5 Programmdirektiven

Die Programmdirektiven von ABAP erlauben die Dokumentation quelltextbasierter Entwicklungsobjekte oder beeinflussen die Prüfergebnisse verschiedener Prüfwerkzeuge.

5.1 ABAP Doc

7.40

ABAP Doc ermöglicht ab Release 7.40 SP02 die Dokumentation von Deklarationen in ABAP-Programmen auf der Grundlage spezieller ABAP-Doc-Kommentare. In einer ABAP-Entwicklungsumgebung wie den *ABAP Development Tools* (ADT), die ABAP Doc unterstützt, wird der Inhalt von ABAP-Doc-Kommentaren ausgewertet, nach HTML konvertiert und geeignet dargestellt.

5.1.1 ABAP-Doc-Kommentare

Ein Kommentar für ABAP Doc wird durch die Zeichenfolge `"!` eingeleitet. Es handelt sich um eine spezielle Form eines mit `"` eingeleiteten normalen Zeilenendekommentars. Damit ein ABAP-Doc-Kommentar richtig ausgewertet werden kann, müssen folgende Regeln eingehalten werden:

- Ein ABAP-Doc-Kommentar ist entweder eine einzelne Kommentarzeile, die nichts außer dem Kommentar enthält, oder ein mehrzeiliger Block direkt aufeinanderfolgender Kommentarzeilen. Der Inhalt eines Blocks wird zu einem einzigen ABAP-Doc-Kommentar zusammengefasst.
- Ein ABAP-Doc-Kommentar (eine Zeile oder ein Zeilenblock) muss wie folgt mit genau einer Deklarationsanweisung verknüpft sein:
  - Wenn die Deklarationsanweisung keinen Kettensatz bildet, darf ein ABAP-Doc-Kommentar direkt vor der Deklarationsanweisung stehen und nicht durch Leerzeilen abgetrennt sein.
  - Wenn die Deklarationsanweisung einen Kettensatz bildet, muss der Doppelpunkt hinter dem Schlüsselwort stehen, und ein ABAP-Doc-Kommentar darf direkt vor dem Bezeichner jeder deklarierten Entität stehen.

  An anderen Stellen darf kein ABAP-Doc-Kommentar aufgeführt werden.
- Ein einzeiliger ABAP-Doc-Kommentar darf nicht leer sein. In Blöcken sind Zeilen ohne Inhalt als Formatierungsmittel erlaubt.
- Ein ABAP-Doc-Kommentar darf nur 7-Bit-ASCII-Zeichen enthalten.
- Ein ABAP-Doc-Kommentar kann spezielle Token und Tags enthalten, um die Parameterschnittstelle von Prozeduren zu dokumentieren oder um Formatierungen vorzunehmen (siehe Abschnitt 5.1.3, »Formatierungen«).
- Die Sonderzeichen `"`, `'`, `<`, `>` und `@` müssen in ABAP-Doc-Kommentaren durch `&quot;`, `&apos;`, `<`, `>` und `&#64;` maskiert werden.

Ein Verstoß gegen diese Regeln führt zu einer Warnung der Syntaxprüfung.

Beispiel
Grundlegende Verwendung von ABAP-Doc-Kommentaren als einzelne Zeilen, als Block und in Kettensätzen.

```
"! Basic usage of ABAP Doc
CLASS demo DEFINITION.
  PUBLIC SECTION.
    "! Constant character string for a single blank.
    CONSTANTS blank TYPE string VALUE ` `.
    "! Method to fill the private structure struct with values
    "! and append it to internal table itab.
    METHODS meth.
  PRIVATE SECTION.
    DATA:
      "! Three-component structure
      BEGIN OF struct,
        "! Component one
        comp1 TYPE i,
        "! Component two
        comp2 TYPE i,
        "! Component three
        comp3 TYPE i,
      END OF struct,
      "! Internal table of structure struct
      itab LIKE TABLE OF struct.
ENDCLASS.
```

5.1.2 Parameterschnittstelle von Prozeduren

Die Parameterschnittstelle von Prozeduren sowie von Ereignissen in Klassen kann im zugehörigen ABAP-Doc-Kommentar mit einer speziellen Syntax dokumentiert werden (siehe Tabelle 5.1).

Dokumentation für	Syntax
Schnittstellenparameter	`@parameter name|documentation`
klassenbasierte Ausnahme	`@raising name|documentation`
klassische Ausnahme	`@exception name|documentation`

Tabelle 5.1 Syntax zur Dokumentation von Parameterschnittstellen

Hinter `@parameter`, `@raising` bzw. `@exception` muss der Name `name` eines vorhandenen Parameters bzw. einer Ausnahme angegeben werden. Dahinter muss, abgetrennt durch das Zeichen `|`, die zugehörige Dokumentation folgen. Diese Dokumentation wird durch das nächste `@parameter`, `@raising` bzw. `@exception` oder durch das Ende des ABAP-Doc-Kommentars abgeschlossen. Hinter einer Schnittstellendokumentation darf also keine andere Dokumentation mehr folgen als eine weitere Schnittstellendokumentation. Jeder Schnittstellenparameter bzw. jede Ausnahme darf nur einmal aufgeführt werden.

Hinweis
Die Anordnung der Dokumentation der Parameterschnittstelle von Prozeduren ist unabhängig von der Anordnung der Zeilen in einem ABAP-Doc-Block. Aus Gründen der Lesbarkeit sollte aber jeder Parameter bzw. jede Ausnahme eine eigene Zeile einleiten. Aus den gleichen Gründen sollte die Reihenfolge der Parameter und Ausnahmen im ABAP-Doc-Kommentar der Reihenfolge der Deklarationen entsprechen.

Beispiel
Verwendung von ABAP-Doc-Kommentaren für die Parameterschnittstelle einer Methode.

```
"! Method to check if two sources are identical
"! and that returns a corresponding boolean value.
"!
"! @parameter source1     | First source
"! @parameter source2     | Second source
"! @parameter ignore_case | Pass abap_true to ignore case
"!
"! @parameter result      | Returns abap_true if sources are identic
"!
"! @raising   cx_invalid_source
"!                        | One of the sources is empty
METHODS compare
  IMPORTING
    source1       TYPE text
    source2       TYPE text
    ignore_case   TYPE abap_bool DEFAULT abap_false
  RETURNING
    VALUE(result) TYPE abap_bool
  RAISING
    cx_invalid_source.
```

5.1.3 Formatierungen

Innerhalb der Dokumentationstexte eines ABAP-Doc-Kommentars können die in Tabelle 5.2 aufgeführten Tags verwendet werden, um die Ausgabe der Dokumentation in einer geeigneten Entwicklungsumgebung zu formatieren.

Formatierung	Tag
Überschrift, Ebene1	`<h1>...</h1>`
Überschrift, Ebene2	`<h2>...</h2>`
Überschrift, Ebene3	`<h3>...</h3>`
Absatz	`<p>...</p>`
kursiv hervorgehobener Text	`<em>...</em>`
stark hervorgehobener Text	`<strong>...</strong>`
unnummerierte Liste	`<ul><li>...</li>...<li>...</li></ul>`
nummerierte Liste	`<ol><li>...</li>...<li>...</li></ol>`
Zeilenumbruch	` ` oder ` </br>`

Tabelle 5.2 Tags zur Formatierung der Dokumentationsausgabe

Ein geöffneter Tag muss geschlossen werden, bevor ein neuer Abschnitt des ABAP-Doc-Kommentars beginnt. Ein neuer Abschnitt wird durch `@parameter`, `@raising` oder `@exception` eingeleitet.

Die Tags sind eine Teilmenge von HTML-Tags, die in einer XHTML-konformen Notation angegeben werden müssen:

- Öffnendes und schließendes Tag müssen beide in Großbuchstaben oder beide in Kleinbuchstaben angegeben werden. Davon abgesehen hängt die Wirkung nicht von der Groß-/Kleinschreibung ab.
- Eine Schachtelung von Listen wird derzeit nicht unterstützt.
- Innerhalb der Tags `<h1>`, `<h2>`, `<h3>`, `<p>`, `<em>` oder `<strong>` darf kein `<p>`-, `<ul>`- oder `<ol>`-Tag stehen.
- Innerhalb der Tags `<ol>`, `<ul>` und `<br>` darf kein direkter Text angegeben werden.

Beispiel
Verwendung von Formatierungen in einem ABAP-Doc-Kommentar zu einer Klasse. Die ADT stellen die Dokumentation entsprechend formatiert dar.

```
"!<h1>Class demo</h1>
"!<p>This class must <strong>not</strong> be used productively.</p>
"!The class serves the following tasks:
"!<ul><li>Demo 1</li>
"!    <li>Demo 2</li>
"!    <li>Demo 3</li></ul>
"!<br/><br/>
CLASS demo DEFINITION.
  ...
ENDCLASS.
```

5.1.4 Kurztexte und deren Synchronisation

7.50 Teile von ABAP-Doc-Kommentaren können ab Release 7.50 als Kurztexte gekennzeichnet werden, und die Kurztexte von Klassen und Funktionsbausteinen und deren Komponenten können mit ABAP-Doc-Kommentaren synchronisiert werden. Um einen Teil eines ABAP-Doc-Kommentars als Kurztext zu kennzeichnen, kann er wie folgt getaggt werden:

```
<p class="shorttext">...</p>
```

Ein solcherart gekennzeichneter Absatz wird bei der Anzeige der Dokumentation in den ADT anstelle des Kurztextes, der in der ABAP Workbench zu sehen ist, als Überschrift angezeigt.

Um eine Synchronisation der ABAP-Doc-Kurztexte mit den in der ABAP Workbench angezeigten Kurztexten zu bewirken, kann der Tag optional wie folgt angegeben werden:

```
<p class="shorttext synchronized">...</p>
```

In diesem Fall ist die Länge des Kurztextes in ABAP Doc auf die Länge des entsprechenden Kurztextes der ABAP Workbench beschränkt, und er wird wie folgt mit dem zugehörigen Kurztext in der Originalsprache der Klasse bzw. des Funktionsbausteins synchronisiert:

- Wenn ein Kurztext in ABAP Doc im Quelltext angelegt oder geändert wird, wird der zugehörige Kurztext des Repository-Objekts beim Speichern des Quelltextes übernommen. Bei einem leeren Kurztext in ABAP Doc wird der Kurztext des Repository-Objekts gelöscht. Beim Entfernen eines gesamten Absatzes mit `class="shorttext synchronized"` bleibt der Kurztext des Repository-Objekts erhalten. Diese Synchronisation funktioniert unabhängig vom verwendeten Werkzeug.
- Wenn in der ABAP Workbench ein Kurztext einer Methode oder eines Funktionsbausteins geändert wird, zu dem in ABAP Doc ein Absatz mit `class="shorttext synchronized"` vorkommt, wird der Kurztext im Quelltext beim Speichern entsprechend ersetzt. Das Löschen eines Kurztextes in der ABAP Workbench führt zu einem leeren Absatz. Das Anlegen eines neuen Kurztextes in der ABAP Workbench, zu dem es noch keinen Kurztext in ABAP Doc gibt, führt derzeit noch nicht zum Anlegen des Absatzes im Quelltext.

Ein Kurztext von ABAP Doc ist als Teil des Quelltextes nicht an die Übersetzung angeschlossen. Da er bei einer Synchronisation den an die Übersetzung angeschlossenen Kurztext des Repository-Objekts in dessen Originalsprache ersetzt, muss auch er in der Originalsprache angegeben werden. Das ist eine Abweichung von der Regel, dass ABAP-Doc-Kommentare immer englisch sein sollten. Um die Originalsprache im Quelltext kenntlich zu machen, kann sie wie folgt optional angegeben werden:

```
<p class="shorttext" lang="...">...</p>
```

Das Attribut `lang` folgt dem HTML-Standard. Mit ihm muss die Originalsprache des Repository-Objekts als zweistelliges ISO-Kürzel angegeben werden, andernfalls kommt es zu einer Warnung der Syntaxprüfung. Das Attribut macht im Quelltext deutlich, in welcher Sprache der Kurztext angegeben werden sollte. Darüber hinaus ist es für zukünftige Erweiterungen zur Übersetzbarkeit von Kurztexten vorgesehen.

Beispiel
Öffnen Sie die Klasse `CL_DEMO_ABAP_DOC` in einem Quelltexteditor. Sie enthält ABAP-Doc-Kommentare für die Klasse selbst, einen Typ, eine Methode und ihre Parameter sowie für ein Attribut. Die ABAP-Doc-Kommentare umfassen Kurztexte, die mit den Kurztexten der ABAP Workbench in der Originalsprache Englisch synchronisiert sind. Sie können die Klasse in eine eigene temporäre Klasse kopieren, um die Synchronisation zu testen.

Hinweis
Die folgenden Richtlinien für allgemeine Kommentare gelten insbesondere auch für ABAP-DOC-Kommentare.

- **Programme englisch kommentieren**
  Die Einhaltung dieser Regel ist besonders wichtig, da eine in ABAP Doc vorgenommene Dokumentation als Teil des Quelltextes nicht in andere Sprachen übersetzt wird. Eine Ausnahme sind synchronisierte Kurztexte.
- **Kommentare richtig anordnen**
  Diese Regel betrifft die horizontalen Einrückungen, da die vertikale Anordnung vor Deklarationen syntaktisch festgeschrieben ist.
- **Zeichensatz von Quelltexten**
  Diese Regel wird von der Syntaxprüfung überprüft.

5.2 Pragmas

Pragmas sind Programmdirektiven, die verwendet werden können, um Warnungen verschiedener Prüfwerkzeuge auszublenden. Unterstützt werden:

- Warnungen der Syntaxprüfung des ABAP Compilers
- Warnungen der erweiterten Programmprüfung

Syntax
Ein Pragma hat den Aufbau:

```
##code[par][par]...
```

Es ist unempfindlich gegenüber Groß-/Kleinschreibung und enthält keine Leerzeichen. Der Pragma-Code (`code`) legt die Wirkung fest; Parameter (`par`) grenzen die Wirkung gegebenenfalls weiter ein.

Eine Meldung wird durch ein Pragma beeinflusst, wenn alle angegebenen Parameter mit den konkreten Parametern übereinstimmen. Die konkreten Parameter können ebenfalls dem Langtext zur Meldung entnommen werden. Vorgeschriebene Parameter sind im Langtext unterstrichen und dürfen nicht weggelassen werden. Nicht vorgeschriebene Parameter sind optional. Das Weglassen optionaler Parameter ist durch ein leeres Klammerpaar `[]` an der entsprechenden Position oder durch komplettes Weglassen eines Endstücks möglich.

Hinweise

- Ob zu einer Warnung der Syntaxprüfung ein Pragma existiert, kann man dem – in diesem Fall immer vorhandenen – Langtext der Meldung entnehmen. Die Beschreibung einer Meldung der erweiterten Programmprüfung führt ebenfalls das Pragma auf, das zum Ausblenden verwendet werden kann.
- Pragmas lösen die bis dahin gebräuchlichen Pseudokommentare ab, um Warnungen der erweiterten Programmprüfung auszublenden. Diese Pseudokommentare werden dadurch obsolet und sollten nicht mehr verwendet werden (siehe Abschnitt 55.3.1). Das Programm `ABAP_SLIN_PRAGMAS` zeigt, welche Pragmas anstelle der obsoleten Pseudokommentare zu verwenden sind.
- In einem Programm, das Pragmas zum Ausschalten von Warnungen verwendet, dürfen die Anweisung `SET EXTENDED CHECK` und der Pseudokommentar `#EC *` nicht mehr verwendet werden. Sie führen zu einer nicht ausschaltbaren Warnung der erweiterten Programmprüfung.
- Ein Pragma bei einer mit `TYPES` vorgenommenen Typdefinition – beispielsweise zum Ausblenden der Warnung zu redundanten Sekundärschlüsseln von Tabellentypen – wirkt bei nicht generischen Typen auch für Datendeklarationen mit `DATA` und verwandten Anweisungen, die sich über `TYPE` auf den Datentyp beziehen. Bei einem Bezug auf einen mit `TYPES` definierten generischen Datentyp, d. h. einen Tabellentyp, für den kein primärer Tabellenschlüssel definiert ist, wirkt ein dort angegebenes Pragma aber nicht auf die Datendeklaration, da diese implizit einen vervollständigten Tabellentyp verwendet. In diesem Fall muss das Pragma gegebenenfalls noch einmal angegeben werden.

Beispiele
Ein Beispiel für ein Pragma für Warnungen der Syntaxprüfung ist:

```
##SHADOW
```

Dieses Pragma kann genutzt werden, um bei einer Methodendefinition eine Syntaxwarnung auszublenden, die die Verschattung einer eingebauten Funktion meldet. Das Pragma hat einen optionalen Parameter, in dem zusätzlich der Name der Funktion angegeben werden kann:

```
##SHADOW[SUBSTRING]
```

Eine Warnung zu `SUBSTRING` wird durch die folgenden Pragmas unterdrückt, aber nicht durch `SHADOW[FIND]`.

- `##SHADOW`
- `##SHADOW[SUBSTRING]`
- `##SHADOW[]`

Das folgende Beispiel zeigt Pragmas zum Ausblenden von Warnungen der erweiterten Programmprüfung.

```
DATA text TYPE string   ##needed.
text  = 'Hello Pragmas' ##no_text.
```

In einigen Beispielen zu Schlüsselzugriffen auf interne Tabellen (Lesen, Löschen) werden Syntaxwarnungen durch das zugehörige Pragma ausgeblendet. Die Pragmas wurden im Langtext der Syntaxwarnungen (Auswahl des i-Knopfes) gefunden.

Ein Pragma gilt für die aktuelle Anweisung, d. h. für die Anweisung, die am nächsten dem ».« oder »,« endet. Pragmas, die vor dem »:« eines Kettensatzes stehen, gelten für den gesamten Kettensatz. Pragmas, die bei Aufruf eines Makros stehen, gelten für alle Anweisungen des Makros.

Aus Gründen der Lesbarkeit dürfen Pragmas nur an bestimmten Positionen im Programmtext stehen:

- am Beginn einer Zeile, nach beliebig vielen Leerzeichen
- am Ende einer Zeile, höchstens gefolgt von ».«, »,« oder »:«
- nicht jedoch nach ».«, »,« oder »:«

An zulässigen Positionen dürfen auch mehrere Pragmas – getrennt durch Leerzeichen – hintereinanderstehen.

Unbekannte, formal fehlerhafte, falsch positionierte oder falsch parametrisierte Pragmas führen selbst zu einer Syntaxwarnung.

5.3 Pseudokommentare für den Code Inspector

Pseudokommentare sind Programmdirektiven zur Beeinflussung von Prüfungen und Testabläufen. Pseudokommentare sind weitestgehend obsolet und werden durch Pragmas oder

echte Zusätze abgelöst. Pseudokommentare sind nur noch für den Code Inspector notwendig. Die Pseudokommentare für die erweiterte Programmprüfung und für Testklassen sind obsolet (siehe Abschnitt 55.3).

Eine Zeichenfolge "#EC hinter einer Anweisung oder einem Teil einer Anweisung, der ein Kürzel mit dem Präfix CI_ folgt, definiert einen Pseudokommentar für den Code Inspector. Mit diesen Pseudokommentaren können bestimmte Warnungen des Code Inspectors für die betreffende Anweisung ausgeblendet werden. Die möglichen Kürzel sind beim Code Inspector bzw. bei der Ausgabe von dessen Meldungen dokumentiert.

Hinweis
Es kann nur ein Pseudokommentar pro Programmzeile angegeben werden. Um mehrere Pseudokommentare für eine Anweisung anzugeben, muss diese also auf mehrere Zeilen aufgeteilt werden.

Beispiel
Der folgende Join-Ausdruck umgeht die SAP-Pufferung und führt deshalb zu einer Warnung des Code Inspectors. Wenn die SELECT-Anweisung aber Teil einer Anwendung ist, die selbst für eine Pufferung ausgewählter Daten sorgt, kann die Warnung wie gezeigt ausgeblendet werden. Ein zusätzlicher normaler Kommentar verdeutlicht einem Leser des Quelltextes den Grund für die Verwendung des Pseudokommentars.

```
SELECT d~object, h~dokldate, h~dokltime     "#EC CI_BUFFJOIN
  FROM dokil AS d                          "Buffering is done
       INNER JOIN dokhl AS h               "by application
         ON h~id         = d~id     AND    "with Shared Objects
            h~object     = d~object AND
            h~typ        = d~typ    AND
            h~langu      = d~langu  AND
            h~dokversion = d~version
  WHERE d~id    = 'SD'   AND
        d~typ   = 'E'    AND
        d~langu = @langu AND
        d~object LIKE 'AB%'
  INTO CORRESPONDING FIELDS OF TABLE @docu_tab.
```

17 Werte erzeugen

Die Werte der Attribute einer neu erzeugten Instanz einer Klasse können, falls vorhanden, mit dem Instanzkonstruktor der Klasse konstruiert werden. Die Eingabeparameter des Instanzkonstruktors können mit dem `EXPORTING`-Zusatz der Anweisung `CREATE OBJECT` oder durch Aktualparameter für den Instanziierungsoperator `NEW` versorgt werden.

Die Werte dynamisch erzeugter oder auch statisch deklarierter Datenobjekte können ab Release 7.40 SP02 über folgende Konstruktorausdrücke konstruiert werden: 7.40

- Bei der dynamischen Erzeugung anonymer Datenobjekte mit dem Instanziierungsoperator `NEW` können Werte für alle Datentypen, insbesondere auch strukturierter und tabellarischer Typen, konstruiert und dem erzeugten Datenobjekt zugewiesen werden (siehe Abschnitt 16.3, »Instanziierungsoperator«).
- Der Wertoperator `VALUE` kann verwendet werden, um den Inhalt vorhandener komplexer Datenobjekte (Strukturen, interne Tabellen) zu konstruieren, und geht damit über die Funktionalität des `VALUE`-Zusatzes hinaus.

Der Wertoperator `VALUE` kann wie jeder Konstruktorausdruck an allgemeinen Ausdruckspositionen und funktionalen Operandenpositionen eingesetzt werden, wozu insbesondere auch die rechte Seite einer Zuweisung an eine Inline-Deklaration gehört.

Syntax von VALUE

```
... VALUE type( ... ) ...
```

Ein Konstruktorausdruck mit dem Wertoperator `VALUE` erzeugt ein Resultat eines mit `type` angegebenen Datentyps. Für `type` können angegeben werden:

- ein nicht generischer Datentyp `dtype`
- Das Zeichen `#` als Symbol für den Operandentyp. Diese Angabe ist nur möglich, wenn der an einer Operandenposition benötigte Datentyp eindeutig und vollständig erkennbar ist. Ausnahmen von dieser Regel sind:
  - Bei der Übergabe eines Initialwertes `VALUE #( )` an einen generisch typisierten Formalparameter wird der Typ aus dem generischen Typ abgeleitet.
  - Bei der Konstruktion einer Struktur oder einer internen Tabelle kann der Operand ab Release SP08 hinter `BASE` ausgewertet werden. 7.40
  - Verwendung für einen einzelnen Tabellenausdruck `VALUE #( table_exp )`

Der Operator erzeugt Initialwerte für beliebige nicht generische Datentypen, konstruiert den Inhalt von strukturierten Typen sowie Tabellentypen, steuert die Art des Resultats von Tabellenausdrücken und ermöglicht dabei die Angabe eines Standardwertes für nicht gefundene Zeilen.

Der Inhalt des Resultats wird durch die in den Klammern angegebenen Parameter bestimmt. Die Syntax der Parameterübergabe hängt bei der Konstruktion eines Wertes vom verwendeten Typ ab, wobei es für jeden möglichen Typ entsprechend spezialisierte Arten der Parame-

terübergabe gibt. Wenn als Parameter ein einzelner Tabellenausdruck angegeben ist, konstruiert VALUE keinen Wert, sondern steuert die Art dessen Resultats.

Bei einer Zuweisung eines Konstruktorausdrucks mit VALUE an ein Datenobjekt wird direkt mit diesem gearbeitet. Es wird vollständig mit einem Initial- oder Startwert überschrieben, bevor ihm die in den Klammern angegebenen Werte zugewiesen werden.

Hinweise

- Der Wertoperator VALUE verwendet zur Werterzeugung im Wesentlichen die gleiche Syntax wie der Instanziierungsoperator NEW.
- Elementare Datentypen sowie Referenztypen können bei VALUE zur Konstruktion von Werten außer bei der Erzeugung eines Initialwertes nicht explizit angegeben werden. Anders als beim Instanziierungsoperator NEW ist das entsprechende Ergebnis genauso durch direkte Zuweisungen erreichbar. Aus diesem Grund ist auch die bei NEW mögliche Angabe von unbenannten Argumenten als Einzelwerte nicht notwendig und deshalb nicht erlaubt. Davon nicht betroffen ist die Verwendung von VALUE zur Steuerung von Tabellenausdrücken, wofür beliebige passende Datentypen angegeben werden können.
- Da mit den Resultaten von VALUE zur Konstruktion von Werten außer bei der Erzeugung eines Initialwertes nicht arithmetisch gerechnet werden kann, können solche Konstruktorausdrücke nicht direkt an den Operandenpositionen arithmetischer Ausdrücke aufgeführt werden. Davon nicht betroffen sind Konstruktorausdrücke mit VALUE zur Steuerung von Tabellenausdrücken, mit denen bei passenden Ergebnissen auch gerechnet werden kann.
- Die Lücke, dass mit VALUE keine elementaren Datenobjekte an Operandenpositionen konstruiert werden können, wird durch den Konvertierungsoperator CONV geschlossen.
- Wenn ein Konstruktorausdruck mit VALUE nicht als Quelle einer Zuweisung an ein Datenobjekt verwendet wird, erzeugt der Wertoperator VALUE ein neues temporäres Datenobjekt, dessen Datentyp durch den angegebenen Typ und dessen Inhalt durch die übergebenen Parameter bestimmt wird. Dieses Datenobjekt wird als Operand einer Anweisung verwendet und nach Gebrauch wieder gelöscht. Das Löschen erfolgt bei Abschluss der aktuellen Anweisung bzw. bei der Auswertung eines relationalen Ausdrucks nach dem Feststellen des Wahrheitswertes.
- Bei einer Zuweisung an ein Datenobjekt wird kein temporäres Datenobjekt erzeugt, sondern direkt mit der Zielvariablen gearbeitet. Diese wird vor der Zuweisung der in den Klammern angegebenen Werte initialisiert bzw. vollständig überschrieben. Ihr ursprünglicher Wert steht aber noch in einem optionalen LET-Ausdruck zur Verfügung. Hierin unterscheidet sich VALUE vom Instanziierungsoperator NEW.

Beispiel

Konstruktion der Werte einer inline deklarierten Struktur vom Typ T100.

```
DATA(wa) = VALUE t100( sprsl = 'E'
                       arbgb = 'DEMO'
                       msgnr = '111'
                       text = '...' ).
```


17.1 Initialwert für alle Typen

Syntax

```
... VALUE dtype|#( ) ...
```

Wenn in den Klammern kein Parameter angegeben ist, erhält der Rückgabewert seinen typspezifischen Initialwert. Dies ist für beliebige nicht generische Datentypen dtype möglich. Das Zeichen # kann für entsprechende statisch erkennbare Operandentypen stehen. Des Weiteren wird bei der Übergabe von VALUE #( ) an einen generisch typisierten Formalparameter der Typ wie folgt aus dessen generischem Typ abgeleitet:

- string bei csequence und clike
- xstring bei xsequence
- decfloat34 bei numeric und decfloat
- p der Länge 8 ohne Nachkommastellen bei generischem p
- der Standardschlüssel bei einem Standardtabellentyp mit generischem primären Tabellenschlüssel

Andere generische Datentypen außer Tabellentypen, die explizit generisch bezüglich ihrer sekundären Tabellenschlüssel sind, können nicht konkretisiert werden und führen zu einem Syntaxfehler. Dies gilt insbesondere für die Typen c, n und x mit generischen Längen.

Hinweise

- Während VALUE mit Wertübergabe nur der Konstruktion bestimmter komplexer Werte dient (Strukturen, interne Tabellen), ist VALUE ohne Wertübergabe ein allgemeines Mittel zur Erzeugung typgerechter Initialwerte an beliebigen Operandenpositionen.
- Die Regeln für die Ableitung des Typs bei Angabe von # für Aktualparameter, die an generisch typisierte Formalparameter übergeben werden, können Syntaxfehler in einem Programm verhindern, das eine Prozedur aufruft, bei der die vollständige Typisierung eines Formalparametertyps nachträglich durch Umstellung auf einen generischen Typ verallgemeinert wurde.

Beispiel

Erzeugung einer passenden initialen Struktur für einen nicht optionalen Eingabeparameter einer Methode.

```
CLASS c1 DEFINITION.
  PUBLIC SECTION.
    TYPES: BEGIN OF t_struct,
             col1 TYPE i,
             col2 TYPE i,
           END OF t_struct.
    CLASS-METHODS m1 IMPORTING p TYPE t_struct.
ENDCLASS.
CLASS c1 IMPLEMENTATION.
  METHOD m1.
    ...
```

```
  ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
  c1=>m1( VALUE #( ) ).
```

17.2 Strukturen

Syntax

```
... VALUE dtype|#( [let_exp]
                   [BASE dobj]
                   comp1 = dobj1 comp2 = dobj2 ... ) ...
```

Ein strukturierter Wert wird genauso konstruiert wie mit dem Instanziierungsoperator NEW (siehe Abschnitt 16.3.3, »Strukturen«). Wenn der VALUE-Operator als Quelle einer Zuweisung an eine Struktur verwendet wird, wird diese nach der Auswertung eines eventuellen LET-Ausdrucks zunächst initialisiert bzw. wird erst das Datenobjekt dobj hinter BASE (ab Release 7.40 SP08) zugewiesen, und dann werden die Zuweisungen in der Klammer von links nach rechts direkt mit den Strukturkomponenten als Zielfeldern ausgeführt.

Hinweise

- Die Regel, dass eine Zielstruktur erst vollständig überschrieben und dann direkt bearbeitet wird, kann zu überraschenden Ergebnissen führen, wenn Strukturkomponenten der linken Seite auf der rechten Seite als zuzuweisende Datenobjekte angegeben werden. Es werden nicht erst die Zuweisungen der rechten Seite ausgewertet und zugewiesen, sondern es wird bei jeder Zuweisung der gerade aktuelle Wert verwendet. Wenn die ganze Struktur oder Strukturkomponenten der linken Seite auf der rechten Seite benötigt werden, können sie aber mit einem LET-Ausdruck in lokalen Hilfsvariablen gespeichert werden, da dieser zuerst ausgewertet wird.
- Wenn bei einer Zuweisung an eine vorhandene Struktur die Zielstruktur als dobj hinter BASE (ab Release 7.40 SP08) angegeben ist, findet keine Zuweisung vor der Auswertung der Komponentenzuweisungen statt, sondern die Zielstruktur behält einfach ihren Wert.

Beispiel

Drei unterschiedliche Möglichkeiten, eine geschachtelte Struktur struct mit Werten zu versorgen. Die Struktur wird jedes Mal mit den gleichen Werten versorgt.

```
TYPES:  BEGIN OF t_col2,
          col1 TYPE i,
          col2 TYPE i,
        END OF t_col2.
TYPES: BEGIN OF t_struct,
         col1 TYPE i,
         col2 TYPE t_col2,
       END OF t_struct.
DATA: struct TYPE t_struct,
      col2 TYPE t_col2.
```



```
struct = VALUE t_struct( col1 = 1
                         col2-col1 = 1
                         col2-col2 = 2 ).
col2   = VALUE   t_col2( col1 = 1
                         col2 = 2 ).
struct = VALUE t_struct( col1 = 1
                         col2 = col2 ).
struct = VALUE t_struct( col1 = 1
                         col2 = VALUE #( col1 = 1
                                         col2 = 2 ) ).
```

17.3 Interne Tabellen

Syntax

```
... VALUE dtype|#( [let_exp]
                   [BASE itab]
                   [FOR for_exp1
                    FOR for_exp2
                    ... ]
                   ( line_spec1 )
                   ( line_spec2 )
                     ... ) ...
```

Eine interne Tabelle wird genauso konstruiert wie mit dem Instanziierungsoperator NEW (siehe Abschnitt 16.3.4, »Interne Tabellen«). Wenn der VALUE-Operator als Quelle einer Zuweisung an eine interne Tabelle verwendet wird, wird diese nach der Auswertung eines eventuellen LET-Ausdrucks erst initialisiert bzw. bekommt den Inhalt von itab zugewiesen, und dann werden die Angaben line_spec ausgewertet und direkt in die Zieltabelle eingefügt.

Hinweise

- Bei einer Zuweisung des Konstruktorausdrucks an eine interne Tabelle können deren vorhandene Zeilen nicht direkt als Argument in line_spec verwendet werden, da diese vor der Auswertung line_spec gelöscht bzw. mit dem Inhalt von itab überschrieben werden. Wenn die ganze interne Tabelle oder Zeilen der linken Seite auf der rechten Seite benötigt werden, können sie aber mit einem LET-Ausdruck in lokalen Hilfsvariablen gespeichert werden, da dieser vorher ausgewertet wird.
- Ohne den Zusatz BASE kann der Inhalt von Tabellen mit dem Wertoperator nur neu konstruiert, aber nicht erweitert werden. Durch die Angabe der gleichen Tabelle hinter BASE (ab Release 7.40 SP08), der der Konstruktorausdruck zugewiesen wird, können in diese auch weitere Zeilen eingefügt werden.
- Wenn bei einer Zuweisung an eine vorhandene interne Tabelle die Zieltabelle als itab hinter BASE angegeben ist, findet keine Zuweisung vor der Auswertung der Angaben line_spec statt, sondern die Zieltabelle behält einfach ihren Wert.
- Eine Tabellenfilterung lässt sich effizienter mit dem Filteroperator FILTER erreichen.

- Für Aufgaben, die sich sowohl mit Tabellen-Comprehensions als auch mit speziellen Zuweisungen für Komponenten, insbesondere dem Komponentenoperator `CORRESPONDING`, lösen lassen, empfiehlt sich in der Regel die Verwendung von Zuweisungen.
- Bei Verwendung des `VALUE`-Operators ist zu beachten, dass bei Zuweisungen an interne Tabellen diese auch bei Tabellen-Comprehensions nach der Auswertung eines eventuellen `LET`-Ausdrucks initialisiert werden bzw. den Inhalt von `itab` hinter `BASE` zugewiesen bekommen und dass dann direkt mit der Zieltabelle gearbeitet wird. Die ursprüngliche Tabelle kann deshalb nicht direkt in den `FOR`-Ausdrücken verwendet werden, außer sie wird hinter `LET` einer Hilfsvariablen zugewiesen.

Beispiel 1
Konstruktion von internen Tabellen mit elementarem Zeilentyp. `jtab` wird mit drei Zeilen befüllt, `itab` mit sechs Zeilen. Die erste eingefügte Zeile von `itab` ist initial, und die letzten drei Zeilen werden der zuvor gefüllten Tabelle `jtab` entnommen.

```
TYPES t_itab TYPE TABLE OF i WITH EMPTY KEY.
DATA(jtab) = VALUE t_itab( ( 10 ) ( 20 ) ( 30 ) ).
DATA(itab) = VALUE t_itab( ( ) ( 1 ) ( 2 ) ( LINES OF jtab ) ).
cl_demo_output=>display( itab ).
```

Beispiel 2
Konstruktion einer internen Tabelle mit tabellarischem Zeilentyp und Befüllen mit zwei Zeilen. Der ersten Zeile wird eine zuvor befüllte Tabelle zugewiesen. Die zweite Zeile wird mit `VALUE` konstruiert.

```
TYPES: t_itab1 TYPE TABLE OF i         WITH EMPTY KEY,
       t_itab2 TYPE TABLE OF t_itab1   WITH EMPTY KEY.
DATA itab1 TYPE t_itab1.
DATA itab2 TYPE t_itab2.
itab1 = VALUE #( ( 1 ) ( 2 ) ( 3 ) ).
itab2 = VALUE #( ( itab1 )
                 ( VALUE t_itab1( ( 4 ) ( 5 ) ( 6 ) ) ) ).
```

Beispiel 3
Verwendung von `BASE` für das Anhängen von Zeilen an vorhandene Zeilen einer internen Tabelle ab Release 7.40 SP08.

```
TYPES itab TYPE TABLE OF string WITH EMPTY KEY.
DATA(itab) =
  VALUE itab(
    ( `a` ) ( `b` ) ( `c` ) ).
...
itab =
  VALUE #(
    BASE itab
    ( `d` ) ( `e` ) ( `f` ) ).
cl_demo_output=>display( itab ).
```

TEIL 14

Obsolete Anweisungen

Die in den folgenden Kapiteln beschriebenen Sprachelemente sind obsolet und stehen nur noch aus Gründen der Kompatibilität zu früheren Releases zur Verfügung. Die Sprachelemente können in älteren Programmen noch vorhanden sein, sollen aber nicht mehr verwendet werden.

Die meisten der hier aufgeführten obsoleten Sprachelemente sind in Klassen syntaktisch verboten. Sie können also ohnehin nur noch außerhalb von Klassen verwendet werden. Für alle obsoleten Sprachelemente gibt es Ersatzkonstrukte, die die Effektivität und Lesbarkeit von Programmen erhöhen.

54 Obsolete Programmeigenschaften

Die strikten Modi der Syntaxprüfung von Open SQL erfordern, dass diese Programmeigenschaften nicht abgeschaltet sind.

54.1 Obsoletes Abschalten der Unicode-Prüfung

Standardmäßig ist beim Anlegen eines Programms die Programmeigenschaft UNICODEPRÜFUNGEN aktiv zum Einschalten der Unicode-Prüfungen gesetzt. Ein Programm, bei dem die Unicode-Prüfungen eingeschaltet sind, ist ein Unicode-Programm. Diese Programmeigenschaft darf nicht zurückgesetzt werden.

Unicode-Programme funktionieren in Unicode- und Nicht-Unicode-Systemen. Nicht-Unicode-Programme funktionieren nur in Nicht-Unicode-Systemen.

Ab Release 7.50 von SAP NetWeaver werden nur noch Unicode-Systeme unterstützt. Nicht-Unicode-Programme können ab diesem Release nicht mehr ausgeführt werden. Die Änderbarkeit der Programmeigenschaft wird nur aus Kompatibilitätsgründen angeboten. In aller Regel soll sie nur dazu dienen, die Unicode-Prüfungen in Programmen, in denen sie noch nicht aktiv sind, einzuschalten.

54.2 Obsoletes Abschalten der Festpunktarithmetik

Standardmäßig ist beim Anlegen eines Programms die Programmeigenschaft FESTPUNKTARITHMETIK gesetzt. Diese Programmeigenschaft darf nicht zurückgesetzt werden.

Die Änderbarkeit der Programmeigenschaft wird nur aus Kompatibilitätsgründen angeboten. In aller Regel soll sie nur dazu dienen, die Festpunktarithmetik in Programmen, in denen sie noch nicht aktiv ist, einzuschalten. Bei ausgeschalteter Festpunktarithmetik wird die Stellung des Dezimaltrennzeichens von gepackten Zahlen (Typ `p`) nur bei der Ausgabe auf dem klassischen Dynpro, bei Zuweisungen an Felder der Typen `c` und `string` und bei der Formatierung mittels `WRITE [TO]` berücksichtigt, nicht jedoch bei Berechnungen.