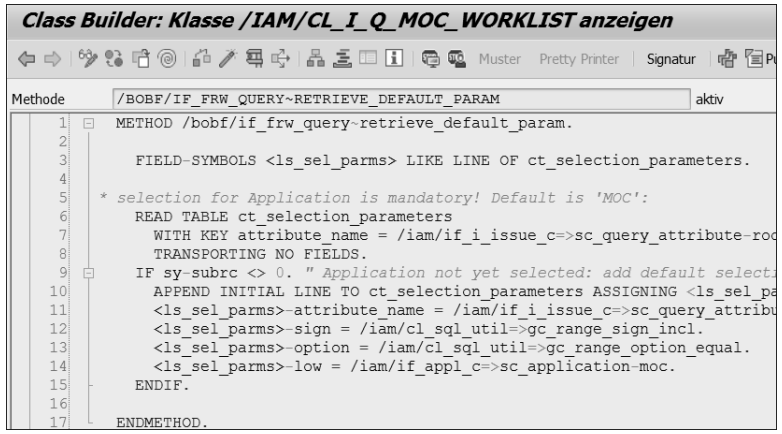

Abbildung 5.4 Definition von Standardwerten für eine Abfrage

Das Auslesen der Standardwerte geschieht mit der Methode *RETRIEVE_DEFAULT_QUERY_PARAM* des Service-Managers. Listing 5.21 zeigt eine beispielhafte Implementierung für die zuvor erwähnte Abfrage *MOC_WORKLIST*.

```
DATA: lt_selection_param TYPE /bobf/t_frw_query_selparam.
lo_serv_mgr->retrieve_default_query_param(
 EXPORTING
  iv_query_key = /iam/if_i_issue_c=>sc_query-root-moc_worklist
 CHANGING
   ct_selection_parameters = lt_selection_param
).
```

Listing 5.21 Auslesen von Standardparametern für eine Abfrage

Abbildung 5.5 zeigt die vom Aufruf aus Listing 5.21 zurückgegebenen Standardwerte. In diesem Fall wird das Attribut *APPLICATION* mit dem Wert *MOC* als Vergleich gesetzt.

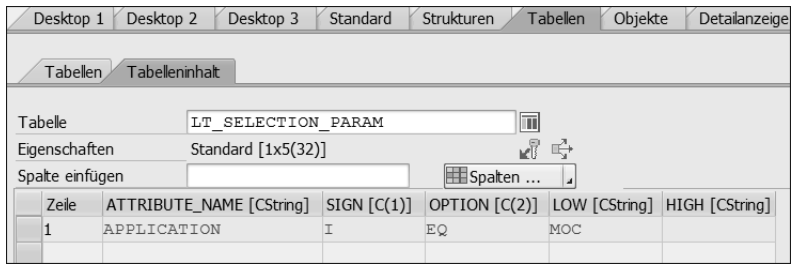

Abbildung 5.5 Rückgabetabelle der Standardwerte für eine Abfrage

5.8 Aktionen

Zur Arbeit mit Aktionen bietet der Service-Manager drei Methoden an:

- die Methode *DO_ACTION* zum Aufrufen einer Aktion
- die Methode *RETRIEVE_DEFAULT_ACTION_PARAM* zur Ermittlung von Standard-Aktionsparametern für den Aufruf der Aktion mit *DO_ACTION*
- die Methode *CHECK_ACTION* zur Überprüfung, ob eine Aktion auf bestimmte Knoteninstanzen ausgeführt werden kann (diese Prüfung wird in Abschnitt 5.9, »Validierungen«, beschrieben)

Aktion anlegen

Wie Sie eine solche Abfrage anlegen und das dazugehörige Interface */BOBF/IF_FRW_ACTION* implementieren, haben wir in Abschnitt 6.3, »Aktionen implementieren«, beschrieben.

5.8.1 Aktionen aufrufen

Mit der Methode *DO_ACTION* können Sie eine Aktion eines Geschäftsobjekts aufrufen. Für den grundlegenden Aufruf müssen Sie lediglich im Eingabeparameter *IV_ACT_KEY* den Schlüssel der Aktion über das Konstanten-Interface mitgeben und im Eingabeparameter *IT_KEY* die Schlüssel der Knoteninstanzen mitgeben, gegen die die Aktion ausgeführt werden soll. Je nach Erfolg der Ausführung sind die Rückgabeparameter unterschiedlich gefüllt:

- Schlägt die Ausführung der Aktion fehl, ist die Rückgabetabelle *ET_FAILED_ACTION_KEY* mit dem Schlüssel der Aktion gefüllt, während die Rückgabetabelle *ET_FAILED_KEY* die fehlgeschlagenen Schlüssel der Knoteninstanzen enthält.
- War die Durchführung der Aktion erfolgreich, ist die Rückgabetabelle *ET_FAILED_ACTION_KEY* leer. Zusätzlich wird, falls in der Aktion eine Rückgabestruktur definiert ist, die Rückgabetabelle *ET_DATA* mit Daten gefüllt.

Listing 5.22 zeigt eine beispielhafte Implementierung des Aufrufs einer Aktion. Hierbei wird die Aktion *ARCHIVE* des Geschäftsobjekts */BOBF/DEMO_SALES_ORDER* ausgeführt.

```
DATA: lt_keys          TYPE /bobf/t_frw_key,
      ls_key           TYPE /bobf/s_frw_key,
      lt_failed_keys   TYPE /bobf/t_frw_key,
      lt_failed_action TYPE /bobf/t_frw_key,
      lo_message       TYPE REF TO /bobf/if_frw_message.
```

```
ls_key-key = '0050568F72901ED7A5949F1AA5B940FF'.
APPEND ls_key TO lt_keys.

lo_serv_mgr->do_action(
  EXPORTING
    iv_act_key  = /bobf/if_demo_sales_order_c=>sc_
                  action-root-archive
    it_key      = lt_keys
  IMPORTING
    eo_message           = lo_message
    et_failed_key        = lt_failed_keys
    et_failed_action_key = lt_failed_action
*    et_data = "falls Aktion eine Rückgabe-Struktur hat
).
```

Listing 5.22 Grundlegender Aufruf einer Aktion

Wenn in der Aktion eine Eingabestruktur definiert ist, können Sie diese über den Eingabeparameter *IS_PARAMETERS* mitgeben. Vor der Übergabe müssen Sie die dahinterliegende Struktur mit *TYPE REF TO* definieren und mit *CREATE DATA* erzeugen.

Listing 5.23 zeigt hierfür eine beispielhafte Implementierung. Dabei wird versucht, die Position 10 eines Verkaufsauftrags zu beliefern. Dies geschieht über die Aktion *DELIVER* des Geschäftsobjekts */BOBF/DEMO_SALES_ORDER*. Als Importparameter wird die Position 10 mitgegeben. Die für den Aufruf benötigten Datendeklarationen bauen auf Listing 5.22 auf. Für die angegebene Variable *LR_DEL_IMPORT* wurde deshalb die Struktur */BOBF/S_DEMO_SALES_ORDER_HDR_D* verwendet, weil diese Importstruktur so in den Eigenschaften der Aktion im Geschäftsobjekt definiert worden ist. Diese Struktur kann für jede Aktion unterschiedlich sein.

```
"Erzeugung der Import-Struktur für die Aktion
DATA: lr_del_import TYPE REF TO /bobf/s_demo_sales_order_hdr_d.
CREATE DATA lr_del_import.
lr_del_import->item_no = '010'.

lo_serv_mgr->do_action(
  EXPORTING
    iv_act_key    = /bobf/if_demo_sales_order_c=>sc_
                    action-root-deliver
    it_key        = lt_keys
    is_parameters = lr_del_import
```



```
  IMPORTING
    eo_message           = lo_message
    et_failed_key        = lt_failed_keys
    et_failed_action_key = lt_failed_action
*    et_data = "falls Aktion eine Rückgabe-Struktur hat
).
```

Listing 5.23 Aufruf einer Aktion mit Importparametern

5.8.2 Standardparameter für Aktionen auslesen

Für eine Aktion können Standardparameter hinterlegt worden sein. Diese können Sie über die Methode *RETRIEVE_DEFAULT_ACTION_PARAM* auslesen. Für den Aufruf der Methode müssen Sie im Eingabeparameter *IV_ACT_KEY* den Schlüssel der Aktion über das Konstanten-Interface mitgeben und im Eingabeparameter *IT_KEY* die Schlüssel der Knoteninstanzen mitgeben, für die die Standardparameter ausgelesen werden sollen.

Je nach Implementierung können die Standardparameter nicht nur statisch innerhalb der Aktionsklasse, sondern auch dynamisch, d. h. abhängig von den Knoteninstanzen und bereits vorbelegten Parametern, definiert worden sein. Daher können Sie im Changing-Parameter *CS_PARAMETERS* nicht nur die ermittelten Standardparameter zurücknehmen, sondern diese auch schon vorbelegt mit in die Methode übergeben. Bei Erfolg wird die in dem Parameter *CS_PARAMETERS* angegebene Struktur angereichert. Konnten keine Standardparameter ermittelt werden (oder ist für die Methode die Aktion nicht implementiert), bleibt die Struktur unverändert.

Listing 5.24 zeigt für die Methode eine beispielhafte Implementierung für die Aktion *DELIVER* des Knotens *ROOT* des Geschäftsobjekts */BOBF/DEMO_SALES_ORDER*.

```
DATA: ls_param TYPE REF TO data.
CREATE DATA ls_param TYPE /bobf/s_demo_sales_order_hdr_d.

lo_serv_mgr->retrieve_default_action_param(
  EXPORTING
    iv_act_key = /bobf/if_demo_sales_order_c=>sc_
                 action-root-deliver
    it_key     = lt_keys
  CHANGING
    cs_parameters = ls_param
).
```

Listing 5.24 Ermittlung von Standard-Aktionsparametern

5.9 Validierungen

Zur Ausführung von Validierungen bietet der Service-Manager drei Methoden an:

- die Methode *CHECK_CONSISTENCY* zur Durchführung von Konsistenz-Validierungen für Knoteninstanzen
- die Methode *CHECK_AND_DETERMINE* zur Durchführung von Konsistenz-Validierungen und die dazugehörigen Ermittlungen für Knoteninstanzen
- die Methode *CHECK_ACTION* für die Validierung von Aktionen

Anlegen einer Validierung

Wie Sie eine solche Abfrage anlegen und das dazugehörige Interface */BOBF/IF_FRW_VALIDATION* implementieren, lesen Sie in Abschnitt 6.5, »Validierungen implementieren«.

5.9.1 Konsistenz-Validierungen

Konsistenz-Validierungen prüfen, ob eine bestimmte Menge von Knoteninstanzen valide, d. h. ob Ihre Attributwerte in sich schlüssig sind. Diese Konsistenz-Validierungen werden automatisch gemäß der eingestellten Ausführungszeitpunkte und der eingestellten Ausführungsreihenfolge ausgeführt. Wollen Sie eine Konsistenz-Validierung dennoch manuell ausführen, können Sie dafür die Methoden *CHECK_CONSISTENCY* und *CHECK_AND_DETERMINE* nutzen. In diesem Abschnitt ist nur die erste Methode beschrieben. Die zweite Methode *CHECK_AND_DETERMINE* beschreiben wir in Abschnitt 5.10, »Ermittlungen«, da diese Methode auch die zu den Knoten zugehörigen Ermittlungen ausführt.

Zur Ausführung der Methode *CHECK_CONSISTENCY* müssen Sie im Parameter *IV_NODE_KEY* den zu prüfenden Knoten über das Konstanten-Interface angeben und die dazugehörigen Knoteninstanzen im Parameter *IT_KEY* mitgeben. Anschließend können Sie über den Parameter *IV_CHECK_SCOPE* entscheiden, ob die Prüfung lokal oder für eine Teilstruktur durchgeführt werden soll:

- Lokal: Mit Angabe der Konstante */BOBF/IF_FRW_C=>SC_SCOPE_LOCAL* wird nur die aktuelle Knotenebene überprüft.
- Teilstruktur: Mit der Angabe von */BOBF/IF_FRW_C=>SC_SCOPE_SUBSTRUCTURE* werden auch alle Unterknoten überprüft.

Listing 5.25 zeigt die manuelle Validierung des Knotens *ROOT* des Geschäftsobjekts */BOBF/SALES_ORDER*.

```
DATA: lt_keys TYPE /bobf/t_frw_key,
      ls_key  TYPE /bobf/s_frw_key,
      lo_message TYPE REF TO /bobf/if_frw_message.

"Beispielhafter Schlüssel
ls_key-key = '0050568F72901ED7A5949F1AA5B940FF'.
APPEND ls_key TO lt_keys.

lo_serv_mgr->check_consistency(
  EXPORTING
    iv_node_key    = /bobf/if_demo_sales_order_c=>sc_node-root
    it_key         = lt_keys
    iv_check_scope = /bobf/if_frw_c=>sc_scope_local
  IMPORTING
    eo_message     = lo_message
).
```

Listing 5.25 Aufruf einer Konsistenz-Validierung

Der Rückgabeparameter *EO_MESSAGE* enthält nach der Ausführung die Diagnose zu den übergebenen Knoteninstanzen. Abbildung 5.6 zeigt dies beispielhaft über das im Debugger bereitgestellte Debugging-Skript (siehe Kapitel 10, »Debugging von Geschäftsobjekten«).

BOPF Framework Object: LO_MESSAGE

Business Object Name	Node Name	Node ID	Message Text	Severity
/BOBF/DEMO_SALES_ORDER	ROOT	0050568F72901ED7A5949F1AA5B940FF	Kundennummer nicht angegeben	E
/BOBF/DEMO_SALES_ORDER	ROOT	0050568F72901ED7A5949F1AA5B940FF	Währung nicht angegeben	E

Abbildung 5.6 Ergebnis einer Konsistenz-Validierung in Form eines Nachrichtenobjekts

Wie Sie programmatisch prüfen können, ob Fehlermeldungen zurückgegeben wurden und wie Sie die Fehlertexte aus diesem Nachrichtenobjekt auslesen können, beschreiben wir in Abschnitt 4.7, »Nachrichtenobjekte«.

5.9.2 Aktions-Validierungen

Aktions-Validierungen prüfen, ob auf eine bestimmte Menge von Knoteninstanzen eine Aktion ausgeführt werden kann. Dies geschieht vor der Ausführung einer Aktion automatisch. Wollen Sie dennoch eine solche Aktions-Validierung manuell ausführen, können Sie dafür die Methode *CHECK_ACTION* nutzen. Dazu müssen Sie lediglich im Eingabeparameter *IV_ACT_KEY* den Schlüssel der Aktion über das Konstanten-Interface mitgeben, den Eingabeparameter *IT_KEY* mit den Schlüsseln der Knoteninstanzen versorgen und, falls gegeben, die Aktionsparameter im Eingabeparameter *IS_PARAMETERS* übergeben.

Je nach Erfolg der Ausführung sind die Rückgabeparameter unterschiedlich gefüllt:

- Schlägt die Ausführung der Aktion fehl, ist die Rückgabetabelle *ET_FAILED_ACTION_KEY* mit dem Schlüssel der Aktion gefüllt, während die Rückgabetabelle *ET_FAILED_KEY* die fehlgeschlagenen Schlüssel der Knoteninstanzen enthält.
- War die Durchführung der Aktion erfolgreich, ist die Rückgabetabelle *ET_FAILED_ACTION_KEY* leer.

Listing 5.26 zeigt einen beispielhaften Aufruf der Methode *CHECK_ACTION*. Dazu wird geprüft, ob die Aktion *ARCHIVE* auf die Knoteninstanzen des Geschäftsobjekts */BOBF/DEMO_SALES_ORDER* ausgeführt werden kann.

```
lo_serv_mgr->check_action(
  EXPORTING
    iv_act_key = /bobf/if_demo_sales_order_c=>sc_
                 action-root-archive
    it_key     = lt_keys
IMPORTING
    eo_message           = lo_message
    et_failed_key        = lt_failed_keys
    et_failed_action_key = lt_failed_action
).
```

Listing 5.26 Aufruf einer Aktions-Validierung

5.10 Ermittlungen

Für Ermittlungen bietet Ihnen der Service-Manager nur die Methode *CHECK_AND_DETERMINE* an, die zum einen die Knoteninstanzen über die hinterlegten Validierungen und zum anderen die dafür benötigten Ermittlungen durchführt. Eine explizite Methode zur reinen Ausführung von Ermittlungen steht Ihnen als Anwender nicht zur Verfügung (nur als private Methode *DO_DETERMINATIONS* der Klasse */BOBF/CL_FRW*).

Anlegen einer Ermittlung

Wie Sie eine solche Ermittlung anlegen und das dazugehörige Interface */BOBF/IF_FRW_DETERMINATION* implementieren, können Sie in Abschnitt 6.4, »Ermittlungen implementieren«, nachlesen.

Zur Ausführung der Methode *CHECK_AND_DETERMINE* müssen Sie im Parameter *IV_NODE_KEY* den gewünschten zu prüfenden Knoten über das Konstanten-Interface angeben und die dazugehörigen Knoteninstanzen im Parameter *IT_KEY* mitgeben. Anschließend können Sie über den Parameter *IV_CHECK_SCOPE* entscheiden, ob die Prüfung lokal oder für eine Teilstruktur durchgeführt werden soll:

- Lokal: Bei Angabe der Konstante */BOBF/IF_FRW_C=>SC_SCOPE_LOCAL* wird nur die aktuelle Knotenebene überprüft.
- Teilstruktur: Bei Angabe von */BOBF/IF_FRW_C=>SC_SCOPE_SUBSTRUCTURE* werden auch alle Unterknoten überprüft.

Listing 5.27 zeigt die manuelle Ausführung einer Validierung mit zugehörigen Ermittlungen des Knotens *ROOT* des Geschäftsobjekts */BOBF/SALES_ORDER*.

```
DATA: lt_keys    TYPE /bobf/t_frw_key,
      ls_key     TYPE /bobf/s_frw_key,
      lo_message TYPE REF TO /bobf/if_frw_message.

"Beispielhafter technischer Schlüssel
ls_key-key = '0050568F72901ED79EA326CDC37C60FF'.
APPEND ls_key TO lt_keys.

lo_serv_mgr->check_and_determine(
  EXPORTING
    iv_node_key    = /bobf/if_demo_customer_c=>sc_node-root
    it_key         = lt_keys
    iv_check_scope = /bobf/if_frw_c=>sc_scope_local
  IMPORTING
    eo_message     = lo_message
).
```

Listing 5.27 Aufruf einer Ermittlung mit zugehörigen Prüfungen