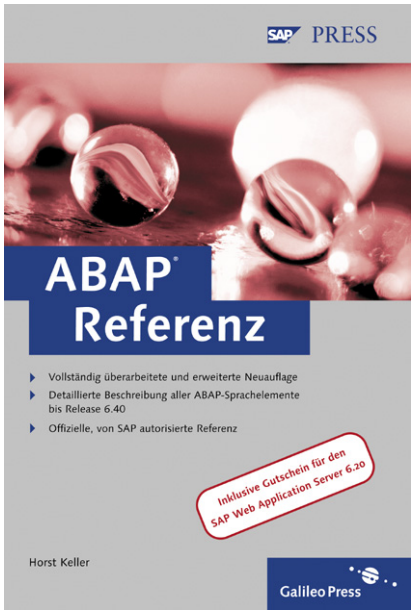
Horst Keller

ABAP-Referenz



Galileo Press

Inhaltsübersicht

Inhalt

42 Obsolete Anweisungen 1061

Vorwort zur zweiten Auflage

Knapp zwei Jahre nach dem Erscheinen der ersten Auflage der ABAP-Referenz liegt nun die zweite Auflage vor. In dieser Neuauflage wird zum einen der vollständige Sprachumfang von ABAP zum aktuellen Release 6.40 dargestellt und zum anderen wurde der bisherige Text korrigiert und in großen Teilen vollständig überarbeitet (siehe Abschnitt 1.8).

Als wichtigste Neuerungen zu Release 6.40 seien hier die Evolution der RTTI nach RTTS und die Shared Objects genannt. Die RTTS (Run Time Type Services) enthalten neben der bisherigen RTTI (Run Time Type Information) auch eine RTTC (Run Time Type Creation), die es erlaubt, zur Programmlaufzeit beliebige Datentypen zu erzeugen und zu verwenden. Damit wird endlich der Wunsch vieler ABAP-Entwickler erfüllt, Strukturen dynamisch erzeugen zu können. Das Beipiel in Abschnitt 9.2.6 zeigt, wie es geht. Shared Objects sind Objekte im Shared Memory, auf die alle Programme eines Applikationsservers zugreifen können. Die zugehörigen Sprachelemente und Begriffe sind in diesem Buch beschrieben. Weitere wichtige Neuerungen zu Release 6.40 sind die Einführung von Simple Transformations zur Serialisierung von ABAP-Daten nach XML und umgekehrt und das in die Sprache integrierte Testwerkzeug ABAP Unit.

Der SAP Web Application Server und mit ihm die Sprache ABAP ist seit Release 6.30 Bestandteil von SAP NetWeaver™. SAP NetWeaver ist eine offene Integrations- und Anwendungsplattform, die die vorhergehende mySAP-Technologie ablöst und erweitert. Neben der gewohnten ABAP-Umgebung enthält der SAP Web Application Server seit Release 6.30 auch eine Umgebung für die Anwendungsentwicklung in Java. Da diese aber nicht in direktem Zusammenhang mit den Sprachelementen von ABAP steht, wird in diesem Buch auch nicht auf den Java-Teil des Web Application Servers eingegangen.

Da Herr Joachim Jacobitz, mein Koautor der ersten Auflage, inzwischen andere Aufgaben innerhalb der SAP übernommen hat, bin ich jetzt als einziger Autor für den Inhalt dieses Buches verantwortlich. Trotzdem hat Herr Jacobitz mit großem Engagement alle Kapitel nochmals Korrektur gelesen und mir viele wertvolle Hinweise gegeben, für die ich ihm ganz herzlich danke.

Als alleiniger Autor dieses seitenstarken Werkes ist es mir ganz besonders wichtig, hervorzuheben, dass diese Arbeit ohne die Gruppe »NetWeaver

Development Tools ABAP«, deren hervorragende Arbeit ich hier vorstellen darf, nicht möglich gewesen wäre. Wie schon die Erstellung der ersten Auflage lebt auch die zweite Auflage von der konstruktiven Kritik und dem Korrekturlesen vieler Kollegen. Neben den im Vorwort zur ersten Auflage genannten Mitarbeitern haben sich um die zweite Auflage besonders verdient gemacht: Ulrich Brink, Christian Hansen, Ulrich Koch, Jürgen Lehmann, Mathias Müller, Helmut Prestel, Michael Redford, Christian Stork, Wolf Hagen Thümmel und Christoph Wedler. Allen anderen Kollegen, die hier nicht namentlich genannt sind, die ich aber auch mit Fragen quälte oder von denen der eine oder andere Tipp kam, sei ebenfalls von Herzen gedankt.

Mein ganz besonderer Dank gilt dieses Mal meinen studentischen Hilfskräften, Frau Agnieszka Chelminska, Frau Beata Kouchnir und Herrn Christian Pretzsch, die mir bei der Erstellung und Korrektur des Manuskripts und der Hand in Hand gehenden Überarbeitung der Online-Dokumentation äußerst engagiert und tatkräftig zur Seite standen.

Den beteiligten Mitarbeitern von Galileo Press und meinem Lektor, Herrn Florian Zimniak, danke ich dafür, dass sie die rasche Veröffentlichung einer Überarbeitung dieses mir sehr wichtigen Buches ermöglicht haben, und für die fortwährend nette Zusammenarbeit.

Abschließend danke ich wie immer meiner lieben Frau Ute für ihre Geduld und ihr Verständnis dafür, dass ich auch nach dem Erscheinen der ersten Auflage einfach nicht aufhörte, an diesem Buch zu arbeiten.

Walldorf, im Mai 2004

Horst Keller

Vorwort zur ersten Auflage

Gleich nachdem die Programmiersprache ABAP mit der Einführung von ABAP Objects zu Release 4.6 einen großen Schritt in Richtung zeitgemäße Programmierung vollzogen hatte, stellte sich für die Weiterentwicklung der Sprache bereits die nächste Herausforderung: Um den Anforderungen eines vom Internet bestimmten, internationalen Programmierumfelds gerecht zu werden, musste der SAP Web Application Server als Nachfolger der SAP-Basis und mit ihm die Programmiersprache ABAP unicode-fähig gemacht werden. Diese Anforderung wurde mit Release 6.10 erfüllt.

Mit der Implementierung von Release 6.10 bzw. 6.20 wird jedoch niemand gezwungen, seine Programme anzupassen, solange das SAP-System nicht auf Unicode umgestellt wird. Unabhängig davon, ob ein Programm in einem Unicode- oder einem Nicht-Unicode-System eingesetzt werden soll, bedeutet die Verwendung der im Zusammenhang mit Unicode eingeführten Neuerungen dennoch einen wichtigen Schritt in Richtung sicherer und fehlerfreier Programmierung.

Daher war vor allem die Einführung von Unicode ein wichtiger Beweggrund für dieses Buch. Die im System vorhandene ABAP-Schlüsselwortdokumentation wuchs mit der Sprache, indem Neues hinzugefügt und Altes unverändert gelassen wurde. Es gibt dort beispielsweise ein Teilgebiet »ABAP Objects« und seit Release 6.10 eben auch ein Teilgebiet »Unicode«, während die Anweisungen des klassischen Reporting nach wie vor so beschrieben sind, als wäre jedes ABAP-Programm ein Report, der mit einer logischen Datenbank verknüpft ist. Da diese Art der Dokumentation als Direkthilfe zwar vertretbar, für eine einheitliche Gesamtbeschreibung aber nicht mehr ausreichend ist, haben wir die Gelegenheit ergriffen, der Einführung in die SAP-Programmierung mit ABAP Objects eine ABAP-Referenz folgen zu lassen.

Das Ziel, das wir uns gesteckt haben, ist die vollständige Neufassung und Vereinheitlichung der ABAP-Dokumentation. Dies betrifft nicht nur die Gliederung und Zusammenfassung der einzelnen Anweisungen, sondern auch die inhaltliche Aufbereitung und stilistische Anpassung der Beschreibung. Speziell für die Darstellung der Syntaxdiagramme wurde daher eine neue Sichtweise in Form einer Pseudosyntax eingeführt, die jede Anweisung ausgehend von ihrer Grundform schrittweise in ihre einzelnen Bestandteile auflöst. Erstmals werden auch die Anforderungen an alle in einer ABAP-Anweisung vorkommenden Operanden vollständig und in

einer einheitlichen Form beschrieben. Unser Anspruch ist die umfassende Beschreibung aller im aktuellen Release 6.20 zur Verwendung freigegebenen ABAP-Sprachelemente. Dies beinhaltet auch eine detaillierte Beschreibung aller zwar als obsolet gekennzeichneten, aber nur in ABAP Objects verbotenen Sprachelemente. Die ABAP-Referenz soll den Leser in die Lage versetzen, jedes ABAP-Programm zu analysieren und neue Programme zu schreiben. Sie kann und will zwar kein Best-Practices-Buch sein, wir hoffen aber, dass Sinn und Zweck mancher Anweisung durch die genaue Beschreibung klarer wird, als es bisher der Fall war.

Es ist leichter gesagt als getan, eine Sprache mit etwa 500 wesentlichen Sprachelementen durchgehend auf gleich hohem Niveau und in gleichem Detaillierungsgrad zu beschreiben, zumal ABAP seit zwanzig Jahren ständig weiterentwickelt wird und aus Kompatibilitätsgründen kaum eine Entwicklung jemals wieder zurückgenommen wurde. Dies erklärt auch die wiederholten Verschiebungen des Erscheinungstermins dieses Buches und wir danken allen, die so lange gewartet haben. Als Entschädigung halten Sie jetzt aber ein Buch in Händen, das das aktuellste Release der Sprache ABAP beschreibt und lange gültig bleiben wird. Da wir jedoch trotz sorgfältiger Durchsicht aller Kapitel nicht ausschließen können, dass dieses Buch noch kleinere Unstimmigkeiten enthält, freuen wir uns auf alle diesbezüglichen Anregungen und Hinweise. Sie können sie auf den Internetseiten von SAP PRESS unter *www.sap-press.de* einreichen.

Ohne die mittelbare und unmittelbare Hilfe vieler Personen bei der Erstellung und Prüfung des Manuskripts wäre das Buch in der vorliegenden Form nicht zustande gekommen. An dieser Stelle gilt der Dank den Kollegen Masoud Aghadavoodi, Thomas Bareiss, Adrian Goerler, Christian Jendel, Gerd Kluger, Björn Mielenhausen, Andreas Simon Schmitt und Christoph Stöck. Erhardt Vortanz danken wir für seine Mitarbeit bei der Erstellung des Manuskripts. Dem Development Manager der Gruppe Business Programming Languages, Andreas Blumenthal, danken wir, dass er uns dieses Projekt überhaupt ermöglichte und weitgehend freie Hand in der Gestaltung ließ. Dem unermüdlichen Michael Demuth ist es zu verdanken, dass diesem Buch wieder zwei CDs mit einem SAP-System, dieses Mal in Form eines SAP Web Application Server 6.10, beiliegen. Dank gilt schließlich auch allen Mitarbeitern von Galileo Press – insbesondere Iris Warkus und Florian Zimniak – für die gute Zusammenarbeit bei der Korrektur des Manuskripts und dafür, dass sie die Hoffnung nie aufgaben, dass wir unser Manuskript irgendwann überhaupt noch einmal einreichen würden.

Horst Keller dankt ganz besonders seiner Frau Ute, die ihn in den letzten Monaten fast nur noch über dem aufgeklappten Laptop zu Gesicht bekam, für ihre Geduld und ihr Verständnis dafür, dass ein großer Anteil der ohnehin spärlichen gemeinsamen Freizeit schon wieder für ein Buchprojekt geopfert wurde.

Wir wünschen nun allen Leserinnen und Lesern viel Spaß beim Durchstöbern dieser ABAP-Referenz.

Walldorf, im Juli 2002
Horst Keller, Joachim Jacobitz

40 ABAP und XML

Durch Aufruf der Anweisung `CALL TRANSFORMATION` (siehe Abschnitt 40.4) lassen sich ABAP-Daten in das XML-Format umwandeln und umgekehrt. Dabei werden Transformationsprogramme aufgerufen, die entweder als XSL-Transformationen oder als Simple Transformations (seit Release 6.40) vorliegen können.

40.1 XSL-Transformationen

Eine XSL-Transformation ist ein in XSLT geschriebenes Programm im Repository (XSLT-Programm) zur Transformation von XML-Dokumenten. Beim Aufruf einer XSL-Transformation mit der Anweisung `CALL TRANSFORMATION` können auch ABAP-Daten direkt nach XML transformiert werden und umgekehrt. Hierfür wird implizit eine Serialisierung bzw. Deserialisierung vorgenommen.

Bei Transformationen, die ABAP-Daten als Quelle verwenden, werden die ABAP-Daten zuerst in eine kanonische XML-Repräsentation (asXML, siehe 40.2) serialisiert, die dann als eigentliche Quelle der XSL-Transformation dient. Bei Transformationen, die ABAP-Daten als Resultat erwarten, wird das Resultat der XSL-Transformation in die ABAP-Daten deserialisiert. Voraussetzung für die Deserialisierung ist, dass das Resultat eine kanonische XML-Repräsentation darstellt.

Für die Ausführung der Transformationen enthält die ABAP-Laufzeitumgebung seit Release 6.10 einen XSLT-Prozessor. Dieser unterstützt fast alle Anweisungen von XSLT und bietet Erweiterungen (so genannte Extension Instructions) wie etwa die Möglichkeit, ABAP-Methoden aus XSLT-Programmen aufzurufen.

40.1.1 XSL-Transformationen im Repository

Mit `CALL TRANSFORMATION` aufrufbare XSL-Transformationen müssen als XSLT-Programme im Repository vorhanden sein. Im Object Navigator der ABAP Workbench können XSLT-Programme über **Objekt bearbeiten · Weitere · Transformation** (bzw. **XSLT-Programm** vor Release 6.40 und **XSL-Transformation** vor Release 6.20) und Auswahl von XSLT-Programm angelegt und bearbeitet werden.

Unter der Bezeichnung ID wird von SAP die Identitäts-Transformation ausgeliefert. Bei einer Identitäts-Transformation von XML nach XML ist das Resultat eine Kopie des Quelldokuments. Bei einer Identitäts-Trans-

formation von ABAP nach XML ist das Resultat die kanonische XML-Repräsentation (asXML) der ABAP-Daten (explizite Serialisierung). Bei einer Identitäts-Transformation von XML nach ABAP wird eine kanonische XML-Repräsentation in ABAP-Daten transformiert (explizite Deserialisierung).

40.2 Kanonische XML-Repräsentation

Die kanonische XML-Repräsentation ist das Format eines XML-Dokuments, das bei einer Serialisierung von ABAP-Daten entsteht bzw. für eine Deserialisierung vorausgesetzt wird. Dieses Format wird auch als asXML (ABAP Serialization XML) bezeichnet. Die kanonische XML-Repräsentation unterstützt alle ABAP-Datentypen.

Das asXML-Format ist in folgenden Fällen von Bedeutung:

- Für selbst geschriebene XSL-Transformationen von ABAP-Daten in beliebige XML-Formate muss das asXML-Format des Ergebnisses der Serialisierung bekannt sein.
- Wenn externe XML-Dokumente erzeugt werden sollen, die in ABAP-Daten deserialisierbar sind, müssen diese im asXML-Format vorliegen.

Hinweis

Das asXML-Format von serialisierten ABAP-Daten oder Objekten kann mit der vordefinierten Identitäts-Transformation ID erzeugt und untersucht werden.

40.2.1 Generelles asXML-Format

Die folgenden Zeilen zeigen das generelle Format der kanonischen XML-Repräsentation[1] ohne den XML-Header, wobei Zeilenumbrüche und Einrückungen hier nur zur Verdeutlichung eingefügt wurden. Ein ausführliches Beispiel findet sich in Abschnitt 40.5.

```
<asx:abap version = "1.0"
          xmlns:asx = "http://www.sap.com/abapxml">
  <asx:values>
    <bn1>…</bn1>
    ...
    <bnn>...</bnn>
  </asx:values>
```

1 Das asXML-Format ist ein generelles Format, das nicht vollständig durch ein XML-Schema definiert werden kann. Der Grund hierfür ist, dass auf beliebige ABAP-Typen Bezug genommen wird.

```
  <asx:heap>
    ...
  </asx:heap>
</asx:abap>
```

Das Wurzelement eines asXML-Dokuments ist `abap` im Namensraum (XML-Namespace) `http://www.sap.com/abapxml`. Das optionale Attribut `version` hat zurzeit immer den Wert »1.0« und ist für zukünftige Erweiterungen von asXML vorgesehen. Das Wurzelelement `abap` muss ein Unterelement `values` des gleichen Namensraums enthalten. Die Unterelemente `bni` von `values` repräsentieren dabei die ABAP-Datenobjekte, die im Zusatz `source` der Anweisung `CALL TRANSFORMATION` als `e1`, `e2`, ... bzw. im Zusatz `result` als `f1`, `f2`, ... angegeben sind (siehe Abschnitt 40.4). Die Namen der Elemente `bn1`, `bn2`, ... sind die dort angegebenen Namen in Großbuchstaben. Die Textinhalte der Elemente `<bn1>...</bn1>` (bzw. `<bn1 ... />`), ... stellen die Inhalte benannter Datenobjekte außer Referenzvariablen dar. Referenzvariablen werden durch Elemente ohne Textinhalt, aber mit einem speziellen Attribut dargestellt (siehe Abschnitt 40.2.3.1). Das optionale Element `heap` enthält die Inhalte von referenzierten anonymen Datenobjekten und Objekten (siehe Abschnitte 40.2.3.2 und 40.2.3.3).

Die Bezeichner der Elemente `bn1`, `bn2`, ... enthalten mit Ausnahme der Sonderfälle in Tabelle 40.1 nur Großbuchstaben. Die in den Zusätzen `source` und `result` der Anweisung `CALL TRANSFORMATION` angegebenen Namen `bn1`, `bn2`, ... (bzw. Komponenten von Strukturen oder Objekten, siehe 40.2.2.2 und 40.2.3.3) können nur dann als (großbuchstabige) Namen für XML-Elemente verwendet werden, wenn sie ausschließlich die Zeichen »a« bis »z«, »A« bis »Z«, »0« bis »9« oder »_« enthalten, wobei das erste Zeichen ein Buchstabe oder »_« sein muss. Andere Zeichen werden gemäß Tabelle 40.1 ersetzt.

Zeichen im ABAP-Namen	Ersetzungszeichen im XML-Namen
ASCII-Zeichen ungleich »a« bis »z«, »A« bis »Z«, »0« bis »9« oder »_« und Zeichen »0« bis »9« als erstes Zeichen.	»_--hex(c)«, wobei hex(c) die zweistellige Hexadezimaldarstellung des ASCII-Codes des Zeichens c ist.
»/«	»_-«
»xml« als die ersten drei Zeichen in beliebiger Kombination von Groß- und Kleinschreibung	»x-ml« in entsprechender Kombination von Groß- und Kleinschreibung

Tabelle 40.1 Ersetzungsregeln für Zeichen in ABAP-Namen

40.2.2 asXML-Format für benannte Datenobjekte außer Referenzvariablen

Benannte Datenobjekte außer Referenzvariablen werden als die Textinhalte der Elemente <bn1>...</bn1>, ... dargestellt. Die Darstellung benannter Datenobjekte in <bn1>...</bn1>, ... richtet sich nach dem dazugehörigen ABAP-Datentyp.

40.2.2.1 Elementare Datentypen

Die asXML-Darstellung elementarer Datenobjekte mit eingebauten ABAP-Typen aus Tabelle 5.2 entspricht der kanonischen Repräsentation von XML-Schema-Datentypen (http://www.w3.org/TR/xmlschema-2/#built-in-datatypes, siehe Tabelle 40.2), wobei Datum und Zeit nach ISO-8601 und binäre Daten zur Basis 64 dargestellt werden.

ABAP-Typ	ABAP-Beispiel	XML-Schematyp	XML-Beispiel
c	" Hi"	string	" Hi"
d	"20020204"	date	"2002-02-04"
f	-3.140...0E+02	double	"-3.14E2"
i, b, s	-123	int, unsignedByte, short	"-123"
n	"001234"	string (pattern [0-9]+)	"001234"
p	-1.23	decimal	"-1.23"
string	" Hello "	string	" Hello "
t	"201501"	time	"20:15:01"
x	"ABCDEF"	base64Binary	"q83v"
xstring	"456789AB"	base64Binary	"RweJqw=="

Tabelle 40.2 Kanonische XML-Repräsentation (asXML) von eingebauten ABAP-Typen

40.2.2.2 Strukturen

Die Komponenten einer ABAP-Struktur werden in asXML als eine Folge von Unterelementen des Struktur-Elements dargestellt. Der Inhalt jedes Unterelements entspricht der kanonischen Repräsentation des Komponentenwertes. Der Name jedes Unterelements ist der Name der entsprechenden Komponente. Bei der Serialisierung werden die Unterelemente in der Reihenfolge der Komponenten in der Struktur dargestellt. Bei der Deserialisierung der asXML-Repräsentation einer Struktur spielt die Reihenfolge der Unterelemente keine Rolle und überzählige XML-Ele-

mente werden ignoriert. Komponenten der Struktur, für die es kein Unterelement gibt, bleiben initial.

40.2.2.3 Interne Tabellen

Die Zeilen einer internen Tabelle werden in asXML als eine Folge von Unterelementen des Tabellen-Elements dargestellt. Der Inhalt jedes Unterelements entspricht der kanonischen Repräsentation des Zeilenwertes. Der Name eines Unterelements ist irrelevant. Wenn die kanonische XML-Repräsentation durch eine Serialisierung erstellt wird, wird bei Bezug des Zeilentyps auf das ABAP Dictionary der dortige Name verwendet, ansonsten der Name `item`. Jede Tabellenart ist erlaubt. Bei der Serialisierung wird keine Information über die Tabellenart in das XML-Dokument übertragen. Wenn das Zielfeld einer XSL-Transformation eine sortierte Tabelle ist, werden die Zeilen bei der Deserialisierung entsprechend sortiert.

40.2.3 asXML-Format für Referenzvariablen und referenzierte Objekte

Anonyme Datenobjekte und Instanzen von Klassen (Objekte) werden in ABAP ausschließlich über Referenzen in Referenzvariablen adressiert. Das zugehörige asXML-Format besteht aus Unterelementen von `values` für benannte Referenzvariablen (siehe Abschnitt 40.2.3.1) und Unterelementen von `heap` (siehe Abschnitt 40.2.3.2 und 40.2.3.3) für die referenzierten Objekte. Der Bezug der Referenz-Elemente auf die Objekt-Elemente wird über einen XML-Referenz-Mechanismus hergestellt, bei dem ein referenziertes Objekt im gleichen XML-Dokument über einen Schlüssel identifiziert wird. Bei den Objekt-Elementen unter `heap` ist der dynamische Typ der Referenzvariablen zum Zeitpunkt der Serialisierung angegeben, um eine eindeutige Deserialisierung zu gewährleisten.

40.2.3.1 Benannte Referenzvariablen

Eine benannte Referenzvariable wird als einziges Attribut des zugehörigen Unterelements von `values` ohne Textinhalt dargestellt. Ein Attribut für eine Referenzvariable hat den Namen `href` und den Inhalt "#key" wobei `key` der eindeutige Schlüssel eines Objekts im Element `heap` ist. Ein Element einer initialen Referenz hat kein Attribut `href` und keinen sonstigen Inhalt. Bei der Serialisierung wird der Schlüssel `key` von der ABAP-Laufzeitumgebung gesetzt, für die Deserialisierung kann der Schlüssel beliebig sein.

40.2.3.2 Anonyme Datenobjekte

Die Darstellung eines anonymen Datenobjekts als Unterelement von `heap` erfolgt in der Form:

```
<asx:heap xmlns:nspace ...>
  <type id = "key" attri="...">...</type>
</asx:heap>
```

Der Wert eines solchen Unterelements wird in der asXML-Darstellung für benannte Datenobjekte (siehe Tabelle 40.2) bzw. für benannte Referenzvariablen (siehe Abschnitt 40.2.3.1) dargestellt. Falls das anonyme Datenobjekt selbst eine nicht-initiale Referenzvariable ist, verweist sie nach obigen Regeln auf ein weiteres Element von `heap`. Der Elementname `type` ist der als XML-Schematyp-Name aus dem Namensraum `nspace` (siehe Tabelle 40.3) angegebene Datentyp des Datenobjekts (bzw. der dynamische Typ der Referenzvariablen), wobei gegebenenfalls Attribute `attri` technische Eigenschaften des Typs angeben. Das obligatorische Attribut `id` enthält den eindeutigen Schlüssel `key` des Elements, über den es von der Darstellung der zugehörigen Referenzvariablen in `values` oder `heap` referenziert wird.

Der XML-Schematyp-Name wird nach folgender Hierarchie konstruiert:

1. Falls der Datentyp des Datenobjekts im ABAP Dictionary definiert ist, ist der XML-Schematyp-Name der Name des Datentyps aus dem ABAP Dictionary im zugehörigen Namensraum (siehe Tabelle 40.3).
2. Falls der Datentyp ein elementarer ABAP-Typ ist, ist der XML-Schematyp-Name in Tabelle 40.4 angegeben.
3. Falls der Datentyp als Komponente einer globalen oder lokalen Klasse bzw. eines Interfaces definiert ist, setzt sich der XML-Schematyp-Name aus dem Namen der Klasse bzw. des Interfaces und dem Namen des Datentyps zusammen, die durch einen Punkt (`.`) getrennt sind. Der zugehörige Namensraum (siehe Tabelle 40.3) zeigt an, ob es sich um eine globale oder lokale Klasse bzw. ein Interface handelt.
4. Falls der Datentyp ein mit `REF TO data` bzw. `REF TO object` definierter generischer Referenztyp ist, ist der XML-Schematyp-Name `refData` bzw. `refObject`. Beide haben den Namensraum `http://www.sap.com/abapxml/types/built-in`.
5. Ansonsten ist der XML-Schematyp-Name der Name eines mit `TYPES` definierten Datentyps, wobei der zugehörige Namensraum (siehe Tabelle 40.3) anzeigt, wo der Datentyp definiert ist.

Voraussetzung für die Konstruktion eines XML-Schematyp-Namens ist, dass der Datentyp des Datenobjekts einen statisch verwendbaren Namen hat. Falls der Datentyp nur als Eigenschaft eines Datenobjekts existiert und damit nur einen technischen Namen hat (vergleiche Abschnitt 6.1.4), kommt es bei der Serialisierung zu einer behandelbaren Ausnahme.

Tabelle 40.3 zeigt die Namensräume für die XML-Schematyp-Namen, wobei `types` in der ersten Spalte für `http://www.sap.com/abapxml/types` steht. Die Namensräume zeigen an, wo ein Datentyp definiert ist. In den Bezeichnern `prg`, `cpool`, `fpool`, `tpool`, `meth`, `func`, `form` und `class` werden Zeichen ungleich »a« bis »z«, »A« bis »Z«, »0« bis »9«, »_« oder »-« als »!hex(c)« dargestellt, wobei hex(c) die zweistellige Hexadezimaldarstellung des ASCII-Codes des Zeichens c ist.

Namensraum	Stelle der Definition
`types/dictionary`	ABAP Dictionary
`types/program/prg`	ABAP-Programm `prg`
`types/class-pool/cpool`	Class-Pool `cpool`
`types/type-pool/tpool`	Typgruppe `tpool`
`types/function-pool/fpool`	Funktionsgruppe `fpool`
`types/function/func`	Funktionsbaustein `func`
`types/program.form/prg/frm`	Unterprogramm `frm` in Programm `prg`
`types/function-pool.form/fpool/frm`	Unterprogramm `frm` in Funktionsgruppe `fpool`
`types/method/class/meth`	Methode `meth` einer globalen Klasse `class`
`types/program.method/prg/class/meth`	Methode `meth` einer lokalen Klasse `class` in Programm `prg`
`types/class-pool.method/cpool/class/meth`	Methode `meth` einer lokalen Klasse `class` in Class-Pool `cpool`
`types/function-pool.method/fpool/class/meth`	Methode `meth` einer lokalen Klasse `class` in Funktionsgruppe `fpool`

Tabelle 40.3 Namensräume für XML-Schematyp-Namen, wobei types in der ersten Spalte für http://www.sap.com/apapxml/types steht

Tabelle 40.4 zeigt die XML-Schematyp-Namen für elementare ABAP-Typen. Diese unterscheiden sich teilweise von den kanonischen XML-Schema-Datentypen aus Tabelle 40.2, da der Datentyp von anonymen Datenobjekten vollständig spezifiziert sein muss. Die Namensräume

nspace für die elementaren ABAP-Typen anonymer Datenobjekte sind entweder xsd="http://www.w3.org/2001/XMLSchema" für allgemeine Schematypen oder abap="http://www.sap.com/abapxml/types/built-in" für spezielle ABAP-Schematypen, bei denen teilweise technische Attribute angegeben werden müssen.

ABAP-Typ	XML-Schematyp-Name	Attribute
c	abap:string	maxLength
d	abap:date	-
f	xsd:double	-
i, b, s	xsd:int, xsd:unsigned Byte, xsd:short	-
n	abap:digits	maxLength
p	abap:decimal	totalDigits, fractionDigits
string	xsd:string	-
t	abap:time	-
x	abap:base64Binary	maxLength
xstring	xsd:base64Binary	-

Tabelle 40.4 XML-Schematyp-Namen für elementare ABAP-Typen

Das Attribut maxLength gibt die Länge für die ABAP-Typen generischer Länge an. Der XML-Schematyp abap:digits beschränkt den Wertebereich eines Elements auf Ziffern. Beim XML-Schematyp abap:decimal werden die Länge und die Nachkommastellen über die Attribute totalDigits und fractionDigits angegeben. Die Längenangabe totalDigits gibt die Anzahl der Stellen zwischen 1 und 31 an. In ABAP-Programmen wird die Länge für Datenobjekte vom Typ p in Bytes angegeben und die Anzahl der Dezimalstellen berechnet sich aus 2×len-1 (siehe Tabelle 5.2). Bei der Serialisierung ist der Wert von totalDigits daher immer ungerade. Bei der Deserialisierung wird ein gerader Wert von totalDigits implizit um eins erhöht.

Instanzen von Klassen

Die Darstellung der Instanz einer Klasse (Objekt) als Unterelement von heap erfolgt in der Form:

```
<asx:heap xmlns:nspace ...>
  <class id = "key">
    <part classVersion = "…">
```

```
        <name>...</name>
      </part>
      ...
    </class>
  </asx:heap>
```

Der Elementname `class` ist der XML-Schematyp-Name der Klasse des Objekts (bzw. der dynamische Typ der Referenzvariablen) aus dem Namensraum `nspace` (siehe Tabelle 40.5) in Großbuchstaben. Das obligatorische Attribut `id` enthält den eindeutigen Schlüssel `key` des Elements, über den es von der Darstellung der zugehörigen Referenzvariablen in `values` referenziert wird. Die Unterelemente `<part>...</part>` enthalten die Werte der Instanzattribute einzelner Objektteile als Unterelemente `<name>...</name>`. Die einzelnen Objektteile werden durch die serialisierbaren Klassen der aktuellen Vererbungshierarchie definiert (siehe unten).

Der Namensraum des Klassennamens gibt an, wo die Klasse definiert ist. Tabelle 40.5 zeigt die möglichen Namensräume, wobei `classes` in der ersten Spalte für `http://www.sap.com/apapxml/classes` steht. Für die Bezeichner `prg`, `cpool` und `fpool` gilt die gleiche Ersetzungsregel wie für die Namensräume in Tabelle 40.3.

Namensraum	Stelle der Definition
`classes/global`	Klassenbibliothek
`classes/program/prg`	Programm `prg`
`classes/class-pool/cpool`	Class-Pool `cpool`
`classes/function-pool/fpool`	Funktionsgruppe `fpool`

Tabelle 40.5 Namensräume für Klassennamen, wobei classes in der ersten Spalte für http://www.sap.com/apapxml/classes steht

Die serialisierbaren Werte einer Instanz einer Klasse (Instanzattribute oder Ausgabeparameter einer speziellen Methode, siehe unten) werden in der asXML-Darstellung für benannte Datenobjekte (siehe Tabelle 40.2) bzw. für Referenzvariablen als Inhalt bzw. Attribut von `<name>...</name>` dargestellt, wobei `name` der Name eines Instanzattributs oder Ausgabeparameters in Großbuchstaben ist. Falls es sich um ein Interface-Attribut handelt, steht vor dem Namen der durch einen Punkt (`.`) abgetrennte Namen des Interfaces, um es von einem gleichnamigen Klassenattribut zu unterscheiden. Für die Bezeichner gelten die Ersetzungsregeln aus Tabelle 40.1.

Die serialisierbaren Werte einer Instanz einer Klasse werden durch Implementierung des Systeminterfaces IF_SERIALIZABLE_OBJECT in der Klasse festgelegt (siehe Abschnitt B.2). Wenn die Klasse das Interface IF_SERIALIZABLE_OBJECT nicht implementiert, enthält das Element `class` keine Unterelemente. Standardmäßig werden alle Instanzattribute einer Klasse, in der das Interface IF_SERIALIZABLE_OBJECT implementiert ist, serialisiert und in diese deserialisiert. Diese Verhalten ist durch die Deklaration spezieller Hilfsmethoden änderbar (siehe unten). Statische Attribute werden weder bei der Serialisierung noch bei der Deserialisierung berücksichtigt (mit Ausnahme der speziellen Konstanten SERIALIZABLE_CLASS_VERSION, siehe unten).

Standardverhalten

Wenn die Klasse das Interface IF_SERIALIZABLE_OBJECT implementiert, enthält das Element `<class>...</class>` mindestens ein Unterelement `<part>...</part>`. Diese Unterelemente entsprechen einzelnen serialisierbaren Objektteilen und enthalten die Darstellungen der Instanzattribute des jeweiligen Objektteils im asXML-Format. Ein Objektteil wird durch die Klasse bestimmt, in der Instanzattribute deklariert sind bzw. in der ein Interface eingebunden ist, das Instanzattribute enthält. Eine serialisierbare Klasse enthält einen Objektteil für sich selbst sowie Objektteile für alle Oberklassen im aktuellen Pfad des Vererbungsbaums bis einschließlich der Klasse, die das Interface IF_SERIALIZABLE_OBJECT implementiert. Der Name `part` ist der Name der jeweiligen Klasse. Falls es sich um eine lokale Klasse handelt, steht vor dem Namen das durch einen Punkt (.) abgetrennte Präfix `local`, um diese von einer gleichnamigen globalen Klasse zu unterscheiden. Objektteile von Oberklassen, in denen das Interface IF_SERIALIZABLE_OBJECT nicht implementiert ist, sind nicht serialisierbar und haben kein entsprechendes Unterelement `part`. D.h., eine Klasse, in der das Interface IF_SERIALIZABLE_OBJECT nicht implementiert ist (weder in der Klasse selbst noch in einer Oberklasse), erzeugt bei der Serialisierung ein leeres XML-Element `class`.

Bei der Serialisierung werden die XML-Elemente `part` der Objektteile von den Oberklassen zu den Unterklassen hin erzeugt und standardmäßig die XML-Elemente der Instanzattribute in der Reihenfolge angelegt, wie sie in der Klasse deklariert sind.

Bei der Deserialisierung wird ein Objekt der betreffenden Klasse erzeugt, wobei der Instanzkonstruktor nicht ausgeführt wird. Alle Instanzattribute haben nach der Objekterzeugung ihren Initialwert bzw. den mit dem Zusatz `VALUE` der Anweisung `DATA` angegebenen Startwert. Standard-

mäßig werden die Instanzattribute mit den Werten der zugehörigen XML-Elemente versorgt, wobei die Reihenfolge der Objektteile und der Attribute keine Rolle spielt. Instanzattribute ohne zugehöriges XML-Element behalten ihren Wert. Überflüssige XML-Elemente werden ignoriert, solange sie keinem Namensraum angehören, ansonsten erzeugen sie eine behandelbare Ausnahme. Bei der Deserialisierung eines Elements ohne Unterelemente `part` wird kein Objekt erzeugt, sondern die Zielreferenzvariable initialisiert.

Wenn eine Klasse das Interface IF_SERIALIZABLE_OBJECT implementiert, kann in jedem Objektteil, d.h. in jeder beteiligten Klasse des Vererbungsbaums, die private Konstante SERIALIZABLE_CLASS_VERSION vom Typ `i` deklariert werden. Bei der Serialisierung wird der Wert der Konstanten dem Attribut `classVersion` des XML-Elements `part` zugewiesen. Standardmäßig kommt es bei der Deserialisierung zu einer behandelbaren Ausnahme, wenn der Wert des Attributs nicht mit dem Wert der Konstanten in der angegebenen Klasse übereinstimmt. Ein Objektteil ist nur deserialisierbar, wenn die Werte übereinstimmen oder weder das Attribut noch die Konstante vorhanden sind. Dieses Verhalten kann durch Deklaration spezieller Hilfsmethoden geändert werden.

Angepasstes Verhalten

Standardmäßig werden alle Instanzattribute eines Objektteils unabhängig von ihrer Sichtbarkeit serialisiert und die Version der Klasse wird überprüft. Um dieses Verhalten zu ändern, können für jeden Objektteil die Instanzmethoden SERIALIZE_HELPER und DESERIALIZE_HELPER in der jeweiligen Klasse deklariert und implementiert werden. Diese Methoden können nur als private Instanzmethoden in Klassen deklariert werden, die das Interface IF_SERIALIZABLE_OBJECT implementieren. Die Deklaration einer der Methoden bedingt die Deklaration der anderen und die Schnittstelle wird von der Syntaxprüfung wie folgt vorgeschrieben:

- Die Methode SERIALIZE_HELPER darf nur Ausgabe- und die Methode DESERIALIZE_HELPER nur Eingabeparameter haben, deren Typisierung nicht generisch ist.
- Zu jedem Ausgabeparameter der Methode SERIALIZE_HELPER muss es einen gleichnamigen Eingabeparameter der Methode DESERIALIZE_HELPER gleicher Typisierung geben. Zusätzliche Eingabeparameter der Methode DESERIALIZE_HELPER müssen optional sein.

- Die Methode SERIALIZE_HELPER darf keinen Ausgabeparameter des Namens SERIALIZABLE_CLASS_VERSION, die Methode DESERIALIZE_HELPER darf einen solchen optionalen Eingabeparameter vom Typ `i` haben. Dieser wird bei der Deserialisierung mit dem Wert des Attributs `classVersion` des Elements `part` versorgt, wobei die Standardüberprüfung der Version (siehe oben) umgangen wird.

Wenn in einem Objektteil die Methoden SERIALIZE_HELPER und DESERIALIZE_HELPER deklariert sind, werden nicht die Instanzattribute des Objektteils serialisiert und deserialisiert. Stattdessen wird bei der Serialisierung die Methode SERIALIZE_HELPER ausgeführt und die Werte aller Ausgabeparameter werden in der angegebenen Reihenfolge im asXML-Format als Unterelemente in das entsprechende Element `part` geschrieben. Dabei ist der Name eines Unterelements der Name des jeweiligen Ausgabeparameters in Großbuchstaben. Bei der Deserialisierung wird die Methode DESERIALIZE_HELPER aufgerufen, wobei die Werte der Unterelemente des entsprechenden Elements `part` den gleichnamigen Eingabeparametern der Methode übergeben werden. Dabei spielt die Reihenfolge keine Rolle und überflüssige XML-Elemente werden ignoriert.

40.3 Simple Transformations

Simple Transformations (ST) ist eine SAP-eigene Programmiersprache, um Transformationen zwischen ABAP-Daten und XML-Formaten zu beschreiben. ST ist auf die beiden für die Datenintegration wesentlichen Modi der Serialisierung (ABAP nach XML) und Deserialisierung (XML nach ABAP) von ABAP-Daten beschränkt. Transformationen von ABAP nach ABAP und XML nach XML wie beim allgemeineren XSLT sind in ST nicht möglich.

Im Gegensatz zu XSLT liegen bei ST-Programmen die Schwerpunkte wie folgt:

- ST-Programme sind deklarativ und somit leicht lesbar.
- ST-Programme haben nur seriellen Zugriff auf die XML-Daten und sind deshalb auch für große Datenmengen sehr effizient.
- ST-Programme beschreiben gleichzeitig Serialisierung und Deserialisierung, d.h., in der Regel können mit ST in XML serialisierte ABAP-Daten mit demselben ST-Programm auch wieder deserialisiert werden.

Mit CALL TRANSFORMATION aufrufbare Simple Transformations müssen im Repository vorhanden sein. Im Object Navigator der ABAP Workbench können ST-Programme über **Objekt bearbeiten · Weitere · Transformation** und Auswahl von **Simple Transformation** angelegt und bearbeitet werden.

Eine genaue Beschreibung der Sprache ST geht über den Rahmen der vorliegenden ABAP-Referenz hinaus. Hier sei auf die entsprechende Online-Hilfe verwiesen. Ein einführendes Beispiel findet sich in Abschnitt 40.6.

40.4 Aufruf einer XSL- oder ST-Transformation

CALL TRANSFORMATION

AB RELEASE 6.10

Aufruf eines XSLT- oder ST-Programms.

Syntax

```
CALL TRANSFORMATION transformation
                    [PARAMETERS parameters]
                    [OBJECTS objects]
                    [OPTIONS options]
                    SOURCE source
                    RESULT result.
```

Diese Anweisung ruft die angegebene XSL-Transformation (XSLT) oder eine Simple Transformation (ST, ab Release 6.40) auf. Die Quelle der Transformation wird hinter SOURCE angegeben und das Ergebnis wie hinter RESULT angegeben abgelegt. Mit PARAMETERS und OBJECTS können Parameter an die Transformation übergeben werden. Mögliche Transformationsarten sind:

- von XML nach XML (nur bei XSLT)
- von XML nach ABAP (bei XSLT und ST)
- von ABAP nach XML (bei XSLT und ST)
- von ABAP nach ABAP (nur bei XSLT)

Die letzten beiden Arten sind erst ab Release 6.20 verfügbar.

Behandelbare Ausnahmen

Die gemeinsame Oberklasse aller Ausnahemklassen für CALL TRANSFORMATION ist CX_TRANSFORMATION_ERROR (seit Release 6.40). Die zugehörigen Laufzeitfehler sind nicht abfangbar.

Ausnahmen bei XSL-Transformationen
Die Ausnahmeklassen für XSL-Transformationen sind alle Unterklassen von CX_XSLT_EXCEPTION.

Tritt beim Parsen eines XML-Dokuments ein Fehler auf oder wird ein anderer Fehler vom XSLT-Prozessor gemeldet, wird eine durch die Klasse CX_XSLT_RUNTIME_ERROR definierte Ausnahme ausgelöst. Führt der Aufruf einer ABAP-Methode aus dem XSLT-Programm zu einem Fehler, wird eine durch die Klasse CX_XSLT_ABAP_CALL_ERROR definierte Ausnahme ausgelöst, wobei das Attribut PREVIOUS auf das Ausnahmeobjekt des ursprünglichen Fehlers zeigt.

Hat ein XML-Dokument bei der Deserialisierung nicht das asXML-Format, wird eine durch die Klasse CX_XSLT_FORMAT_ERROR definierte Ausnahme ausgelöst, wobei das Attribut TREE_POSITION die Fehlerposition enthält. Treten bei der Serialisisierung oder Deserialisierung ungültige Werte oder Datentypen auf, werden durch die Klassen CX_XSLT_SERIALIZATION_ERROR bzw. CX_XSLT_DESERIALIZATION_ERROR definierte Ausnahmen ausgelöst, wobei das Attribut PREVIOUS gegebenenfalls auf das Ausnahmeobjekt des ursprünglichen Fehlers zeigt und das Attribut TREE_POSITION die Fehlerposition bei der Deserialisierung enthält.

Ausnahmen bei Simple Transformations
Die Ausnahmeklassen für Simple Transformations sind alle Unterklassen von CX_ST_ERROR.

40.4.1 Angabe der Transformation

Syntax von *transformation*

```
... trans | (name) ...
```

Der Name der Transformation kann entweder direkt als `trans` oder als Inhalt eines eingeklammerten zeichenartigen Datenobjekts `name` angegeben werden. Die angegebene Transformation muss als XSLT-Programm oder als Simple Transformation im Repository vorhanden sein.

40.4.2 Transformationsquelle

Syntax von *source*

```
... { XML sxml }
  | {{bn1 = e1 bn2 = e2 ...}|(stab)} ...
```

40.4.2.1 Transformation eines XML-Dokuments

Durch die Angabe von `XML sxml` wird das in `sxml` enthaltene XML-Dokument transformiert, wobei `sxml` in folgenden Formen vorliegen kann:

- als Datenobjekt vom Typ `string` und `xstring` oder als Standardtabelle mit flachem zeichenartigem Zeilentyp
- als Interface-Referenzvariable vom Typ IF_IXML_ISTREAM, die auf einen iXML-Input-Stream zeigt (nur bei XSLT)
- als Interface-Referenzvariable vom Typ IF_IXML_NODE, die auf ein iXML-Nodeset zeigt (nur bei XSLT)
- als Klassen-Referenzvariable vom Typ CL_FX_READER, die auf einen XML-Reader zeigt (nur bei ST)

Hinweis
Die Interfaces IF_IXML_ISTREAM und IF_IXML_NODE sind Komponenten der Pakete »Stream« und »DOM« der von SAP ausgelieferten iXML Library.

40.4.2.2 Transformation von ABAP-Daten

Mir `bn1 = e1, bn2 = e2, ...` oder `(stab)` werden die zu transformierenden ABAP-Daten `e1, e2, ...` angegeben.

- Bei Aufruf eines XSLT-Programms werden die ABAP-Daten in die kanonische XML-Repräsentation serialisiert, die dann als Quelle der XSL-Transformation verwendet wird. Mit `bn1, bn2, ...` werden die Namen der XML-Elemente angegeben, die die ABAP-Datenobjekte in der kanonischen XML-Darstellung repräsentieren sollen.
- Beim Aufruf einer Simple Transformation wird in der Transformation über die Namen `bn1, bn2, ...` lesend auf die ABAP-Daten zugegriffen.

Statt über eine statische Parameterliste können die Datenobjekte auch dynamisch als Wertepaare in den Spalten einer internen Tabelle `stab` übergeben werden, die den Typ ABAP_TRANS_SRCBIND_TAB aus der Typgruppe ABAP hat.

Folgende Datenobjekte sind nicht serialisierbar und führen zu einer behandelbaren Ausnahme:

- Datenobjekte vom Typ `n`, deren aktueller Inhalt nicht ausschließlich aus Ziffern besteht.
- Datenobjekte vom Typ `p`, deren aktueller Inhalt keine gültige gepackte Zahl darstellt.

- Datenobjekte vom Typ `d` und `t`, deren aktueller Inhalt führende oder schließende Leerzeichen und gleichzeitig die in der Darstellung nach ISO-8601 verwendeten Trennzeichen (»-« bzw. »:«) enthält.
- Datenreferenzvariablen, die auf Datenobjekte zeigen, deren Datentyp nur einen technischen Namen hat (siehe Abschnitt 40.2.3).

Datenreferenzvariablen, die auf Datenobjekte zeigen, die nicht mit `CREATE DATA` erzeugt wurden, werden bei der Serialisierung wie initiale Referenzvariablen behandelt.

40.4.3 Transformationsergebnis

Syntax von *result*

```
... { XML rxml }
  | {{bn1 = f1 bn2 = f2 ...}|(rtab)} ...
```

40.4.3.1 Transformation in ein XML-Dokument

Durch die Angabe von `XML rxml` wird in ein XML-Dokument transformiert und dieses nach `rxml` gestellt, wobei `rxml` in folgenden Formen vorliegen kann:

- als Datenobjekt vom Typ `string` und `xstring` oder als Standardtabelle mit flachem zeichenartigem Zeilentyp
- als Interface-Referenzvariable vom Typ IF_IXML_OSTREAM, die auf einen iXML-Output-Stream zeigt (nur bei XSLT)
- als Interface-Referenzvariable vom Typ IF_IXML_DOCUMENT, die auf ein iXML-Dokument zeigt (nur bei XSLT)
- als Klassen-Referenzvariable vom Typ CL_FX_WRITER, die auf einen XML-Writer zeigt (nur bei ST)

Hinweise

- Die Interfaces IF_IXML_OSTREAM und IF_IXML_DOCUMENT sind Komponenten der Pakete »Stream« und »DOM« der von SAP ausgelieferten iXML Library.
- Wenn der Datentyp `xstring` für `rxml` verwendet wird, wird das Ergebnis in der Zeichendarstellung UTF-8 abgelegt. Dies ist vorteilhaft, wenn das resultierende XML-Dokument in einer Datei gespeichert werden soll (siehe Kapitel 32).

40.4.3.2 Transformation in ABAP-Daten

Mit `bn1 = f1, bn2 = f2, ...` oder `(rtab)` werden die ABAP-Zielfelder `f1`, `f2`, ... angegeben, in die die XML-Daten transformiert werden sollen.

- Bei Aufruf eines XSLT-Programms wird das Resultat der XSL-Transformation in ABAP-Datenobjekte deserialisiert, falls es eine kanonische XML-Repräsentation darstellt. Mit `bn1`, `bn2`, ... werden die Namen der XML-Elemente angegeben, die die ABAP-Datenobjekte in der kanonischen XML-Darstellung repräsentieren, während mit `f1`, `f2`, ... vom Datentyp her passende ABAP-Datenobjekte angegeben werden, in die diese deserialisiert werden sollen.
- Beim Aufruf einer Simple Transformation wird in der Transformation über die Namen `bn1`, `bn2`, ... schreibend auf die ABAP-Daten zugegriffen

Statt über eine statische Parameterliste können die Datenobjekte auch dynamisch als Wertepaare an die Spalten der internen Tabelle `rtab` übergeben werden, die den Typ ABAP_TRANS_RESBIND_TAB aus der Typgruppe ABAP hat.

Ein XML-Element muss in das zugehörige ABAP-Datenobjekt konvertierbar sein, wobei abweichend zu den Konvertierungsregeln in Anhang A folgende Verschärfungen gelten:

- Es darf keinen Datenverlust durch Deserialisierung in zu kurze Datenobjekte vom Datentyp `c` oder `n` geben, außer es sind nur führende und schließende Leerzeichen beim Datentyp `c` und führende Nullen beim Datentyp `n` betroffen.
- Es darf keinen Datenverlust durch Deserialisierung in ein Datenobjekt vom Datentyp `p` mit zu wenig Dezimalstellen geben.
- Es darf keinen Datenverlust durch Deserialisierung in ein zu kurzes Datenobjekt vom Datentyp `x` geben.
- Strukturen können nicht in elementare Datenobjekte konvertiert werden.

Wenn ein XML-Element nicht in das ABAP-Datenobjekt konvertierbar ist, wird eine behandelbare Ausnahme ausgelöst.

Bei der Deserialisierung in eine Referenzvariable muss diese gleich oder allgemeiner als der dynamische Typ des im XML-Dokument abgelegten Objekts sein. Die zugehörigen ABAP-Datenobjekte bzw. Instanzen einer Klasse werden während der Deserialisierung erzeugt.

40.4.4 Parameter für eine XSL-Transformation

Syntax von *parameters*

```
... {p1 = e1 p2 = e2 ...}|(ptab) ...
```

Mit diesem Zusatz können ABAP-Datenobjekte `e1`, `e2`, ... als Parameter `p1`, `p2`, ... an eine XSL-Transformation übergeben werden. In Release 6.10 müssen die Datenobjekte `e1`, `e2`, ... zeichenartig sein, ab Release 6.20 sind alle elementaren Datenobjekte und Objektreferenzen erlaubt.

Statt über eine statische Parameterliste können die Parameter auch dynamisch als Wertepaare in den Spalten der internen Tabelle `ptab` übergeben werden, die den Typ ABAP_TRANS_PARMBIND_TAB aus der Typgruppe ABAP hat.

Die angegebenen Parameter müssen in der XSL-Transformation wie folgt als Eingabeparameter definiert sein:

```
<xsl:param name="..." type="..."/>
```

Das Attribut `name` muss den Namen des Parameters in Großbuchstaben darstellen. Für das optionale Attribut `type` kann einer der Typbezeichner `string`, `number`, `boolean`, `xstring`, `nodeset` oder `object(...)` angegeben sein, wobei in den Klammern hinter `object` der Name einer globalen ABAP-Klasse anzugeben ist.

Wenn in der XSL-Transformation kein Typ angegeben ist, werden die Datentypen elementarer Parameter gemäß Tabelle 40.6 auf XSL-Typen abgebildet.

ABAP-Datentyp	XSL-Parametertyp
`c`, `d`, `n`, `string`	`string`
`i`, `s`, `b`, `f`, `p`	`number`
`x`, `xstring`	`string`, wobei der Inhalt zur Basis 64 dargestellt wird

Tabelle 40.6 Abbildung von ABAP-Datentypen auf die Typen von XSL-Parametern

Falls in der XSL-Transformation die in Tabelle 40.6 gezeigten XSL-Typen explizit angegeben sind, müssen passende elementare ABAP-Parameter angegeben werden, die in den Typ konvertierbar sind:

- Der XSL-Typ `boolean` erwartet ABAP-Parameter vom Typ `c` der Länge 1, wobei ein Leerzeichen als »falsch« und ein anderes Zeichen als »wahr« interpretiert wird.
- Der XSL-Typ `xstring` erwartet ABAP-Parameter vom Typ `x` oder `xstring` und der Inhalt wird hexadezimal dargestellt.
- Die XSL-Typen `nodeset` und `object` erwarten eine Objektreferenzvariable, die auf eine Instanz einer Klasse zeigt, wobei der Typ `nodeset` passende Objekteigenschaften erwartet.

Falls ein Parameter nicht zum XSL-Typ passt, kommt es zu einer unbehandelbaren Ausnahme. Falls ein in der XSL-Transformation definierter Parameter nicht übergeben wird, wird er in der Transformation auf einen Standardwert gesetzt. Ein angegebener Parameter, der nicht in der XSL-Transformation definiert ist, wird ignoriert.

Hinweis
Die XSL-Typen `string`, `number`, `boolean` und `nodeset` sind XSL-Standardtypen, während `xstring` und `object` spezielle SAP-Erweiterungen darstellen. Der Typ `xstring` ermöglicht es, Byteketten hexadezimal statt zur Basis 64 darzustellen. Der Typ `object` erlaubt den Aufruf von ABAP-Methoden aus dem XSL-Programm.

40.4.5 Externe Objekte an eine XSL-Transformation übergeben

Syntax von *objects*

```
... {o1 = e1 o2 = e2 ...}|(otab) ...
```

Mit diesem Zusatz können Objektreferenzen `e1`, `e2`, ... als externe Objekte `o1`, `o2`, ... an eine XSL-Transformation übergeben werden, um dort deren Methoden aufzurufen.

Statt über eine statische Parameterliste können die Objekte auch dynamisch als Wertepaare in den Spalten der internen Tabelle `otab` übergeben werden, die den Typ ABAP_TRANS_OBJBIND_TAB aus der Typgruppe ABAP hat.

Hinweis
Der Zusatz `OBJECTS` ist ab Release 6.20 obsolet. Ab Release 6.20 werden externe Objekte wie Parameter behandelt und Objektreferenzen sollen entsprechend mit dem Zusatz `PARAMETERS` (siehe Abschnitt 40.4.4) übergeben werden.

40.4.6 Steuerung der Transformation

AB RELEASE 6.40

Syntax von *options*

```
... a1 = e1 a2 = e2 ...
```

Dieser Zusatz erlaubt die Angabe von Werten `e1`, `e2`, ... zu weiteren Steuerungsoptionen `a1`, `a2`, ... der Transformation. Die Werte `e1`, `e2`, ... müssen vom Typ `c` oder `string` sein.

Für `a1`, `a2`, ... können angegeben werden:

- `XML_HEADER` zur Steuerung der Ausgabe des XML-Headers, falls nach XML transformiert und in ein Datenobjekt vom Typ `c`, `string` oder eine interne Tabelle geschrieben wird.

Mögliche Werte	Bedeutung
`no`	Es wird kein XML-Header ausgegeben.
`without_encoding`	Es wird ein XML-Header ohne Angabe des Encodings ausgegeben.
`full`	Standardeinstellung, es wird ein XML-Header mit Angabe des Encodings ausgegeben.

- `DATA_REFS` zur Steuerung der Ausgabe von Datenreferenzen, falls von ABAP nach XML transformiert wird.

Mögliche Werte	Bedeutung
`no`	Standard bei ST, es werden keine Datenreferenzen ausgegeben.
`heap`	Standard bei XSLT und nur dort möglich, referenzierte Daten werden als Unterelemente des asXML-Elements `<asx:heap>` ausgegeben.
`embedded`	Referenzierte Daten werden mit der Referenz ausgegeben.

- `INITIAL_COMPONENTS` zur Steuerung der Ausgabe von initialen Strukturkomponenten, falls von ABAP nach XML transformiert wird.

Mögliche Werte	Bedeutung
`include`	Standardeinstellung, initiale Komponenten von Strukturen werden ausgegeben.
`suppress`	Initiale Komponenten von Strukturen werden nicht ausgegeben.

40.5 Beispiel für eine XSL-Transformation

Serialisierung von Datenobjekten in einen String `xmlstr` über die identische Transformation ID. Es werden ein Datumsfeld `date`, ein Zeitfeld `time` und eine Datenreferenzvariable `dref1` serialisiert. Die Datenreferenzvariable zeigt auf eine anonyme Objektreferenzvariable, die wiederum auf ein Objekt der Klasse `c2` zeigt. Solcherart serialisierte Objekte können persistent gespeichert werden, beispielsweise in einem Daten-Cluster. Nach dem Einlesen aus der Ablage werden die Objekte in andere Datenobjekte deserialisiert. Nach der Deserialisierung zeigt `dref2` auf eine andere anonyme Referenzvariable wie `dref1`. Dieses anonyme Datenobjekt wie auch die Instanz der Klasse `c2`, auf die es zeigt, werden bei der Deserialisierung erzeugt.

```
PROGRAM xmltst.

CLASS c1 DEFINITION.
  PUBLIC SECTION.
    INTERFACES if_serializable_object.
  PROTECTED SECTION.
    DATA carriers TYPE TABLE OF scarr.
ENDCLASS.

CLASS c2 DEFINITION INHERITING FROM c1.
  PUBLIC SECTION.
    METHODS constructor.
  PRIVATE SECTION.
    DATA lines TYPE i.
    METHODS: serialize_helper
               EXPORTING count TYPE i,
             deserialize_helper
               IMPORTING count TYPE i.
ENDCLASS.

CLASS c2 IMPLEMENTATION.
  METHOD constructor.
    super->constructor( ).
    SELECT * UP TO 2 ROWS
           FROM  scarr
           INTO  TABLE carriers.
  ENDMETHOD.
  METHOD serialize_helper.
    count = LINES( carriers ).
  ENDMETHOD.
  METHOD deserialize_helper.
```

```
    lines = count.
  ENDMETHOD.
ENDCLASS.

DATA: oref   TYPE REF TO object,
      dref1  LIKE REF TO oref,
      xmlstr TYPE string,
      date   TYPE d,
      time   TYPE t,
      dref2  LIKE dref1.

...

CREATE DATA dref1 LIKE oref.
CREATE OBJECT dref1->* TYPE c2.

CALL TRANSFORMATION id
                    SOURCE xmldat = sy-datum
                           xmltim = sy-uzeit
                           ref    = dref1
                    RESULT XML xmlstr.

EXPORT obj = xmlstr TO DATABASE indx(hk)
                    ID 'OBJECT'.

...

IMPORT obj = xmlstr FROM DATABASE indx(hk) ID 'OBJECT'.

CALL TRANSFORMATION id
                    SOURCE XML xmlstr
                    RESULT xmldat = date
                           xmltim = time
                           ref    = dref2.
```

Das bei der Serialisierung erzeugte XML-Dokument hat unten stehenden Inhalt, wobei hier Zeilenumbrüche und Einrückungen eingefügt wurden. Das Element `values` enthält die asXML-Darstellungen der drei übergebenen Datenobjekte (siehe Abschnitt 40.2). In den Bezeichnern `X-MLDAT` und `X-MLTIM` wurde »xml« gemäß Tabelle 40.1 ersetzt. Das Attribut `href` des Elements `REF` verweist über den Schlüssel »d1« auf die Darstellung des zugehörigen anonymen Datenobjekts im Element `heap`. Diese verweist über den Schlüssel »o3« auf die Darstellung der Instanz der Klasse `c2`, die sich ebenfalls im Element `heap` befindet. Diese Darstellung ist in die Objektteile für die Klassen `c1` und `c2` unterteilt. Im Objektteil für `c1` befindet sich die Darstellung der zweizeiligen strukturierten internen Tabelle `carriers`. Im Objektteil für `c2` befindet sich die Darstellung für den Ausgabeparameter `count` der Methode SERIALIZE_HELPER.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<asx:abap xmlns:asx="http://www.sap.com/
abapxml" version="1.0">
  <asx:values>
    <X-MLDAT>2003-04-15</X-MLDAT>
    <X-MLTIM>14:57:53</X-MLTIM>
    <REF href="#d1" />
  </asx:values>
  <asx:heap
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:abap="http://www.sap.com/abapxml/types/built-in"
    xmlns:cls="http://www.sap.com/abapxml/classes/global"
    xmlns:dic="http://www.sap.com/abapxml/types/dictionary">
    <abap:refObject href="#o3" id="d1" />
    <prg:C2
    xmlns:prg="http://www.sap.com/abapxml/classes/program/
XMLTST"
        id="o3">
      <local.C1>
        <CARRIERS>
          <SCARR>
            <MANDT>000</MANDT>
            <CARRID>AA</CARRID>
            <CARRNAME>American Airlines</CARRNAME>
            <CURRCODE>USD</CURRCODE>
            <URL>http://www.aa.com</URL>
          </SCARR>
          <SCARR>
            <MANDT>000</MANDT>
            <CARRID>AB</CARRID>
            <CARRNAME>Air Berlin</CARRNAME>
            <CURRCODE>DEM</CURRCODE>
            <URL>http://www.airberlin.de</URL>
          </SCARR>
        </CARRIERS>
      </local.C1>
      <local.C2>
        <COUNT>2</COUNT>
      </local.C2>
    </prg:C2>
  </asx:heap>
</asx:abap>
```

40.6 Beispiel für eine Simple Transformation

Serialisierung einer geschachtelten Struktur. In folgendem ABAP-Programmabschnitt wird eine geschachtelte Struktur `struc1` mit der Simple Transformation ST_TRAFO nach `xml_string` serialisiert und mit der gleichen Transformation wieder deserialisiert.

```
DATA: BEGIN OF struc1,
        col1(10) TYPE c VALUE 'ABCDEFGHIJ',
        col2     TYPE i VALUE 111,
        BEGIN OF struc2,
          col1 TYPE d VALUE '20040126',
          col2 TYPE t VALUE '084000',
        END OF struc2,
      END OF struc1.

DATA: xml_string TYPE string,
      result LIKE struc1.

TRY.

    CALL TRANSFORMATION st_trafo
      SOURCE para = struc1
      RESULT XML xml_string.

    ...

    CALL TRANSFORMATION st_trafo
      SOURCE XML xml_string
      RESULT para = result.

  CATCH cx_st_error.

    ...

ENDTRY.
```

Die Simple Transformation ST_TRAFO habe folgende Form:

```
<?sap.transform simple?>
<tt:transform template="temp"
    xmlns:tt="http://www.sap.com/transformation-templates"
    version="0.1">

  <tt:root name="PARA"/>

  <tt:template name="temp">
    <X>
      <X1>
        <tt:value ref="PARA.COL1" />
```

```
      </X1>
      <X2>
        <tt:value ref="PARA.COL2" />
      </X2>
      <X3>
        <X1>
          <tt:value ref="PARA.STRUC2.COL1" />
        </X1>
        <X2>
          <tt:value ref="PARA.STRUC2.COL2" />
        </X2>
      </X3>
    </X>
  </tt:template>
</tt:transform>
```

Die Transformation besteht aus einem Template `temp`, das den Aufbau des XML-Dokuments definiert und Beziehungen zwischen Werteknoten und Komponenten der Struktur herstellt. Das Ergebnis der Transformation sieht wie folgt aus, wobei Zeilenumbrüche und Einrückungen zur besseren Verdeutlichung eingefügt wurden:

```
<X>
  <X1>ABCDEFGHIJ</X1>
  <X2>111</X2>
  <X3>
    <X1>2004-01-26</X1>
    <X2>08:40:00</X2>
  </X3>
</X>
```

Die Konvertierung der elementaren Datentypen ist wie bei asXML (siehe Tabelle 40.2). Die Rücktransformation erzeugt in der Struktur `result` den gleichen Inhalt wie in `struc1`.

Index

D

E

F

G

H

I

M

N

O

P

S

T

U

X

Y

Z