SDF-P V2.4A

Programmieren in der Kommandosprache

Ausgabe Mai 2007

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@fujitsu-siemens.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2000

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2000 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Inhalt

1 Einleitung

1.1 Kurzbeschreibung des Produkts

Das Softwareprodukt SDF-P ist eine Prozedursprache, die die Kommandosprache des BS2000 zu einer Programmiersprache erweitert, in der strukturiertes Programmieren analog zu höheren Programmiersprachen möglich ist. Bereits der Anfänger kann mit SDF-P kleinere Prozeduren schnell und einfach erstellen. Dazu werden auch das Erstellen und die Wartbarkeit großer und komplexer Prozeduren stark vereinfacht. Schließlich ist durch das Zuweisen strukturierter Variablenströme (kurz: S-Variablenströme) möglich, strukturierte Ausgaben in Variablen zu speichern, die in vielfacher Weise weiterverarbeitet werden können: z.B. können diese in grafische Benutzeroberflächen umgelenkt werden.

Prozeduren, die entsprechend den Regeln von SDF-P erstellt werden, heißen strukturierte Prozeduren oder S-Prozeduren. (Prozeduren, die nicht nach SDF-P-Regeln erstellt werden, heißen Nicht-S-Prozeduren.)

SDF-P unterstützt auf der Kommandoebene Funktionen höherer Programmiersprachen.

Im Lieferumfang von SDF-P sind vordefinierte Funktionen enthalten, mit denen unter anderem Variablen bearbeitet und konvertiert oder Umgebungsinformationen wie Auftragsstatus, Rechnername oder das Tagesdatum abgefragt werden können. Diese Funktionen können in Prozeduren so eingesetzt werden wie Funktionen höherer Programmiersprachen in Programmen.

SDF-P bietet im BS2000 ein Variablenkonzept, wie es aus höheren Programmiersprachen bekannt ist. Dabei unterstützt SDF-P nicht nur einfache, sondern auch zusammengesetzte Variablen. Charakterisiert sind Variablen in SDF-P außerdem durch ihren Datentyp und ihre Lebensdauer oder Sichtbarkeit. Variablenbearbeitung ist in SDF-P sowohl an der Kommando- als auch an der Programmschnittstelle möglich.

Wie in höheren Programmiersprachen wird der Ablauf der Prozedur durch Schleifen und Verzweigungen gesteuert, die mit SDF-P-Kommandos realisiert werden. Die Namen dieser Kommandos wurden so gewählt, wie sie aus verschiedenen Programmiersprachen bekannt sind: WHILE, FOR, REPEAT, IF, ELSE.

SDF-P ist eine blockorientierte Programmiersprache. Das heißt, ein wesentliches Merkmal von Prozeduren unter SDF-P ist der Aufbau von Prozeduren durch so genannte Kommandoblöcke. Nicht nur Schleifen und Verzweigungen bilden solche Kommandoblöcke, der Programmierer kann auch beliebige zusammengehörende Prozedurteile als Kommandoblock definieren.

Die Bildung von Kommandoblöcken hat zum einen den Vorteil, dass der Prozeduraufbau übersichtlich bleibt, zum anderen ist auch die Fehlerbehandlung blockorientiert und kann so auf definierte Prozedurteile angewendet werden.

Produktstruktur

Die gesamte Funktionalität von SDF-P ist in zwei Subsystemen realisiert: SDF-P und SDFPBASY (Liefereinheit SDF-P-BASYS) . Das Subsystem SDF-P ist kostenpflichtig, das Subsystem SDFPBASY wird mit dem BS2000-Grundausbau ausgeliefert. Prozeduren mit der Funktionalität von SDF-P können unter folgenden Bedingungen ausgeführt werden:

– Das Subsystem SDF-P ist installiert und geladen.
– Das Subsystem SDF-P ist nicht installiert, aber die Prozeduren liegen in einem (kompilierten) Zwischenformat vor, das auf einer Anlage mit dem Subsystem SDF-P erzeugt wurde.
  Auf einer Anlage ohne SDF-P können S-Prozeduren syntaktisch analysiert werden, mit Ausnahme von Kontrollflusskommandos und dem Kommando COMPILE-PROCEDURE. Zur Ausführungszeit werden Prozeduren mit kostenpflichtiger Funktionalität aber zurückgewiesen, wenn sie nicht in kompilierter Form vorliegen.
  Kommandos und Funktionen des Subsystems SDF-P-BASYS sind auch im Handbuch „Kommandos, Band 6“ [4] beschrieben.

Die Versionsabhängigkeiten zwischen SDF-P und SDF-P-BASYS sind im Abschnitt „SW-Konfiguration“ auf Seite 810 beschrieben.

Zielgruppe

Dieses Handbuch wendet sich zum einen an Anwender und Benutzer des BS2000, die Prozeduren erstellen, um ihre tägliche Arbeit zu erleichtern. Zum anderen wendet es sich an Programmierer oder Systembetreuer, die zum Beispiel Aufgaben der Systembetreuung über komplexe Prozeduren lösen.

Da Prozeduren in SDF-P ähnlich wie in höheren Programmiersprachen erstellt werden, wird der „Ersteller“ der Prozedur allgemein als Programmierer bezeichnet.

1.2 Konzept des Handbuchs

In diesem Handbuch werden das Prozedurkonzept von SDF-P, das Erstellen von S-Prozeduren und alle SDF-P-Schnittstellen beschrieben. Welche Regeln für Nicht-S-Prozeduren gelten, ist nicht Thema des Handbuchs. Das Verhalten von Nicht-S-Prozeduren wird nur dann beschrieben, wenn Inkompatibilitäten zwischen S- und Nicht-S-Prozeduren zu berücksichtigen sind.

Im Folgenden erhalten Sie einen inhaltlichen Überblick über die einzelnen Kapitel:

Kapitel 1 „Einleitung“
enthält eine Kurzbeschreibung des BS2000-Produkts SDF-P und gibt Hinweise zur Benutzung des Handbuchs.

Kapitel 2 „Schnelleinstieg SDF-P“
führt den Benutzer in die Erstellung von S-Prozeduren ein, ausgehend von der Erstellung von Nicht-S-Prozeduren.

Kapitel 3 „Prozedurkonzept von SDF-P“
beschreibt die Richtlinien, nach denen die Prozedursprache SDF-P konzipiert wurde.

Kapitel 4 „S-Prozeduren erstellen“
erklärt, wie in SDF-P Prozeduren aufgebaut sind und wie sie richtig erstellt werden.

Kapitel 5 „Prozeduren aufrufen und steuern“
stellt dar, was der Benutzer wissen muss, um Prozeduren im Vordergrund oder im Hintergrund aufzurufen zu können, oder wenn er sie schachteln will usw.

Kapitel 6 „Variablen in S-Prozeduren verwenden“
enthält das Variablenkonzept und die Richtlinien für die Variablendeklaration und Variablenverarbeitung.

Kapitel 7 „S-Variablenströme“
erklärt die strukturierte Ausgaben von Kommandos und Programmen in S-Variablen.

Kapitel 8 „Funktionen“
beschreibt den Einsatz von Systemverwalter-Funktionen und vordefinierten Funktionen.

Kapitel 9 „Ausdrücke“
stellt die Verwendung von Operanden, Operatoren usw. dar.

Kapitel 10 „S-Prozeduren optimieren“
beschreibt, wie S-Prozeduren bei der Erstellung in Bezug auf die Performance optimiert werden können.

Kapitel 11 „S-Prozeduren testen“
beschreibt, welche Mittel dem Programmierer von S-Prozduren bei der Fehlersuche zur Verfügung stehen.

Kapitel 12 „Nicht-S-Prozeduren umstellen“
gibt u.a. Hinweise, wie Vordergrund- und Hintergrund-Nicht-S-Prozeduren umgestellt werden können.

Kapitel 13 „Programmschnittstellen“
ist ein Nachschlageteil über Programmschnittstellen.

Kapitel 14 „Vordefinierte Funktionen“
ist ein Nachschlageteil über vordefinierte Funktionen.

Kapitel 15 „SDF-P-Kommandos„
ist ein Nachschlageteil über SDF-P-Kommandos.

Kapitel 16 „„Installation und Konfiguration“
nennt die zur Installation benötigten Syntax-, Meldungs- und Produktdateien und beschreibt die Versionsabhängigkeiten der involvierten Subsysteme.

Kapitel 17 „Meldungen“
ist eine Zusammenstellung aller Meldungen der Meldungsklasse SDP.

Am Ende des Handbuchs finden Sie ein Fachwort-, Literatur- und Stichwortverzeichnis.

Verwendete Darstellungsmittel

Die Konventionen, die für die Kapitel des Nachschlageteils gelten, sind dort jeweils beschrieben. Im Einführungsteil gelten folgende Konventionen:

Hinweis
Das Wort „*Hinweis*“ vor einem eingerückten Absatz zeigt an, dass der folgende Absatz wichtige Informationen enthält.

[1]
Zahlen in eckigen Klammern verweisen im Text auf die entsprechende Position im Literaturverzeichnis am Ende des Handbuchs.

Fettdruck
Wo Syntaxdarstellungen erläutert werden, sind die Zeilen, die aktuell erläutert werden, halbfett dargestellt.
Im Übrigen gelten für Syntaxdarstellungen die Regeln, wie sie in den entsprechenden Kapiteln des Nachschlageteils beschrieben sind.

`SYNTAX /Beispiel`
Syntaxdarstellungen und Beispiel-Eingaben und -Ausgaben werden durch andere Schriften hervorgehoben. Syntaxdarstellungen sind außerdem von einem Rahmen umgeben.

`[ ]`
Eckige Klammern in Syntaxdarstellungen: Die Zeichen innerhalb der Klammern dürfen weggelassen werden.

1.3 Änderungen gegenüber der letzten Handbuchausgabe

Das vorliegende Handbuch beschreibt die Funktonalität von SDF-P V2.4A. Gegenüber der letzten Ausgabe des Handbuchs ergeben sich folgende Änderungen:

Allgemeine Änderungen und Ergänzungen

Das Handbuch wurde in allen Kapiteln überarbeitet. Alle Beispiele wurden auf Ablauffähigkeit überprüft und auf dem Stand BS2000/OSD-BC V7.0 aktualisiert.

Funktionelle Änderungen und Erweiterungen aus SDF-P V2.3A

Funktionen

Name	Operand	Funktionalität, Bemerkung
HOST( )		Funktionsänderung: kann jetzt unabhängig vom Subsystem JV aufgerufen werden
SEARCH-LIST-INDEX( )	DIRECTION=	Neuer Operand: ermöglicht die Vorwärts- und Rückwärtssuche
SESSION-NUMBER( )		Funktionsänderung: kann jetzt unabhängig vom Subsystem JV aufgerufen werden

Kommandos

Name	Operand	Funktionalität, Bemerkung
FREE-VARIABLE	FROM-INDEX=*LAST	Neuer Operandenwert: spezifiziert das letzte Element
	NUMBER-OF-ELEMENTS=*REST	Neuer Operandenwert: spezifiziert die Anzahl der Elemente vom Startwert bis zum letzten Element der Liste
IMPORT-VARIABLE	VARIABLE-NAME=<structured-name 1..20 with-wild(40)> / list-poss(2000): <structured-name 1..20>	Operand erweitert: ermöglicht Musterzeichen im Variablennamen sowie die Angabe mehrerer Variablennamen in Listenform

Name	Operand	Funktionalität, Bemerkung
READ-VARIABLE	INPUT=<file-name>(...) BEGIN-RECORD= END-RECORD= BEGIN-COLUMN= END-COLUMN= PATTERN= PATTERN-TYPE= INPUT=*SYSDTA(...) REMOVE-KEY=	Struktur erweitert um neue Operanden: liest vom Anfangs- bis zum Enddatensatz liest innerhalb des Datensatzes von der Anfangs- bis zur Endspalte selektiert Datensätze, die eine Suchzeichenfolge enthalten gibt an, ob die Suchzeichenfolge ein String oder ein regulärer Ausdruck ist Struktur erhält neuen Operanden: ermöglicht Angabe zur Behandlung des ISAM-Schlüssels falls SYSDTA aus einer ISAM-Datei gelesen wird
SHOW-STRUCTURE-LAYOUT	NAME=<structured-name 1..20 with-wild(40)>	Operand erweitert: ermöglicht Musterzeichen im Namen des Strukturlayouts
SHOW-VARIABLE	OUTPUT=*LIBRARY(...) WRITE-MODE=	Struktur erhält neuen Operanden: gibt an, ob Bibliothekselemente erweitert oder überschrieben werden sollen
SORT-VARIABLE		neues Kommando: sortiert Elemente einer Listenvariablen

Funktionelle Änderungen und Erweiterungen aus SDF-P V2.4A

Funktionen

Name	Operand	Funktionalität, Bemerkung
INDEX( )	BEGIN-COLUMN= END-COLUMN=	neuer Operand: sucht ab dieser Spaltenposition neuer Operand: sucht bis zu dieser Spaltenposition
INTEGER-TO-X-LITERAL		Neue Funktion: konvertiert eine Zahl in eine 4 Byte langen X-String.
X-LITERAL-TO-INTEGER		Neue Funktion: konvertiert einen bis zu 4 Byte langen String in eine Zahl

Kommandos

Name	Operand	Funktionalität, Bemerkung
SELECT-VARIABLE	MESSAGE=	Neuer Operand: ermöglicht die Ausgabe einer Meldung zu Beginn des ersten Selektionbildschirms
SET-VARIABLE	FROM-INDEX=*LAST NUMBER-OF-ELEMENTS=*REST	Neuer Operandenwert: spezifiziert das letzte Element einer Liste Neuer Operandenwert: spezifiziert die Anzahl der Elemente vom Startwert bis zum letzten Element der Liste
SHOW-VARIABLE	VARIABLE-NAME=*LIST(...)	Neuer Operandenwert: ermöglicht die Ausgabe einer Listenvariablen bzw. einzelner Listenelemente
SHOW-VARIABLE-ATTRIBUTES	VARIABLE-NAME=*LIST(...)	Neuer Operandenwert: ermöglicht die Ausgabe der Attribute einer Listenvariablen bzw. einzelner Listenelemente

Readme-Datei

Funktionelle Änderungen und Nachträge zur aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei.
Sie finden die Readme-Datei auf Ihrem BS2000-Rechner unter den Dateinamen SYSRME.*produkt*.*version*.*sprache*, für SDF-P V2.4 also unter SYSRME.SDF-P.024.D .
Die Benutzerkennung, unter der sich die Readme-Datei befindet, erfragen Sie bitte bei Ihrer zuständigen Systembetreuung. Der vollständige Pfadname wird auch durch folgendes Kommando ausgegeben:

```
/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=SDF-P,LOGICAL-ID=SYSRME.D
```

Sie können die Readme-Datei am Bildschirm mit dem Kommando /SHOW-FILE oder einem Editor ansehen oder auf einem Standarddrucker mit folgendem Kommando ausdrucken:

```
/PRINT-DOCUMENT <Pfadname>, LINE-SPACING=*BY-EBCDIC-CONTROL
```

2 Schnelleinstieg SDF-P

2.1 Vorbemerkungen für Umsteiger

Dieser Abschnitt wendet sich an Leser, die bereits über Erfahrungen mit Nicht-S-Prozeduren verfügen, also denjenigen Prozeduren, die mit dem Kommando PROCEDURE oder BEGIN-PROCEDURE anfangen.

S-Prozeduren sind im Wesentlichen als eine Weiterentwicklung des bisherigen Prozedurformats anzusehen, die sich an den Gepflogenheiten höherer Programmiersprachen orientiert. Nicht-S-Prozeduren lassen sich oft ohne größere Änderungen in S-Prozeduren umwandeln (siehe Kapitel „Nicht-S-Prozeduren umstellen“ auf Seite 293ff). Die eigentlichen Vorteile des neuen Prozedurformats lassen sich jedoch erst dann effektiv nutzen, wenn einige in „alten“ Prozeduren häufig verwendete Programmiermethoden umgestellt werden auf die Nutzung der von SDF-P gebotenen neuen Konstrukte. Welche erweiterten Möglichkeiten sich dadurch bieten, zeigt die folgende Tabelle:

Merkmal	**Nicht-S-Prozeduren**	**S-Prozeduren**
Aufbewahrung von Informationen	Prozedurparameter (konstant), Jobvariablen	S-Variablen; Jobvariablen
Kontrollflusssteuerung	bedingte und unbedingte Sprünge, Fehlerbehandlung mit Spinoff und SET-JOB-STEP-Kommando	Bedingungsblöcke, Schleifen, GOTO, Fehlerbehandlung mit Spinoff bzw. Returncode und Fehlerbehandlungsblöcken
Textverarbeitung	Editoraufruf	Stringoperatoren, vordefinierte Funktionen
Arithmetik	Editoraufruf, eigene Programme	Integeroperatoren, vordefinierte Funktionen
Zugriff auf Systemausgaben	SYSOUT-Zuweisung auf Datei mit anschließender Editorbearbeitung	strukturierte Ausgabe in S-Variablen, vordefinierte Funktionen

Merkmal	Nicht-S-Prozeduren	S-Prozeduren
Prüfung der Prozedurintegrität	Aufruf im SDF-Testmodus	Variablen- und Typdeklarartionen, Voranalyse von Kontrollstrukturen beim Aufruf, Aufruf im SDF-Testmodus
Debugging von Prozeduren	– – –	Prozedurtestmodus, Einzelschrittverfolgung

Im später folgenden „Abschnitt „Aufbau einer S-Prozedur“ ab Seite 30 wird eine Beispielprozedur angegeben, die die letzten Zeilen einer Textdatei nach SYSOUT ausgibt, wobei der Dateiname und die Zeilenanzahl wählbar sind. Eine funktional nahezu gleichwertige Prozedur im Nicht-S-Format kann nur unter Zuhilfenahme von TU-Programmen, z.B. des Texteditors EDT, formuliert werden. Um den Kontrast zur S-Prozedur deutlich werden zu lassen, sei hier eine mögliche Realisierung im Nicht-S-Format wiedergegeben:

```
/BEG-PROC LOG=*NO,PROC-PAR=(&DATEI,&ANZAHL=10),ESC-CHAR='&'
/ASS-SYSDTA *SYSCMD
/MOD-JOB-SW ON=(4,5)
/START-EXEC-PROG $EDT
@ SETSW OFF=4-5
@ PRO 1
@ 1.00
@N    Parameter DATEI analysieren:
@N    entweder  vollqual. Dateiname ohne Gen/Vers
@N    oder     '[*lib-elem](bibliothek,element[(vers)][,typ])'
@@CR #S1 = '&DATEI'
@@ON #S1 F '('
@@IF .F. GOTO 3
@ 2.00
@N    Es wurde ein Element angegeben:
@N    Bibliotheksnamen extrahieren
@@#I1 = #I0 + 1
@@#I2 = L #S1
@@CR #S1: #S1:#I1-#I2:
@@ON #S1 D R ')'
@@IF .F. GOTO 4
@@ON #S1 F','
@@IF .F. GOTO 5
@@#I3 = #I0 - 1
@@CR #S2: #S1:1-#I3:
@@#I3 = #I3 + 2
@@#I2 = L #S1
@@CR #S1: #S1:#I3-#I2:
@N    Element einlesen
@@PRO 11
@@CR 1: '@COPY L=', #S2, '(', #S1, ')'
```

```
@@END
@@DO 11
@@IF NO ERRORS: @GOTO 7
@@GOTO 6
@ 3.00
@N    Datei einlesen
@@READ '&DATEI'
@@IF NO ERRORS: @GOTO 7
@@GOTO 6
@ 4.00
@@CR #S1: 'Schliessende Klammer nach Elementangabe fehlt.'
@@P #S1 N
@@GOTO 6
@ 5.00
@@CR #S1: 'Komma fehlt zwischen Bibliotheks- und Elementname.'
@@P #S1 N
@@GOTO 6
@ 6.00
@@CR #S1: 'Fehler beim Einlesen von "&DATEI"!'
@@P #S1 N
@@GOTO 9
@ 7.00
@N    jetzt Zeilennummer bestimmen, ab der die
@N    Ausgabe erfolgen soll (nicht < 1!)
@@#I1 = &ANZAHL
@@IF #I1 < 1 GOTO 9
@@#L1 = $
@@#S1 = C #L1
@@#I2 = S #S1:1-4:
@@IF #I2 > #I1 GOTO 8
@N    alles ausgeben
@@P & N
@@GOTO 9
@ 8.00
@N    Bereich ausgeben
@@#I1 = #I1-1
@@#L1 = $-#I1
@@P #L1.-.$ N
@ 9.00
@@N
@ END
@ DO 1
@N    aufraeumen (sonst kommt Anfrage beim @HALT)
@ DROP ALL
@ HALT
/SET-JOB-STEP
/MOD-JOB-SW OFF=(4,5)
/END-PROC
```

2.2 Was sind S-Prozeduren?

S-Prozeduren sind im einfachsten Falle Folgen gewöhnlicher Kommandos, die sequenziell (in der Reihenfolge der vorgesehenen Ausführung) in einer Textdatei oder einem Bibliothekselement abgelegt sind. Die Kommandos sind mit einem einleitenden Schrägstrich zu schreiben. Anweisungen an Programme können an geeigneten Stellen zwischen die Kommandos eingefügt werden und müssen mit zwei Schrägstrichen beginnen; ebenso können Datenzeilen (für Eingaben von SYSDTA) vorhanden sein, die daran erkannt werden, dass sie in der ersten Spalte keinen Schrägstrich enthalten.

Zur gezielten Steuerung der Verarbeitung, zur Modifikation von Kommandos und zur Beeinflussung der Bearbeitungsreihenfolge (also für Zwecke der eigentlichen Prozedurprogrammierung) können zusätzliche, SDF-P-spezifische Kommandos und Sonderzeichenfolgen vorhanden sein; diese sind aber nicht zwingend erforderlich. Die folgenden Zeilen stellen beispielsweise eine vollständige S-Prozedur dar:

```
/CREATE-FILE #HELLO, SUPPRESS-ERRORS=*FILE-EXIST
/WRITE-TEXT 'Hello, world!'
```

Die Prozedur kann auch mehrmals nacheinander aufgerufen werden. Zum Aufruf kann das Kommando CALL-PROCEDURE verwendet werden, in dem nur der Name der Prozedurdatei angegeben werden muss.

2.3 Schreibregeln für S-Prozeduren

2.3.1 Groß- und Kleinschreibung

Groß- und Kleinschreibung werden in Kommandos nur innerhalb von Operandenwerten unterschieden, bei denen dies für die Kommandoausführung relevant ist; so werden im obigen Beispiel die Groß- und Kleinbuchstaben vom WRITE-TEXT-Kommando unverändert wieder ausgegeben. Dennoch werden in diesem Handbuch für die Darstellung von Kommandos vorwiegend Großbuchstaben verwendet, die auch traditionell von allen Editoren, Terminals und Ausgabegeräten problemlos unterstützt werden.

Kleinbuchstaben bezeichnen in Kommandodefinitionen oft variable Bestandteile, die im Anwendungsfall durch geeignete aktuelle Werte zu ersetzen sind (siehe Hinweise zur Metasyntax ab Seite 548).

2.3.2 Zeilenformat

In vielen Fällen wird in jeweils einer Textzeile ein Kommando stehen; die maximale Zeilenlänge beträgt fast 4 KB und dürfte damit auch für umfangreiche Kommandos ausreichen.

Da jedoch die Zeilen nur bis zu einer gewissen Länge (meist 72 oder 80 Zeichen) am Bildschirm günstig zu lesen und zu editieren sind, werden lange Kommandos meist auf mehrere Zeilen verteilt. Wenn ein Kommando in einer Textzeile nicht abgeschlossen wird, muss diese Zeile mit einem Bindestrich enden. Die folgende Zeile beginnt wieder mit einem Schrägstrich, an den sich die Fortsetzung des Kommandos unmittelbar anschließt. Um übersichtliche Einrückungen der Folgezeilen zu ermöglichen, ohne dass im Ablaufprotokoll entsprechend viele Leerzeichen erscheinen, können in einer Folgezeile auch vor dem ersten Schrägstrich Leerzeichen stehen, die nicht Bestandteil des Kommandos werden.

Insbesondere bei Kommandos mit verschachtelten Operandenstrukturen bietet es sich an, Fortsetzungszeilen und Einrückungen zu verwenden, um die Lesbarkeit zu verbessern:

```
/ MODIFY-FILE-ATTRIBUTES -
        /FILE-NAME = INFO.TXT, -
        /PROTECTION = *PARAMETERS(-
               /ACCESS = *READ, -
               /USER-ACCESS = *ALL-USERS), -
        /MIGRATE = *ALLOWED
```

Andererseits ist es auch zulässig, mehrere Kommandos durch Semikolon voneinander getrennt in eine Zeile zu schreiben:

```
/ IF (MODE == 'DEBUG'); WRITE-TEXT 'Datei &A pruefen'; END-IF
```

2.3.3 Kommentare

Kommentare können als beliebiger Text, der in doppelte Anführungszeichen eingeschlossen ist, in Kommandos eingefügt werden. Sie sind lediglich innerhalb von Namen, Schlüsselwörtern, Operatoren, Zahlen oder Zeichenkonstanten aus syntaktischen Gründen nicht zugelassen. Ein Kommentar darf auch in einer ansonsten leeren Kommandozeile stehen:

```
/   "Vorbereitungen fuer den Aufruf der Prozedur P.YY"
/COPY-FILE ALTE-DATEI "erstellt vom Programm A.23" , -
/          NEUE-DATEI "wird in Prozedur P.YY gebraucht"
```

Außerdem wird bei einem Auftreten der Zeichenfolge &* der weitere Inhalt der betreffenden Zeile ignoriert („Zeilenendekommentar“). Dadurch können Kommandozeilen (vorübergehend) unwirksam gemacht werden, ohne dass darin enthaltene Anführungszeichen besonders behandelt werden müssen:

```
/     START-EXE YZ-ERZEUGEN
/ &*  CALL-PROC P.UMKODIEREN,(YZ-DATEI) "Umlaute aendern"
/     START-PERCON
//    ASSIGN-INPUT-FILE DISK-FILE(...)
```

2.4 S-Prozeduren erstellen

2.4.1 Aufbau einer S-Prozedur

Wenn eine Prozedur von den Möglichkeiten der Programmierung mit SDF-P Gebrauch machen soll, sind über die reinen Ausführungskommandos hinaus weitere Deklarations- und Steuerkommandos erforderlich. Dies soll anhand eines Beispiels anschaulich gemacht werden; die Bedeutung der einzelnen Kommandos wird anschließend schrittweise erläutert.

Die folgende Beispielprozedur dient dazu, die letzten (standardmäßig zehn) Zeilen einer Textdatei nach SYSOUT auszugeben:

```
/SET-PROC-OPT IMPLICIT-DECLARATION=*NO
/BEG-PARAM-DECL
/  DECL-PARAM DATEI(TYPE=*STRING)
/  DECL-PARAM ANZAHL-ZEILEN(TYPE=*INTEGER,INIT=10)
/END-PARAM-DECL
/
/"Als DATEI wird ein vollqualifizierter Dateiname (ohne  "
/"Generation/Version) oder eine Elementangabe der Form   "
/" '[*LIBRARY-ELEMENT](library,element[(version)],type)' "
/"erwartet                                               "
/
/"Benoetigte Hilfsvariablen:"
/DECL-VAR I(TYPE=*INTEGER)
/DECL-VAR DATEN(TYPE=*STRING),MULTIPLE-ELEMENTS=*LIST
/DECL-VAR BIB(TYPE=*STRING)
/
/"Pruefung, ob die Datei/das Element existiert,"
/"damit eine gezielte Fehlermeldung ausgegeben werden kann"
/I = INDEX(DATEI,'(')
/IF (I > 0)   "Name enthaelt '(', also Bibliothek"
/  BIB = SUBLIST(SUBSTR(DATEI,I),1)   "Bibliotheksname"
/  IF NOT IS-CATALOGED-FILE(BIB)
/    WRITE-TEXT 'Bibliothek &BIB existiert nicht!'
/    EXIT-PROCEDURE ERROR=*YES
/  END-IF
/  IF NOT IS-LIBRARY(BIB)
/    WRITE-TEXT 'Datei &BIB ist keine Bibliothek!'
/    EXIT-PROCEDURE ERROR=*YES
/  END-IF
/ELSE-IF NOT IS-CATALOGED-FILE(DATEI)
/  WRITE-TEXT 'Datei &DATEI existiert nicht!'
/  EXIT-PROCEDURE ERROR=*YES
/END-IF
```

```
/
/"Daten in eine Variablenliste einlesen"
/READ-VAR *LIST(DATEN),STRING-QUOTES=*NO,INPUT=&DATEI
/I = SIZE('DATEN') - ANZAHL-ZEILEN + 1 "Startposition berechnen"
/IF (I < 1); I = 1; END-IF          "nicht vor die erste Zeile!"
/FOR I = *COUNTER( FROM = I, TO = SIZE('DATEN'), INCREMENT = 1 )
/  WRITE-TEXT &(TO-C-LITERAL(DATEN#I))
/END-FOR
```

2.4.2 Prozedurkopf

Die ersten fünf Zeilen im letzten Beispiel bilden den Prozedurkopf. Er beginnt mit dem Kommando SET-PROCEDURE-OPTIONS, in dem globale Festlegungen über das Prozedurformat getroffen und wichtige Prozedureigenschaften voreingestellt werden. Einige dieser Prozedureigenschaften können im weiteren Verlauf der Prozedur noch durch das Kommando MODIFY-PROCEDURE-OPTIONS dynamisch geändert werden. Die Angabe IMPLICIT-DECLARATIONS=*NO verhindert, dass Variablen (unbeabsichtigt) allein dadurch entstehen, dass ihnen ein Wert zugewiesen wird. Damit lassen sich Schreibfehler in Variablennamen besser erkennen.

Deklaration von Prozedurparametern

Prozedurparameter sind spezielle Variablen, denen beim Aufruf der Prozedur, beispielsweise durch das Kommando CALL-PROCEDURE, veränderliche Werte zugewiesen werden können. Ihre Deklaration ist ebenfalls Bestandteil des Prozedurkopfs: DECLARE-PARAMETER-Kommandos vereinbaren für jeden Prozedurparameter den Namen, einen Typ, den Defaultwert und die Art der Wertübergabe. Wenn mehrere DECLARE-PARAMETER-Kommandos vorhanden sind, müssen sie zwischen BEGIN- und END-PARAMETER-DECLARATION eingeschlossen werden.

Die Beispielprozedur deklariert die beiden Parameter DATEI und ANZAHL-ZEILEN. Letzterer kann nur ganzzahlige Werte annehmem (TYPE=*INTEGER); wenn er beim Aufruf der Prozedur nicht mit einem aktuellen Wert versorgt wird, gilt die Vorbelegung (INITIAL-VALUE) mit dem Wert 10. Dem Parameter DATEI hingegen muss beim Aufruf eine Zeichenfolge (TYPE=*STRING) zugewiesen werden, da kein INITIAL-VALUE vorhanden ist.

Beim Aufruf der Prozedur können die aktuellen Parameterwerte den Parametervariablen zugeordnet werden, indem (beispielsweise im Kommando CALL-PROCEDURE) die Werte in der Reihenfolge der Parameterdeklarationen spezifiziert werden. Alternativ kann die Zuordnung auch über die Parameternamen erfolgen, wobei Abkürzungen erlaubt sind, solange die Eindeutigkeit gewahrt bleibt:

```
/CALL-PROCEDURE P.SHOW-TAIL,(DATEI=PROTO.L,ANZAHL-ZEILEN=20)
/CALL-PROCEDURE P.SHOW-TAIL,(PROTO.L,20)
/CALL-PROCEDURE P.SHOW-TAIL,(A-Z=20,D=PROTO.L)
/CALL-PROCEDURE P.SHOW-TAIL,(PROTO.L)
```

Die ersten drei Kommandos sind gleichwertig, da DATEI der erste und ANZAHL-ZEILEN der zweite deklarierte Parameter ist und D bzw. A-Z gültige Abkürzungen für die Parameternamen darstellen. Im letzten Aufruf wird für ANZAHL-ZEILEN der Standardwert (INITIAL-VALUE) von 10 angenommen, da kein Wert für diesen Parameter spezifiziert wurde.

Wenn bei der Parameterübergabe oder bei der Verarbeitung des Prozedurkopfes ein Fehler auftritt, wird mit der Ausführung der Prozedur gar nicht erst begonnen. Ein solcher Fehler kann nur vom Aufrufer, nicht von der gerufenen Prozedur behandelt werden. Dies gilt auch für den Fall, dass ein Prozedurparameter weder durch explizite Angabe im Aufrufkommando noch durch einen INITIAL-VALUE einen Wert erhält.

Alle Kommandos zur Parameterdeklaration können entfallen, wenn eine Prozedur keine Parameter erwartet. Ebenso kann auf das Kommando SET-PROCEDURE-OPTIONS verzichtet werden, wenn die voreingestellten Werte gelten sollen, sodass der Prozedurkopf im einfachsten Fall gar nicht vorhanden zu sein braucht.

2.4.3 Prozedurrumpf

Der auf den Prozedurkopf folgende Teil einer S-Prozedur wird als Prozedurrumpf bezeichnet. Er kann normale SDF- (oder auch ISP-) Kommandos enthalten, die über die sog. &-Ersetzung beeinflusst werden können: z.B. durch die aktuellen Werte der Prozedurparameter, durch die aktuellen Werte weiterer Variablen und durch Zugriffe auf Umgebungsinformationen (wie z.B. die Benutzerkennung oder das Tagesdatum). Außerdem können SDF-P-Kontrollflusskommandos auftreten, durch die die Reihenfolge der Kommandobearbeitung geändert werden kann in Abhängigkeit von Variablen- und Funktionswerten.

2.4.4 S-Variablen

Neben den Prozedurparametern, die beim Aufruf der Prozedur erzeugt werden und einen Wert erhalten, gibt es die sog. S-Variablen, die an beliebiger Stelle innerhalb einer Prozedur erzeugt werden können. Dies geschieht explizit durch das Kommando DECLARE-VARIABLE, implizit durch die erste Zuweisung eines Wertes an die Variable (sofern dies nicht durch die Option IMPLICIT-DECLARATION=*NO im Kommando SET-PROCEDURE-OPTIONS unterbunden wurde) oder automatisch während der Ausführung einiger spezieller Kommandos (wie z.B. OPEN-VARIABLE-CONTAINER oder EXECUTE-CMD).

Bei expliziter Deklaration kann, ähnlich wie bei Parametern, für die Variablen ein Typ und auch ein Anfangswert (INITIAL-VALUE) vereinbart werden.

Die Namen (einfacher) S-Variablen dürfen bis zu 20 Zeichen lang sein und gemäß den Regeln für den SDF-Datentyp <structured-name> aus Buchstaben, Ziffern und Bindestrichen bestehen. Die mit SYS beginnenden Namen sind der Systemsoftware vorbehalten; einige weitere Namen sind für Rechenoperatoren und boolesche Konstanten reserviert.

Wertzuweisung

Der Wert einer Variablen kann durch verschiedene Kommandos und auch durch Programme verändert werden. Dies gilt übrigens auch für Prozedurparameter, die während der Ausführung der Prozedur wie normale S-Variablen behandelt werden.

Insbesondere ist das Kommando SET-VARIABLE dazu vorgesehen, einer Variablen einen neuen Wert zuzuweisen. Da der Name dieses Kommandos nicht nur nach den üblichen Regeln abgekürzt, sondern auch ganz weggelassen werden darf, lassen sich Zuweisungen an Variable in der von Programmiersprachen her bekannten Form schreiben:

```
/ variablenname = neuer-wert
```

Konstanten

Die Werte, die den Variablen zugewiesen werden, können unter anderem konstante Zeichenfolgen, Zahlen oder boolesche (Wahrheits-) Werte sein.

Konstante Zeichenfolgen (bezeichnet als STRING-Typ) werden in Hochkommata (einfache Anführungszeichen) eingeschlossen, wobei jedes darin enthaltene Hochkomma und jedes &-Zeichen zu verdoppeln ist. Groß- und Kleinbuchstaben werden unterschieden.

Alternativ können Zeichenfolgen auch in hexadezimaler Schreibweise (als sog. X-Strings) angegeben werden, um die Darstellung nichtabdruckbarer Zeichen zu ermöglichen. Die beiden folgenden Kommandos sind daher gleichwertig:

```
/ DATEI = 'ARB.4'
/ DATEI = X'C1D9C24BF4'
```

Ganze Zahlen (Typ INTEGER) können als dezimale Konstanten aus dem Wertebereich -2^{31} .. $+2^{31}-1$ dargestellt werden; das positive Vorzeichen ist optional.

Boolesche Konstanten (Typ BOOLEAN) werden durch die Schlüsselwörter TRUE bzw. FALSE bezeichnet. Alternativ können auch die Namen YES und ON bzw. NO und OFF benutzt werden.

Variablentypen

Eine Variable nimmt nach einer Wertzuweisung immer den Typ des zugewiesenen Wertes an (also STRING, INTEGER oder BOOLEAN). Um fehlerhafte Zuweisungen frühzeitig zu erkennen und damit die Ablaufsicherheit der Prozeduren zu verbessern, kann bei der Deklaration von Variablen mit dem TYPE-Operanden festgelegt werden, dass die Variable nur Werte eines bestimmten Typs annehmen darf; andere Zuweisungen führen dann zu einem Fehler.

Geltungsbereiche

Eine Variable existiert normalerweise nur innerhalb der Prozedur, in der sie (explizit oder implizit) deklariert wurde. Wenn die Prozedur beendet wird, endet auch die Lebensdauer der Variablen. Wenn mehrere Aufrufe derselben oder auch unterschiedlicher Prozeduren innerhalb einer Task mit derselben Variable arbeiten und diese jeweils ihren Wert weiterhin behalten soll, kann durch eine explizite Deklaration mit der Angabe SCOPE=*TASK eine sog. Taskvariable eingerichtet werden, die von verschiedenen Prozeduren aus benutzt werden kann.

Wenn aus einer Prozedur heraus eine weitere Prozedur aufgerufen wird und beide mit Variablen gleichen Namens arbeiten, so werden diese Variablen grundsätzlich völlig getrennt voneinander angelegt und verwaltet. Nur wenn eine Prozedur über das Kommando INCLUDE-PROCEDURE aufgerufen wird, hat sie die Möglichkeit, mit der Angabe SCOPE= *PROCEDURE in Deklarationen auf den Variablenbereich der rufenden Prozedur zuzugreifen und dort beispielsweise Variablen anzulegen oder zu modifizieren.

Variablencontainer

Normalerweise werden die Definitionen und die Werte der Variablen im privilegierten Speicher der Task aufbewahrt und am Ende der Lebensdauer der Variablen gelöscht. Es gibt jedoch zwei Möglichkeiten, andere Speichermedien zu nutzen: Durch eine Deklaration mit dem Operanden CONTAINER=*JV kann der Variablenwert in einer Jobvariablen abgelegt werden; er bleibt damit auch längerfristig erhalten und kann zudem durch JV-Zugriffe auch von anderen Tasks her abgefragt und verändert werden.

Gibt man bei der Deklaration einer Variablen einen Container an, der in einer Bibliothek angelegt wurde (mit dem Kommando OPEN-VARIABLE-CONTAINER), so werden sowohl die Beschreibung der Variablen als auch ihr Wert auf Anforderung durch das Kommando SAVE-VARIABLE-CONTAINER oder auf Wunsch automatisch bei Beendigung der Prozedur bzw. der Task in der Bibliothek gespeichert und können in einem anderen Prozedurlauf erneut verwendet werden.

2.4.5 Ausdrücke

Neben der Darstellung von Werten als Konstanten des Typs STRING, INTEGER oder BOOLEAN können in Zuweisungen auch bereits existierende Variablenwerte und die Ergebnisse von Funktionsaufrufen verwendet und in verschiedenen Operationen untereinander sowie mit Konstanten verknüpft werden. Solche komplexen Darstellungen werden als Ausdrücke bezeichnet und ähneln auch bezüglich der Schreibweise den entsprechenden Konstrukten höherer Programmiersprachen.
Jeder Ausdruck liefert ein Ergebnis, das einem der drei Basistypen angehört. Im einfachsten Fall besteht ein Ausdruck nur aus einer Konstanten, dem Namen einer Variablen (die einen einfachen Wert besitzen muss) oder dem Aufruf einer Funktion. Darüberhinaus

können diese sog. Basisterme durch Operatoren miteinander verknüpft werden, wobei für jeden Operator festgelegt ist, welche Typen seine Operanden haben dürfen und welchen Ergebnistyp er liefert.

Ausdrücke können folgende Operatoren enthalten:

arithmetische Operatoren:	+, –, *, /, MOD
Vergleichsoperatoren:	<, >, <=, >=, ==, <>
bzw. (gleichwertig Operatoren):	LT, GT, LE, GE, EQ, NE
logische Operatoren:	NOT, OR, AND, XOR
Stringoperator (Verkettung):	//

Eine genauere Beschreibung der einzelnen Operatoren findet sich ab Seite 262.

Bei den Vergleichsoperatoren ist zu beachten, dass beide Operanden vom selben Typ sein müssen (beide STRING, INTEGER oder BOOLEAN; für Letztere sind nur Abfragen auf Gleichheit oder Ungleichheit zulässig). Dementsprechend wird der Vergleich typgerecht ausgeführt: Der Ausdruck 12 > 9 liefert den Wert TRUE, der Stringvergleich '12' > '9' hingegen FALSE, weil Strings von links nach rechts zeichenweise verglichen werden.

Da der Bindestrich sowohl innerhalb von Variablennamen als auch in Ausdrücken als arithmetischer Operator (Minuszeichen) auftreten kann, muss er als Operator in Leerzeichen eingeschlossen werden, falls er sonst als Namensbestandteil interpretiert werden könnte. Beispielsweise subtrahiert das Kommando

```
/ I = I - 1
```

die Konstante 1 von der Variablen I und weist das Ergebnis derselben Variablen als neuen Wert zu. Das Kommando

```
/ I = I-1
```

dagegen nimmt Bezug auf eine Variable I-1 und weist deren Wert der Variablen I zu.

Natürlich müssen auch die aus Buchstabenfolgen bestehenden Operatoren wie MOD, EQ oder AND durch Leerzeichen von benachbarten Namen oder Zahlen getrennt werden. Es empfiehlt sich generell, alle Operatoren im Interesse einer besseren Lesbarkeit in Leerzeichen einzuschließen, auch wenn die Syntaxanalyse dies nicht immer erfordert.

Ausdrücke dürfen, außer in den Zuweisungen durch SET-VARIABLE, auch als Operanden einiger anderer (insbesondere der in diesem Handbuch beschriebenen) Kommandos benutzt werden.

2.4.6 Funktionsaufrufe

Während eine Variable bei jedem Ansprechen den zuletzt abgespeicherten Wert unverändert wieder zurückliefert, läuft beim Aufruf einer Funktion ein SDF-P-interner Algorithmus ab, der schließlich einen Ergebniswert bereitstellt. Je nach Verwendungszweck der Funktion wird dabei auf unterschiedliche Informationen zurückgegriffen: Die Funktionen zur Stringbearbeitung und zur Datenkonvertierung berechnen einen Ergebniswert aus den beim Aufruf mitgegebenen Parametern; andere Funktionen liefern Informationen über die Ablaufumgebung (z.B. Benutzerkennung, Name der Datenstation, Datum, Uhrzeit, Eigenschaften von Dateien) oder die Returncodes vorangegangener Kommandoaufrufe, die sie intern mithilfe von Betriebssystemaufrufen ermitteln.

Innerhalb eines Ausdrucks wird eine Funktion über ihren Namen angesprochen, dem eine in Klammern eingeschlossene Parameterliste folgt. Diese kann leer sein, wenn die Funktion keine Parameter benötigt (wie z.B. die Funktion TSN( ), die die TSN des aktuellen Auftrags liefert) oder wenn die voreingestellten Parameterwerte benutzt werden. Beispielsweise liefert der Aufruf DATE( ) ebenso wie DATE(FORMAT=*ISO) das Tagesdatum im ISO-Format; wird das deutsche Datumsformat gewünscht, so kann dies im Aufruf DATE( ) mit FORMAT=*GERMAN explizit verlangt werden. Im Falle einer leeren Parameterliste können auch die Klammern weggelassen werden, sofern der Funktionsname nicht mit einem Variablennamen kollidiert.

Die Funktionsparameter werden in ähnlicher Weise spezifiziert wie die Operanden in SDF-Kommandos: Sie können wahlweise durch ihre Reihenfolge oder durch ihren Namen angesprochen werden, und die Namen sind abkürzbar, solange die Eindeutigkeit gewahrt bleibt. Die Parameterwerte können Schlüsselwörter sein (mit vorangestelltem Stern) oder Ausdrücke vom Typ STRING, INTEGER oder BOOLEAN. Dabei ist zu beachten, dass Variablennamen bei der Übergabe an die Funktion durch ihre Werte ersetzt werden; soll die Funktion auf den „Namen“ der Variablen zugreifen (um z.B. zu ermitteln, ob die Variable definiert ist), muss der Name als String-Konstante in Hochkommata eingeschlossen werden. Beispiel:

```
/ I = INDEX(DATEI,'(')
/ B = IS-INITIALIZED('AUSGABE')
```

Der erste Aufruf greift auf den Inhalt der Stringvariablen DATEI zu und sucht nach dem ersten Auftreten einer öffnenden Klammer; deren Position wird der Variablen I zugewiesen (oder Null, wenn keine gefunden wurde). Das zweite Kommando prüft, ob die Variable AUSGABE überhaupt einen gültigen Wert enthält und weist in Abhängigkeit davon der Variablen B den Wert TRUE oder FALSE zu.

Neben den vordefinierten Funktionen, die zum Lieferumfang von SDF-P gehören, gibt es Systemverwalter-Funktionen, die bei Bedarf von der Systembetreuung implementiert und allen Anwendern bereitgestellt werden können.

Die Beschreibung aller vordefinierten Funktionen und ihrer Parameter findet sich ab Seite 353. Eine detaillierte Darstellung der Schreibweise von Funktionsparametern enthält der Abschnitt „Eingabeparameter beim Funktionsaufruf“ auf Seite 231ff.

2.4.7 Bedingungen und Schleifen

Der Ablauf einer S-Prozedur kann durch Kontrollflusskommandos gesteuert werden, mit denen – analog zu höheren Programmiersprachen – bedingt auszuführende Kommandosequenzen (IF-Blöcke) und Schleifen (WHILE-, REPEAT-UNTIL- und FOR-Blöcke) gebildet werden können. Anfang und Ende dieser bedingten oder zu wiederholenden Kommandofolgen werden durch je ein besonderes Kommando gekennzeichnet:

```
/IF (bedingung)
/  "then-Kommandofolge"
/END-IF

/WHILE (bedingung)
/  "Schleifenrumpf"
/END-WHILE

/REPEAT
/  "Schleifenrumpf, wird mindestens einmal durchlaufen"
/UNTIL (bedingung)

/FOR variable = werteliste, CONDITION = (bedingung)
/  "Schleifenrumpf"
/END-FOR
```

Die Kontrollstrukturen können daher beliebig und eindeutig ineinander verschachtelt werden. Der IF-Block kann noch zusätzliche Abfragen weiterer Bedingungen und/oder einen ELSE-Zweig enthalten:

```
/IF (bedingung-1)
/  "Kommandos, die ausgefuehrt werden ..."
/  "... wenn bedingung-1 TRUE ist"
/ELSE-IF (bedingung-2)
/  "... wenn bedingung-1 FALSE, aber bedingung-2 TRUE ist"
/  "    (beliebig viele weitere ELSE-IF sind moeglich"
/ELSE
/  "... wenn keine der Bedingungen erfuellt war"
/END-IF
```

Der Deutlichkeit halber können Anfang und Ende eines Blocks durch Marken gekennzeichnet werden, deren Übereinstimmung von SDF-P geprüft wird:

```
/&* Diese Kommandofolge entfernt aus einer Stringvariablen TEXT
/&* alle nachlaufenden Leerzeichen. TEXT sollte wenigstens ein
/&* Zeichen ungleich ' ' enthalten.
/BLANKS-AB: WHILE (SUBSTR(TEXT,LENGTH(TEXT),1) == ' ')
/  TEXT = SUBSTR(TEXT,1,LENGTH(TEXT) - 1)  "um ein Byte kuerzen"
/END-WHILE BLANKS-AB
```

Auf eine solche Marke kann auch mit dem Kommando EXIT-BLOCK Bezug genommen werden, wenn ein umschließender Block vorzeitig verlassen werden soll. Ebenfalls durch Angabe der Marke einer umschließenden Schleife lässt sich mit dem Kommando CYCLE der nächste Schleifendurchlauf anstoßen, bevor der eigentliche Schleifenrumpf bis zum Ende abgearbeitet ist.

Ähnlich einer Schleife fasst ein so genannter BEGIN-Block eine Kommandofolge zusammen, jedoch hat er auf die Abarbeitungsreihenfolge keinen Einfluss. Er kann beispielsweise im Zusammenwirken mit EXIT-BLOCK dazu dienen, einen Prozedurabschnitt unter gewissen Bedingungen vorzeitig zu verlassen:

```
/TAGESARBEIT: BEGIN-BLOCK
/  "Arbeiten, die an jedem Tag auszufuehren sind"
/  IF (DAY() == 'SAT' OR DAY() == 'SUN')
/    EXIT-BLOCK TAGESARBEIT
/  END-IF
/  "Arbeiten, die nur werktags auszufuehren sind"
/  IF (SUBSTR(DATE(),9,2) <> '01')
/    EXIT-BLOCK TAGESARBEIT
/  END-IF
/  "Arbeiten, die nur am Monatsersten auszufuehren sind"
/END-BLOCK TAGESARBEIT
```

Hierdurch kann sich ein gegenüber zahlreichen verschachtelten IF-Blöcken übersichtlicherer Prozeduraufbau ergeben.

Für diejenigen Programmierer, die entgegen seinem schlechten Ruf auf ein GOTO nicht verzichten möchten, ist auch dieses Kommando implementiert; als Sprungziel ist eine Marke anzugeben, die dem Zielkommando vorangestellt wird:

```
/IF (...); GOTO AUFRAEUMEN; END-IF
/"weitere Kommandos"
/AUFRAEUMEN: DELETE-FILE ...
```

Es gelten die üblichen Einschränkungen (z.B. kein GOTO von außen in einen Block hinein).

2.4.8 &-Ersetzung

Auch Kommandos, die keine Ausdrücke als Operandenwerte zulassen, können durch Variablenwerte modifiziert werden. In S-Prozeduren werden an nahezu beliebiger Stelle (bei einigen Kommandos, z.B. SDF-P-Kontrollflusskommandos, bestehen Einschränkungen) im Bereich der Kommandooperanden und auch des Kommandonamens selbst SDF-P-Ausdrücke durch ihren Wert ersetzt, bevor die eigentliche Analyse des Kommandos beginnt. Hierzu ist an der entsprechenden Stelle im Kommando der Ausdruck in Klammern einzuschließen und das Fluchtsymbol & voranzustellen. Falls der Ausdruckswert vom Typ INTEGER oder BOOLEAN ist, wird er zuvor in einen String konvertiert. Im einfachsten Fall besteht der Ausdruck nur aus dem Namen einer einfachen Variablen; die Klammern können dann entfallen, und der Variablenname ist, wenn ihm nicht ohnehin ein Leer- oder Sonderzeichen folgt, durch einen Punkt vom nachfolgenden Text zu trennen:

```
/ZEIT = 40
/ENTER-JOB E.TEST1,CPU-LIMIT=&ZEIT           "Limit = 40 Sekunden"
/ENTER-JOB E.TEST2,CPU-LIMIT=&(ZEIT * 60)    "Limit = 40 Minuten"
/ENTER-JOB E.TEST3,JOB-CLASS=JC&ZEIT.MAX     "Jobklasse = JC40MAX"
```

Die Einfügung von Jobvariablenwerten, für die in Nicht-S-Prozeduren eine ähnliche Syntax gilt, geschieht in S-Prozeduren mittels Aufruf der vordefinierten Funktion JV:

```
/                                &* pruefe Folgendes!
/SHOW-JV JV-CONTENTS=$USERXY.TSN
/&*  die JV soll die TSN eines laufenden Auftrags enthalten
/SHOW-JOB-STATUS TSN=&(JV('$USERXY.TSN'))
```

In den meisten Fällen wird allerdings die Verwendung der Jobvariablen als Container für eine S-Variable einfacher sein:

```
/DECL-VAR DB-TSN(TYPE=STRING),CONTAINER=*JV($USERXY.TSN)
/  "..."
/SHOW-JOB-STATUS TSN=&DB-TSN
```

Es ist zu beachten, dass die &-Ersetzung nicht rekursiv ausgeführt wird, d.h., wenn sie wiederum ein Fluchtsymbol & erzeugt, löst dieses keine erneute Ersetzung aus.

2.4.9 Arrays und Listen

Neben den einfachen Variablen, die jeweils genau einen Wert vom Typ STRING, INTEGER oder BOOLEAN enthalten können, gibt es auch Variablen, die eine größere Zahl gleichartiger Werte aufnehmen können. Hierzu gehören die Arrays; sie bieten eine (vorgebbare) Anzahl von Speicherplätzen, die wahlfrei durch Angabe eines Indexwertes angesprochen werden können:

```
/DECLARE-VARIABLE KOSTEN(TYPE=*INTEGER), -
/                 MULTIPLE-ELEMENTS = *ARRAY(1990,2099)
/KOSTEN#1996 = 4090 "Cents je Stunde"
/JAHR = 1996
/WHILE (JAHR < 2005)    "Schleife ueber 10 Jahre"
/  NEUE-KOSTEN = (KOSTEN#JAHR * 105 + 50) / 100    "+ 5% pro Jahr"
/  JAHR = JAHR + 1
/  KOSTEN#JAHR = NEUE-KOSTEN
/END-WHILE
```

Der Indexwert wird als Zahl oder als Name einer einfachen Integervariablen durch das Zeichen # mit dem Variablennamen verbunden. Innerhalb des zulässigen Indexbereichs, der bei der Deklaration festgelegt wird, können beliebige Arrayelemente unabhängig voneinander erzeugt, angesprochen und auch wieder gelöscht werden (hierzu s.a. das Kommando FREE-VARIABLE).

Listen hingegen bestehen immer aus einer Reihe von Werten, die fortlaufend von 1 an nummeriert sind. Es können nur Werte am Ende oder am Anfang der Liste eingefügt werden; im letzteren Fall werden die Indizes aller existierenden Werte automatisch um 1 erhöht. Da die lückenlose Aufeinanderfolge der Listenelemente stets sichergestellt ist, können Listen besonders einfach in Schleifen abgearbeitet werden (FOR-Kommando). Die Kommandos READ-VARIABLE und SHOW-VARIABLE unterstützen zudem die satzweise Übertragung von Dateiinhalten in bzw. aus Listen. Die folgende Beispielprozedur schreibt die Sätze einer Datei in umgekehrter Reihenfolge in eine Ausgabedatei und bedient sich hierzu zweier Listenvariablen:

```
/DECL-PARAM (VON,NACH)
/DECL-VAR (EIN,AUS),MULT-ELEM=*LIST
/READ-VAR *LIST(EIN),STR-QUOTES=*NO,INPUT=&VON
/FOR ZEILE=*LIST(EIN)
/  AUS = ZEILE, WRITE-MODE=*PREFIX
/END-FOR
/SHOW-VAR AUS,PREFIX=*NO,OUTPUT=&NACH    "###"
```

Eine Sonderrolle nimmt das vordefinierte Array SYSSWITCH ein. Seine 32 Elemente (SYSSWITCH#0 bis SYSSWITCH#31) vom Typ BOOLEAN werden intern auf die Auftragsschalter des aktuellen Auftrags abgebildet, die auf diese Weise unmittelbar gesetzt und abgefragt werden können:

```
/SW4-VORHER = SYSSWITCH#4      "###"
/SYSSWITCH#4 = TRUE
/START-EXECUTABLE-PROGRAM $EDT
@...
@HALT
/SYSSWITCH#4 = SW4-VORHER
```

2.4.10 Strukturen

Anders als Arrays und Listen, die eine Anzahl von Werten gleichen Typs unter einem Variablennamen vereinigen, ermöglicht eine Variable vom Typ STRUCTURE eine Zusammenfassung mehrerer Elemente beliebigen Typs, von denen jedes durch einen eigenen Namen gekennzeichnet ist. Diese Elementnamen werden, durch einen Punkt getrennt, an den Namen der Strukturvariablen angehängt, um die einzelnen Elemente anzusprechen:

```
/DECL-VAR EINKAUF(TYPE=*STRUCTURE)
/EINKAUF.LEITER = 'H. Haegar'
/EINKAUF.ANZ-MITARBEITER = 17
/EINKAUF.UMSATZ = 99000
```

Die Strukturelemente können ihrerseits wieder Arrays oder Listen und auch vom Typ STRUCTURE sein:

```
/DECL-ELEM EINKAUF.PERSONAL(TYPE=*STRUCTURE),MULT-ELEM=*LIST
/EINKAUF.PERSONAL#1.NAME = 'S. Glueckspilz'
/EINKAUF.PERSONAL#1.PERS-NR = 13
/EINKAUF.PERSONAL#2.NAME = 'D. Harry'
/EINKAUF.PERSONAL#2.PERS-NR = 17
```

Das folgende Beispiel zeigt, wie eine Liste von Strukturen als eine einfache Datenbank benutzt werden kann. Zur Erzeugung der Elemente und zur Zuweisung von Werten wird hier eine spezielle Form des SET-VARIABLE-Kommandos benutzt, die einen String im Format einer SDF-Kommando-Unterstruktur erwartet und daraus einzelne Wertzuweisungen generiert. Mit FOR-Schleifen werden die Einträge der „Datenbank“ der Reihe nach abgearbeitet, um dem Benutzer die Eingabealternativen zu präsentieren und danach den gewünschten Eintrag zu selektieren.

```
/DECL-VAR JOB-START(TYPE=*STRUCT),MULT-ELEM=*LIST
/DECL-VAR JOB(TYPE=STRUCT)
/
/&*     Liste der Jobdefinitionen aufbauen
/JOB-START=*STR-TO-VAR('PAR(NAME=CTL,CPU=60,FILE=E.CONTROL)'), -
/          WRITE-MODE=*EXTEND
/JOB-START=*STR-TO-VAR('PAR(NAME=DB,CPU=999,FILE=$AV.DB-START)'), -
/          WRITE-MODE=*EXTEND
/"... beliebig viele weitere Jobdefinitionen nach Bedarf ..."
/
/&*     alle definierten Jobs anzeigen
/TEXT = '  Auswahl:'
/FOR JOB=*LIST(JOB-START)
/  TEXT = TEXT // ' ' // JOB.NAME
/END-FOR
/WRITE-TEXT '&TEXT'
/
/&*     Auswahl durch den Benutzer
/READ-VAR JNAM,PROMPT=' Welchen Job haettens'' denn gern?'
/
/&*     passenden Job in der Liste suchen und starten
/FOR JOB=*LIST(JOB-START)
/  IF (UPPER-CASE(JNAM) == UPPER-CASE(JOB.NAME))
/    ENTER-JOB &(JOB.FILE),CPU-LIMIT=&(JOB.CPU)
/  END-IF
/END-FOR
```

Strukturen, deren Elemente während des Prozedurablaufs nach Belieben erzeugt (und auch gelöscht) werden können, werden als „dynamische Strukturen“ bezeichnet. Im Gegensatz hierzu kann der Aufbau einer Struktur auch zum Zeitpunkt der Deklaration bereits vollständig definiert werden durch ein Strukturlayout, das die Namen und Typen aller Elemente festlegt. Die folgenden Kommandos erzeugen ein Layout mit dem Namen ABTEILUNG, das anschließend dazu benutzt wird, eine Variable (als „statische Struktur“) zu deklarieren:

```
/BEGIN-STRUCTURE ABTEILUNG
/  DECLARE-ELEMENT LEITER(TYPE=*STRING)
/  DECLARE-ELEMENT ANZ-MITARB(TYPE=*INTEGER)
/  DECLARE-ELEMENT UMSATZ(TYPE=*INTEGER)
/END-STRUCTURE ABTEILUNG
/DECL-VAR MEDIZIN(TYPE=*STRUCT(DEFINITION=ABTEILUNG))
/MEDIZIN.LEITER = 'Dr. Zook'
/MEDIZIN.ANZ-MITARB = 2; MEDIZIN.UMSATZ = 99001
```

Mit SET-VARIABLE lassen sich auch ganze Strukturen einander zuweisen; durch den Operanden WRITE-MODE lässt sich einstellen, ob nicht angesprochene Elemente der Zielstruktur erhalten bleiben. Wenn die Zielstruktur statisch angelegt ist, bleiben bei der Zuweisung diejenigen Elemente der Quellstruktur unberücksichtigt, die in der Zielvariablen nicht enthalten sind.

Einige Systemkommandos können wahlweise ihre Ausgaben in Strukturvariablen leiten. Beispielsweise kann durch das Kommando SHOW-FILE-ATTRIBUTES eine Struktur ausgegeben werden, deren Elemente sämtliche Kataloginformationen einer Datei enthalten. Diese Informationen können dann in einer S-Prozedur durch direkte Zugriffe auf Elemente dieser Struktur unmittelbar weiterverarbeitet werden (Näheres hierzu im Abschnitt „Strukturierte Ausgaben in S-Variablen“ auf Seite 201).

2.4.11 Fehlerbehandlung

Wie auch in anderen BS2000-Kommandofolgen führt in einer S-Prozedur ein fehlerhaftes Kommando dazu, dass die nachfolgenden Kommandos nicht mehr ausgeführt werden. Dies wird in Nicht-S-Prozeduren als „Spinoff“, in S-Prozeduren als „SDF-P-Fehlerbehandlung“ bezeichnet. Sind keine weiteren Vorkehrungen getroffen, wird der Prozedurablauf dadurch beendet und an den Aufrufer ein Fehler zurückgemeldet. Dadurch wird auch in einer rufenden Prozedur wiederum der Spinoff bzw. die SDF-P-Fehlerbehandlung ausgelöst; letztlich führt dies zum Abbruch eines Stapelauftrags bzw. im Dialogbetrieb zu einer neuen Eingabeaufforderung am Terminal.

Wenn in einer S-Prozedur nach einem Kommandofehler noch sinnvoll weitergearbeitet werden kann (oder sogar muss), ist der Fehler an einer geeigneten Stelle durch einen Fehlerbehandlungsblock aufzufangen:

```
/DELETE-FILE DATEI.A
/DELETE-FILE DATEI.B
/DELETE-FILE DATEI.C
/IF-BLOCK-ERROR
/  WRITE-TEXT 'Mindestens eine der Dateien liess sich nicht loeschen.'
/END-IF
/"Fortsetzung der Verarbeitung"
```

Die durch IF-BLOCK-ERROR eingeleitete Kommandofolge wird nur ausgeführt, wenn von einem der vorhergehenden Kommandos die SDF-P-Fehlerbehandlung ausgelöst wurde. Es darf auch, analog zum IF-Block, ein ELSE-Zweig vorhanden sein, der im Nicht-Fehlerfall zur Ausführung kommt. Anschließend ist der Spinoff aufgehoben, und die Prozedurbearbeitung wird (nach dem zugehörigen END-IF) normal fortgesetzt.

Um über die einfache Aussage „ok“ bzw. „nicht ok“ der SDF-P-Fehlerbehandlung hinaus eine differenziertere Auswertung zu ermöglichen, meldet jedes Kommando das Ergebnis seiner Ausführung über einen standardisierten Returncode zurück, der aus den Komponenten Maincode, Subcode1 und Subcode2 besteht. Diese werden von SDF-P im Fehlerfall

automatisch gesichert. Wenn keine Fehlerbedingung gesetzt ist, kann das Kommando SAVE-RETURNCODE benutzt werden, um den Returncode des zuletzt ausgeführten Kommandos zu sichern. Die vordefinierten Funktionen MAINCODE, SUBCODE1 und SUBCODE2 machen die gesicherten Werte zur Benutzung in SDF-P-Ausdrücken verfügbar.

Standardmäßig wird die SDF-P-Fehlerbehandlung durch die gleichen Situationen ausgelöst, die in Nicht-S-Prozeduren zum Spinoff führen. Mit der Einstellung ERROR-MECHANISM=*BY-RETURNCODE (Kommando SET-PROCEDURE-OPTIONS) kann die Fehlerbehandlung auch abhängig vom Returncode erfolgen (Fehler, wenn Subcode1 ungleich 0 ist). Siehe auch Abschnitt „Fehlerbehandlung“ auf Seite 69.

Der Kommandoname IF-BLOCK-ERROR soll darauf hinweisen, dass die dadurch eingeleitete Fehlerbehandlung blockorientiert ist: Sie tritt nur in Kraft, wenn der Fehler im selben Block oder einem darin eingeschachtelten Block auftrat. Im Fehlerfall wird jeder neu beginnende Block samt darin enthaltener Fehlerbehandlungen vollständig ignoriert:

```
/DELETE-FILE DATEI.A
/BEGIN-BLOCK
/  DELETE-FILE #TEMP.B
/  DELETE-FILE #TEMP.C
/  IF-BLOCK-ERROR
/    WRITE-TEXT 'Fehler beim Loeschen von temporaeren Dateien'
/  END-IF
/END-BLOCK
```

In diesem Beispiel wird im Falle eines Fehlers beim Löschen von DATEI.A der folgende BEGIN-Block komplett übersprungen; die Fehlerbehandlung wirkt daher nur auf die blockinternen DELETE-FILE-Kommandos für die Dateien #TEMP.B und #TEMP.C.

Damit eine Fehlerbehandlung gezielt auf ein einzelnes Kommando wirken kann, ohne dass dieses in einen eigenen BEGIN-Block eingeschlossen werden muss, bietet SDF-P das Kommando IF-CMD-ERROR. Es leitet einen Block ein, der nur bei Fehlern im unmittelbar davor stehenden Kommando ausgeführt wird.

2.4.12 Programmierung von Anweisungsfolgen

Wenn ein Programm Anweisungen aus einer S-Prozedur lesen will, die nächste Prozedurzeile aber ein Kommando enthält, wird normalerweise die EOF-Bedingung für die Anweisungseingabe gesetzt und damit in der Regel das Programm beendet. Hierdurch soll verhindert werden, dass infolge einer vergessenen END-Anweisung das Programm unbeabsichtigt „weiterlebt“ und zu einem späteren Zeitpunkt durch das Laden eines anderen Programms ungeordnet beendet wird.

Andererseits kann es erwünscht sein, die Abarbeitungsreihenfolge von Anweisungen ebenso wie die von Kommandos durch SDF-P-Kontrollstrukturen zu steuern. Dies wird durch einen BEGIN-Block mit PROGRAM-INPUT=*MIXED-WITH-CMD unterstützt, in dem Kommandos und Anweisungen beliebig gemischt werden können. Das Programm wird, wenn es eine Anweisung erwartet, beim Auftreten eines Kommandos in den Unterbrechungszustand versetzt und mit der nächsten Anweisung wieder fortgesetzt:

```
/DECL-PARAM STEUERDATEI
/&*     SYSDTA und SYSSTMT sind in S-Prozeduren standardmaessig
/&*     schon auf SYSCMD zugewiesen
/DECL-VAR ADDLIST,MULT-ELEM=*LIST
/READ-VAR *LIST(ADDLIST),STRING-QUOTES=*NO,INPUT=&STEUERDATEI
/SHOW-VAR ADDLIST
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/  START-LMS
//   OPEN-LIB PROJEKT-BIBL,*UPDATE
/    FOR NAME=*LIST(ADDLIST)
//     ADD-ELEM PROJ.&NAME,*LIB-ELEM(*STD,&NAME(*INCR),S)
/    END-FOR
// END
/END-BLOCK
```

Hier wird eine Reihe von Namen in eine Listenvariable eingelesen und anschließend für jedes Element der Liste die LMS-Anweisung ADD-ELEMENT einmal ausgeführt, wobei der Name mehrfach durch &-Ersetzung in die Anweisung eingefügt wird.

Analog zu Anweisungen können auch Datensätze auf diese Weise programmiert werden. Damit auch für diese Zeilen die S-Prozedursyntax genutzt werden kann (z.B. hinsichtlich Einrückung und Fortsetzungsbehandlung) und die Verwendung von Variablen problemlos möglich ist (in Datensätzen wird standardmäßig keine &-Ersetzung durchgeführt), gibt es ein Kommando SEND-DATA, das als Operanden einen beliebigen String-Ausdruck erwartet. Dessen Wert wird als Datenzeile an ein von SYSDTA lesendes Programm übergeben.Alternativ kann durch den Operandenwert *EOF auch die Endebedingung für SYSDTA gesetzt werden.

2.4.13 Ablage von Kommandoausgaben in Variablen

Um die Daten- und Meldungsausgaben eines Kommandos innerhalb einer Prozedur aufzufangen und weiter zu analysieren, kann z.B. die SYSOUT-Ausgabe in eine Datei oder eine Listenvariable umgeleitet und anschließend mit Stringzugriffsfunktionen bearbeitet werden. Dieses Verfahren ist jedoch umständlich und fehleranfällig. Mit dem Kommando EXECUTE-CMD können stattdessen die Textausgabe und der Returncode eines beliebigen Kommandos unmittelbar in Variablen übertragen werden. Zusätzlich lassen sich die Ausgaben bestimmter Kommandos auch in Form reiner Nutzdaten in Variablenstrukturen ablegen, sodass sie direkt einer Programmierung zugänglich sind. Das folgende Beispiel erstellt auf diese Weise eine Liste von Dateibezeichnungen, die in einer Schleife zur Versorgung des TRANSFER-FILE-Kommandos benutzt werden:

```
/DECL-PARAM MUSTER
/DECL-VAR DATLIST(TYPE=*STRUCT),MULT-ELEM=*LIST
/DECL-VAR DATEI(TYPE=*STRUCT)
/EXEC-CMD (SHOW-FILE-ATTR &MUSTER), -
/   TEXT-OUTPUT=*NONE,      "keine Ausgabe nach SYSOUT" -
/   STRUCT-OUTPUT=DATLIST,  "strukt. Ausgabe in Variable" -
/   RETURNCODE=*VAR(SUBCODE1=S1,     "== 0 wenn ok" -
/                   SUBCODE2=*NONE,  "nicht benoetigt" -
/                   MAINCODE=M)      "ggf. Fehlercode"
/IF (S1 > 0)
/  WRITE-TEXT 'Fehler &M'
/  EXIT-PROC ERROR=*YES(MAIN=&M)   "Fehler an Aufrufer weitergeben"
/END-IF
/FOR DATEI = *LIST(DATLIST)
/  TRANSFER-FILE TO,HOSTXYZ,(&(DATEI.F-NAME)),(...)
/END-FOR
```

Außerdem besteht die Möglichkeit, auch die während der Kommandoausführung ausgegebenen Meldungen in Variablenstrukturen aufzunehmen. Dort sind auch die Meldungsschlüssel sowie die variablen Einfügungen einzeln abgelegt, sodass sie ohne Analyse des eigentlichen Meldungstexts direkt weiterverarbeitet werden können.

Das Kommando EXECUTE-CMD ist auf Seite 657 ausführlich beschrieben. Die von SDF-P-Kommandos erzeugten Ausgabestrukturen sind im Abschnitt „Strukturierte Ausgaben in S-Variablen“ auf Seite 201, dargestellt; für andere Kommandos, die die strukturierte Ausgabe unterstützen, sei auf die zugehörigen Handbücher verwiesen.

2.4.14 S-Variablenströme

Im Gegensatz zu den herkömmlichen Systemdateien wie SYSDTA, SYSOUT und SYSLST, die lediglich Folgen von Datensätzen übertragen, welche jeweils einem Textstring entsprechen, dienen S-Variablenströme zum Transport ganzer Variablenstrukturen. Mit dem Kommando TRANSMIT-BY-STREAM wird eine solche Struktur an den Strom übergeben; dieser ist zuvor einem geeigneten Server zuzuweisen, der die Struktur entgegennimmt, weiterverarbeitet und mit einer Return-Information quittiert.

Als ein solcher Server steht FHS zur Verfügung, das die Strukturinhalte gemäß einer vordefinierten Bildschirmmaske auf der Datenstation präsentieren und Benutzereingaben zurückliefern kann. Im Abschnitt „FHS als Ausgabe-Server“ auf Seite 207, ist die Benutzung des FHS-Servers ausführlich beschrieben.

Auch Variablen selbst können wiederum als Server dienen, die die durch einen Strom übertragenen Strukturen zur weiteren Auswertung aufnehmen.

Systemseitig sind die Ströme SYSINF, SYSMSG und SYSVAR bereits vordefiniert, denen fortlaufend diejenigen strukturierten Kommando- und Meldungsausgaben zugeleitet werden, die sich auch durch EXECUTE-CMD für ein einzelnes Kommando in eine Variable lenken lassen.

2.4.15 Interaktive Benutzung von SDF-P

Im Dialogbetrieb wird die interaktive Eingabe in vielerlei Hinsicht so behandelt wie eine eigene S-Prozedur: Es können (auch prozedurlokale) Variablen deklariert bzw. erzeugt werden, mehrere Kommandos können durch Semikolon voneinander getrennt werden, und es lassen sich Kontrollstrukturblöcke eingeben. Wird zum Beispiel das Kommando WHILE am Terminal eingegeben, so werden die nachfolgenden Kommandos nicht sofort ausgeführt, sondern zwischengespeichert, bis das zugehörige END-WHILE eingegeben ist. Erst dann kann die Kontrollstruktur analysiert und ausgeführt werden. Um den Benutzer bei der interaktiven Eingabe von Kontrollstrukturen zu unterstützen, zeigt die Eingabeaufforderung stets an, welcher Block noch (als nächster) abgeschlossen werden muss. Auf diese Weise kann ein Dialogbenutzer Schleifen, Bedingungskonstrukte und Variablenzugriffe interaktiv „ausprobieren“, ehe er sie in einer Prozedur einsetzt.

Naturgemäß gibt es auch einige Einschränkungen gegenüber einer „echten“ S-Prozedur: So ist beispielsweise im Dialog außerhalb von Blockeingaben kein GOTO möglich, Fortsetzungszeilen können nicht mit insignifikanten Leerzeichen beginnen (da der Schrägstrich nicht eingegeben zu werden braucht), und Datenzeilen können in Blöcken nur mit SENDDATA erzeugt werden (weil wiederum der Schrägstrich als Unterscheidungskriterium fehlen darf).

Im Einzelnen werden die Regeln für die Dialogeingabe ab Seite 76 beschrieben.

2.4.16 Ablaufsicherheit

Das Kommando SET-PROCEDURE-OPTIONS bietet mehrere Möglichkeiten, eine Prozedur zu schützen: Sowohl die Protokollierung als auch die Unterbrechung der Prozedur kann unterbunden werden. Zusammen mit geeigneten Attributen des Prozedurbehälters (z.B. Lesebeschränkung über BASIC-ACL oder GUARDS) lässt sich damit verhindern, dass ein unbefugter Aufrufer die Prozedur liest oder deren Ablauf unzulässig verändert.

Die beim Aufruf einer Prozedur übergebenen Parameter können entweder innerhalb der Prozedur (z.B. durch die Funktion CHECK-DATA-TYPE) eingehend geprüft oder zuvor von SDF verifiziert werden, indem der Prozeduraufruf in ein Kommando eingeschalt wird (Kommandodefinition durch SDF-A mit IMPLEMENTOR=*PROCEDURE). Mithilfe der Funktion SYSTEM-CALL lässt sich nachprüfen, ob die Aufrufsyntax durch SDF anhand einer System- oder Gruppensyntaxdatei analysiert wurde.

Allgemein kann die Ablaufsicherheit einer Prozedur erhöht werden durch möglichst vollständige Deklaration aller benutzten Variablen (mit Typangabe) und Layouts. Das implizite Anlegen von Variablen, etwa infolge von Schreibfehlern, lässt sich dann über das Kommando SET-PROCEDURE-OPTIONS mit der Angabe IMPLICIT-DECLARATION=*NO verhindern.

2.4.17 Test von Prozeduren

Um Fehler in Prozeduren leichter lokalisieren zu können, ist es in der Testphase oft ratsam, die Protokollierung aller aufgerufenen Prozeduren mit dem nachfolgenden Kommando global einzuschalten (soweit die Prozeduren dies zulassen):

```
/ MODIFY-PROCEDURE-TEST-OPTIONS LOGGING=*YES
```

Zur gezielten Ablaufverfolgung kann eine Prozedur durch Drücken der K2-Taste oder Einfügen des Kommandos HOLD-PROCEDURE an geeigneter Stelle unterbrochen werden. Im Dialog können dann die Variableninhalte der unterbrochenen Prozedur inspiziert (z.B. mit dem Kommando SHOW-VARIABLE) oder auch geändert werden. Anschließend kann die Prozedur mit TRACE-PROCEDURE schrittweise oder mit RESUME-PROCEDURE bis zum Ende bzw. bis zu einer weiteren Unterbrechung fortgesetzt werden.

3 Prozedurkonzept von SDF-P

Mit SDF-P wird ein neues Prozedurformat eingeführt. Prozeduren mit diesem Format heissen „strukturierte Prozeduren“, kurz S-Prozeduren. Prozeduren, die nicht diesem Format entsprechen, werden als Nicht-S-Prozeduren bezeichnet.

Prozeduren sind allgemein immer Folgen von Kommandos, Anweisungen und Datensätzen, die in einem „Prozedurbehälter“ gespeichert werden. SDF-P unterstützt als Prozedurbehälter sowohl BS2000-Benutzerdateien als auch Bibliothekselemente. Das heißt Prozeduren in SDF-P können wie Nicht-S-Prozeduren in Benutzerdateien oder PLAM-Bibliotheken gespeichert werden. Darüber hinaus können S-Prozeduren auch in Listenvariablen abgelegt werden.

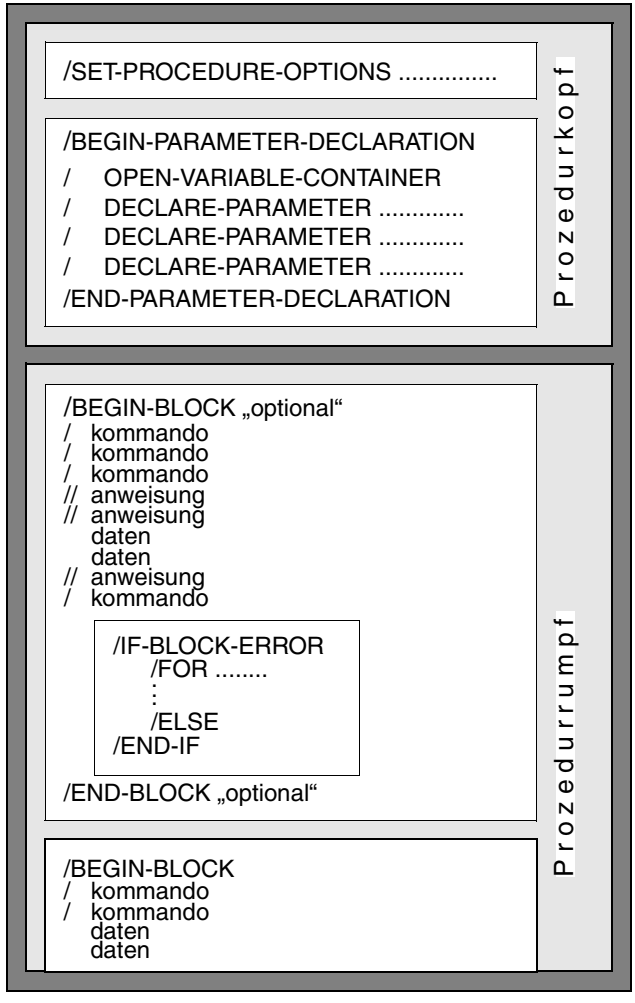
Kommandos sind neben Daten und Anweisungen das wichtigste Element in S-Prozeduren. Weitere Elemente sind Variablen, Funktionen und Ausdrücke: Variablen sind benannte Datenobjekte, denen ein Inhalt zugewiesen werden kann. Funktionen ermitteln aus Eingabeparametern ein eindeutiges Ergebnis, das an Stelle der Funktion eingesetzt wird. Ausdrücke bestehen aus Operanden und Operatoren. Der Ergebniswert wird an Stelle des Ausdrucks eingesetzt.

S-Prozeduren beginnen im Gegensatz zu Nicht-S-Prozeduren nicht mit einem der Kommandos /BEGIN-PROCEDURE oder /SET-LOGON-PARAMETERS. Beim Erstellen von S-Prozeduren muss deshalb nicht berücksichtigt werden, ob die Prozedur später als Vordergrund- oder Hintergrund-Prozedur aufgerufen wird.

Eine S-Prozedur muss nicht mit einem speziellen Prozedurabschlusskommando abgeschlossen werden. SDF-P bietet aber mit /EXIT-PROCEDURE ein solches Kommando an. Mit diesem Kommando kann die Prozedurausführung an jeder beliebigen Stelle beendet werden, und es können Fehlerinformationen an den Aufrufer der Prozedur zurückgeliefert werden.
Zu weiteren Informationen über das Beendigungsverhalten von S-Prozeduren siehe Kapitel „Prozeduren aufrufen und steuern“ auf Seite 107.

3.1 Strukturiertes Prozedurformat

Eine S-Prozedur besteht gemäß dem strukturierten Prozedurformat aus einem Prozedurkopf und einem Prozedurrumpf. In beiden Teilen können logisch zusammengehörige Blöcke definiert werden. Dieses Prinzip führt nicht nur zu sehr übersichtlichen Prozeduren, sondern beinhaltet auch zweckmäßige Funktionalität.



Der Prozedurkopf beginnt immer mit dem Kommando /SET-PROCEDURE-OPTIONS, mit dem die Prozedureigenschaften festgelegt werden. Danach folgt, falls erforderlich, die Deklaration der Prozedurparameter.

Im Prozedurrumpf stehen die Kommandos, Anweisungen und Programmdaten. Logisch zusammengehörige Kommandos bilden einen Kommandoblock, der wiederum aus dem Teil mit Kommandos, Anweisungen, Daten und aus einem Fehlerbehandlungsteil besteht. Der Kommandoblock wird durch ein /BEGIN-BLOCK und ein /END-BLOCK eingeschlossen. Für diesen Block können bestimmte Eigenschaften, z.B. unterschiedliche Behandlung von Kommandos und Daten, vereinbart werden. Im Fehlerfall wird automatisch der zugeordnete IF-BLOCK-ERROR-Block abgearbeitet. Die Prozedur sollte mit dem Kommando /EXIT-PROC abgeschlossen werden.

3.2 Konventionen für S-Prozeduren

In diesem Unterkapitel werden der Aufbau der Prozedurzeilen, Daten und Anweisungen und schließlich die Regeln für die Ausdrucksersetzung oder &-Ersetzung sowie die Regeln für das Einlesen von Programmdaten bzw. Anweisungen erläutert.

3.2.1 Prozedurzeilen

In den Prozedurzeilen stehen die Kommandos, Anweisungen und Daten einer Prozedur.

Wichtige Punkte, die dabei zu beachten sind:

- Länge von Prozedurzeilen
- Erstes Zeichen von Prozedurzeilen
- Länge von Kommandos
- Trennung von Kommandos
- Fortsetzungsbehandlung
- Kommentar-Syntax
- Marken

Länge von Prozedurzeilen

In welcher Länge Prozedurzeilen ausgewertet werden, wird beeinflusst über den Operanden INPUT-FORMAT des Kommandos SET-PROCEDURE-OPTIONS. (Das Kommando SET-PROCEDURE-OPTIONS ist Bestandteil des Prozedurkopfs, es ist im Abschnitt „Prozedurkopf erstellen“ auf Seite 81 ausführlich beschrieben).

Die Standardeinstellung für die Länge der Prozedurzeile ist INPUT-FORMAT = *FREE-RECORD-LENGTH. Damit werden in S-Prozeduren im Gegensatz zu Nicht-S-Prozeduren Prozedurzeilen in voller Länge ausgewertet oder bis zum Fortsetzungszeichen (siehe Abschnitt „Prozedurzeilen“ auf Seite 53).

Aus Kompatibilität zu Nicht-S-Prozeduren wurde im Kommando SET-PROCEDURE-OPTIONS der Operand INPUT-FORMAT = *BY-SDF-OPTION eingeführt. Er bewirkt, dass Prozedurzeilen, die Kommandos enthalten, nur bis Spalte 72 ausgewertet werden. In welcher Spalte das Fortsetzungszeichen dann stehen muss, hängt davon ab, wie im SDF-Kommando MODIFY-SDF-OPTIONS der Operand CONTINUATION eingestellt wurde.

Hinweis

ISAM-Dateien werden nur mit KEY-POS = 5 und KEY-LEN = 8 als S-Prozeduren akzeptiert.

Erstes Zeichen von Prozedurzeilen

Folgendes ist zu unterscheiden:

- Die erste Prozedurzeile muss mit *einem* Schrägstrich (/) beginnen.
- Sonstige Prozedurzeilen, die mit nur einem Schrägstrich beginnen, enthalten Kommandos. Kommandos erwartet SDF aus der logischen Systemdatei SYSCMD (siehe dazu Handbuch „Kommandos, Bd. 1-5“ [3]).
- Prozedurzeilen, die mit zwei Schrägstrichen beginnen, enthalten Anweisungen im SDF-Format (an ein Programm mit SDF-Schnittstelle). Anweisungen erwartet SDF aus der logischen Systemdatei SYSSTMT, für die dieselbe Zuweisung gilt, die für die Systemdatei SYSDTA getroffen wurde (siehe Handbuch Handbuch „Kommandos, Bd. 1-5“ [3]).
- Prozedurzeilen, die nicht mit Schrägstrich beginnen, sind Datenzeilen. Sie enthalten Programm-Eingabe-Daten.
- Fortsetzungszeilen für Kommandos müssen als erstes relevantes Zeichen einen Schrägstrich enthalten. Fortsetzungszeilen für Anweisungen müssen als erste relevante Zeichen zwei Schrägstriche enthalten. Die Fortsetzungsbehandlung von Datenzeilen ist vom verarbeitenden Programm abhängig.
- Kommandos, Anweisungen und Daten dürfen nicht in einer Prozedurzeile verkettet werden.

Länge von Kommandos

Wenn die Länge von Kommandos geprüft wird, ist zu berücksichtigen, ob im Kommandoaufruf eine Ausdrucksersetzung vorgesehen ist. Kommandos dürfen nach einer Ausdrucksersetzung nicht länger als 16364 Zeichen sein.

Trennung von Kommandos

Jede Prozedurzeile kann mehrere Kommandos enthalten. Diese Kommandos müssen durch Semikolon voneinander getrennt werden.

Das erste Kommando steht am Anfang der Prozedurzeile und muss mit einem Schrägstrich (/) beginnen. Wenn in dieser Prozedurzeile mehrere Kommandos stehen sollen, müssen sie durch Semikolon voneinander getrennt werden und dürfen nicht mit einem Schrägstrich eingeleitet werden.

Kommandos, die nach AID-Kommandos durch Semikolon getrennt geschrieben werden, werden als Teil der AID-Kommandofolge bearbeitet, d.h. sie sind nicht Teil der Prozedureingabe, sondern werden direkt von AID verarbeitet.

Fortsetzungsbehandlung

Kommandos, Kommandofolgen oder Anweisungen können über mehrere Zeilen verteilt werden. Die „Fortsetzungsbehandlung“ bestimmt, wie zusammengehörende Zeilen erkannt und ausgewertet werden.

In S-Prozeduren können über Fortsetzungszeilen bis zu 16364 Zeichen miteinander zu einer Kommandofolge verbunden werden. (Bei ISP-Kommandos bis zu 4090 Zeichen.)

Das Fortsetzungszeichen ist der Bindestrich (–). An welcher Stelle der Prozedurzeile das Fortsetzungszeichen stehen darf, ist implizit über den Operanden INPUT-FORMAT im Kommando SET-PROCEDURE-OPTIONS eingestellt.

Standardmäßig kann das Fortsetzungszeichen in einer beliebigen Spalte der Prozedurzeile stehen (INPUT-FORMAT = *FREE-RECORD-LENGTH). Allerdings dürfen dem Fortsetzungszeichen keine weiteren Zeichen folgen; es muss das letzte Zeichen der Prozedurzeile (abgesehen von Leerzeichen) sein.

Gilt INPUT-FORMAT = *BY-SDF-OPTION, muss die Einstellung des Operanden CONTINUATION im Kommando MODIFY-SDF-OPTION berücksichtigt werden:

– Bei CONTINUATION = *OLD-MODE muss das Fortsetzungszeichen für Kommandos genau auf der Spalte 72 stehen.
– Bei CONTINUATION = *NEW-MODE kann das Fortsetzungszeichen auf einer beliebigen Spalte zwischen den Spalten 2 und 72 stehen.

Kommentare in Anführungszeichen

Kommentare in Anführungszeichen werden zur internen Dokumentation von Prozeduren eingesetzt. Innerhalb eines Kommandos oder einer Anweisung werden Kommentare überall dort zugelassen, wo Leerzeichen erlaubt sind. (Ausnahme: nicht nach Fortsetzungszeichen.)

Als Kommentar ist beliebiger Text zugelassen, der in Anführungszeichen (" ") eingeschlossen ist.

Auch wenn ein Kommentar separat in eine Zeile geschrieben wird, muss er nach dem einleitenden Schrägstrich (/) in Anführungszeichen eingeschlossen werden. SDF-P interpretiert eine solche Prozedurzeile als Kommando, das nur aus einem Kommentar besteht. Es wird nicht ausgeführt und nicht protokolliert. Wenn ein solches „Kommentar-Kommando“ mit einer Marke versehen ist, wird nur die Marke protokolliert. Außerdem findet für solche separat in eine Zeile geschriebenen Kommentare keine Ausdrucksersetzung statt.

Zeilenende-Kommentare

Zeilenende-Kommentare sollen vor allem der Entwicklung von S-Prozeduren Rechnung tragen. Zeilenende-Kommentare werden durch das Zeichenpaar „&*“ markiert. Der Text nach diesem Zeichenpaar wird ignoriert, wenn die Prozedur ausgewertet wird. Fortsetzungszeichen, Zeichenseparatoren (wie das Semikolon) und &-Zeichen müssen deshalb vor diesem Zeichenpaar geschrieben werden.

Beispiele

Man kann mithilfe des Zeichenpaars „&*“ einen Hinweis hinter ein Kommando anfügen:

```
/PRINT-DOC &DATEI &* Hier wird meine Datei ausgedruckt.
```

oder man kann damit die Eingabe abstimmen:

```
(1)  PRINT-DOC &FILE ,PRINT-JOB-NAME='&(SUBSTRING(FILE,1,8))'

(2)  PRINT-DOC &FILE &*,PRINT-JOB-NAME='&(SUBSTRING(FILE,1,8))'
```

So wird bei (1) das PRINT-DOCUCMENT-Kommando modifiziert, während bei (2) nur durch Einfügung von „&*“ auf die Standardeinstellung zurückgegriffen wird. Dabei kann der Zustand (1) sehr leicht wieder hergestellt werden. Es muss nur „&*“ wieder gelöscht werden.

Hinweise

- Die Zeichenfolge „&*“ war bisher in S-Prozeduren - sogar in eingeschlossenen Anführungszeichen - nicht erlaubt. Lediglich in Kommentarzeilen ohne Operationsnamen waren sie in der Vergangenheit nicht verboten: z.B. „ cmt1 &* cmt2 “.
  Das sollte bei eventuell auftretenden Inkompatibilitäten mit früher erstellten Prozeduren berücksichtigt werden.
- Das Zeichenpaar „&*“ darf nicht durch Leerzeichen oder sonstige Zeichen getrennt werden.
- Es gibt eine Möglichkeit das Zeichenpaar „&*“ zum Teil der Eingabe zu machen, und zwar durch Verdoppelung des &-Zeichens: &&*.
  Allerdings muss dabei aufgepasst werden, dass das &-Zeichen nicht zu oft verdoppelt wird. So wird durch „&&&*“ die Absicht zunichte gemacht, für das Zeichenpaar „&*“ die Sonderfunktion aufzuheben. In diesem Fall leitet diese Zeichenfolge wieder einen Zeilenende-Kommentar ein.

Marken

Prozedurzeilen können mit Marken versehen werden. Diese Marken können bei der Schachtelung von Kommandoblöcken oder bei Sprüngen mit den Sprungkommandos als Sprungziele verwendet werden. Sie werden auch als S-Marken bezeichnet.

Für S-Marken gelten folgende Regeln:

– SDF-Datentyp: <structured-name>
– Maximale Länge: 255 Zeichen
– Zeichenvorrat: A...Z, 0...9, $, #, @, -
– Erstes Zeichen: Buchstabe
– Letztes Zeichen: Doppelpunkt
– Der Doppelpunkt muss ohne Leerzeichen an die Marke anschließen.
– Marken dürfen nicht durch Ausdrucksersetzung erzeugt werden.
– Marken müssen vor dem Operationsnamen stehen und von ihm mindestens durch ein Leerzeichen getrennt sein. („Operationsname“ ist der Name, mit dem ein Kommando aufgerufen wird.)

Zur Behandlung von Nicht-S-Marken (.marke) siehe auch Umstellhinweise im Kapitel „Nicht-S-Prozeduren umstellen“ auf Seite 293ff.

3.2.2 Ausdrucksersetzung

Ausdrucksersetzung spielt eine wichtige Rolle beim Aufruf von Kommandos. Sie ermöglicht es, Kommandos dynamisch zu erzeugen und den Inhalt von Prozedurzeilen zu ändern. (Nähere Informationen über Ausdrücke enthält das Kapitel „Ausdrücke“ auf Seite 255.)

Ausdrucksersetzung läuft so ab, dass nicht die endgültigen Eingaben bzw. Eingabeteile angegeben werden, sondern nur Platzhalter, die erst zum Ablaufzeitpunkt durch aktuelle Werte ersetzt werden.

Escape-Zeichen

Als Escape-Zeichen wird das Zeichen bezeichnet, das die Ausdrucksersetzung einleitet. Es wird dem zu ersetzenden Ausdruck direkt vorangestellt und deutet an, dass die folgenden Zeichen nicht den aktuellen Operandenwert sondern einen Platzhalter darstellen.

Für das Escape-Zeichen werden zwei Anwendungen unterschieden:

– Kommandos und Anweisungen
– Datenzeilen

Für Kommandos und Anweisungen gilt das Zeichen & als Escape-Zeichen.

Welches Zeichen in Datenzeilen als Escape-Zeichen verwendet werden soll, kann im Prozedurkopf mit dem Kommando SET-PROCEDURE-OPTIONS vereinbart werden (Operand DATA-ESCAPE-CHAR). Zur Wahl stehen die Zeichen &, #, *, @ und $.

Standardmäßig findet in Datenzeilen keine Ausdrucksersetzung statt; es gilt DATA-ESCAPE-CHAR = *NONE.

Da das Zeichen & das Standard-Escape-Zeichen ist, wird die Ausdrucksersetzung im folgenden auch als &-Ersetzung bezeichnet.

Wird einem Ausdruck bzw. einer Variablen ein Escape-Zeichen vorangestellt, wird der Wert dieses Ausdrucks bzw. dieser Variablen nicht direkt vom Kommando verarbeitet wird, sondern so als ob der Wert statt der Escape-Zeichenfolge im Eingabesatz geschrieben steht. (Direkte Ausdrucksersetzung, z.B. /SET-VARIABLE A=B, wird von einem Kommando nur dann ausgeführt, wenn das Kommando einen Ausdruck unterstützt, ansonsten müssen Escape-Zeichenfolgen verwendet werden.)

Die in den folgenden Abschnitten beschriebenen Regeln gelten sowohl für die Ausdrucksersetzung in Kommandos und Anweisungen als auch in Datenzeilen.

Beispiel

```
/SHOW-FILE-ATTRIBUTES &DATEI
```

Es werden die Dateiattribute einer Datei angezeigt, deren Name in der Variablen DATEI hinterlegt ist.

Syntax

In strukturierten Prozeduren kann das Escape-Zeichen nicht nur auf Variablen (einschließlich Prozedurparameter) angewendet werden, sondern auch auf Funktionen und Ausdrücke sowie auf Jobvariablen. Für die Ausdrucksersetzung gilt folgende Syntax:

&(ausdruck)

oder:

&name

Es bedeuten:

&	Escape-Zeichen
ausdruck:	Ausdruck oder Name einer Jobvariablen
name:	Name einer Variablen (wenn sie keinen Punkt enthält) oder Name einer vordefinierten Funktion ohne Parameter

Regeln

„ausdruck“ wird ausgewertet, der Ergebniswert in den Datentyp STRING konvertiert und anschließend als aktueller Operandenwert eingesetzt.

Ist „ausdruck“ ein Funktionsaufruf in der Form &(funktion( )), wird diese Funktion ausgeführt und das Ergebnis der Funktion als aktueller Parameterwert eingesetzt. Der Funktionsaufruf kann Eingabeparameter enthalten.

Ist „ausdruck“ ein Name, wird zunächst eine Variable mit dem entsprechenden Variablennamen gesucht. Kompatibel zu Nicht-S-Prozeduren wird eine Angabe &(name) folgendermaßen ausgewertet:

1. Es wird eine Variable dieses Namens gesucht.
2. Wenn es keine solche Variable gibt, wird eine Funktion dieses Namens gesucht.
3. Wenn es weder eine Variable noch eine Funktion mit dem angegebenen Namen gibt, hängt es von der Einstellung des Operanden JV-REPLACEMENT im Kommando SET-PROCEDURE-OPTIONS bzw. MODIFY-PROCEDURE-OPTIONS ab, ob eine entsprechende Jobvariable gesucht wird. Wird hier der Wert AFTER-BUILTIN-FUNCTION eingestellt, wird eine Jobvariable gesucht; wird der Wert *NONE eingestellt, wird keine Jobvariable gesucht.
4. Wenn der angegebene Name weder eine Variable noch eine Funktion oder Jobvariable bezeichnet, wird die Fehlerbehandlung angestoßen.

Auf das Escape-Zeichen darf als erstes Zeichen folgen:

– eine offene Klammer (als Trennzeichen (Angabe: &(ausdruck))
– das erste Zeichen des Namens einer Variablen oder einer Funktion (Angabe: &name)

Damit im Text &-Zeichen erhalten bleiben, die sonst als Escape-Zeichen wirken würden, sollen einfache durch doppelte Escape-Zeichen (&&) ersetzt werden.

Wenn auf „name“ ein Punkt „.“ folgt, geht dieser Punkt bei der Ausdrucksersetzung verloren (kompatibel mit Nicht-S-Prozeduren).
Wenn auf „name“ Sonderzeichen folgen, werden diese als Trennzeichen interpretiert. Bindestriche im Namen werden im Gegensatz zu Nicht-S-Prozeduren mitübersetzt und wirken nicht als Trennzeichen (Beispiel: &JOB-CLASS wird ersetzt durch die aktuelle Jobklasse, z. B. JCB00200).

Kommandos oder Datenzeilen können beliebig viele Ausdrucksersetzungen enthalten.

Ausdrucksersetzungen dürfen geschachtelt werden; rekursive Ausdrucksersetzung ist jedoch nicht möglich. Wird &(ausdruck) z.B. durch die Zeichen A + &B ersetzt, wird &B nicht weiter ausgewertet; das Zeichen & bei B bleibt erhalten.

Vor oder in einer Marke darf keine Ausdrucksersetzung durchgeführt werden.

Wenn bei der Ausdrucksersetzung ein Fehler auftritt, wird standardmäßig die Fehlerbehandlung ausgelöst. Die Fehlerbehandlung kann für Datensätze unterbunden werden, wenn der Operand DATA-ERROR-HANDLING = *NO im Kommando SET-PROCEDURE-OPTIONS angegeben wird.

Wenn eine Prozedurzeile ein einzeln stehendes &-Zeichen enthält, wird ebenfalls die Fehlerbehandlung ausgelöst.

Einschränkungen

Über Ausdrucksersetzung dürfen keine Kontrollflusskommandos erzeugt werden. Wenn ein Kontrollflusskommando dennoch im Kommandonamen eine Ausdrucksersetzung enthält, wird es zum Ablaufzeitpunkt der Prozedur abgewiesen.

Zeichen oder Namen, die nicht durch Ausdrucksersetzung erzeugt werden dürfen:

- Fortsetzungszeichen am Ende einer Prozedurzeile
- Escape-Zeichen, das dann rekursiv wirksam sein soll
- das Trennzeichen für Kommandos oder Anweisungen (Semikolon „;“)
- Sprungmarken oder Blocknamen
- Kommandos, die nicht durch Ausdrucksersetzung erzeugt werden dürfen:
- SDF-P-Kontrollflusskommandos:

```
BEGIN-BLOCK
BEGIN-PARAMETER-DECLARATION
CYCLE
DECLARE-PARAMETER
ELSE, ELSE-IF
END-BLOCK, END-FOR, END-IF, END-WHILE
END-PARAMETER-DECLARATION
EXIT-BLOCK
FOR
GOTO
IF, IF-BLOCK-ERROR, IF-CMD-ERROR
INCLUDE-BLOCK
REPEAT
UNTIL
WHILE
```

- AID-Kommandos, denen eine Kommandoliste oder Subkommandoliste folgt (siehe Handbuch „AID (BS2000)“ [6])
- das Kommando SET-JOB-STEP (siehe Handbuch „Kommandos, Bd. 1-5“ [3])
- OPEN-VARIABLE-CONTAINER im DECLARE-PARAMETER-Block vor dem ersten DECLARE-PARAMETER-Kommando.

3.2.3 Daten und Anweisungen

Neben Kommandos können Prozedurzeilen ebenso Daten und Anweisungen enthalten. Daten können in S-Prozeduren auch mit dem Kommando SEND-DATA übergeben werden und Anweisungen mit dem Kommando SEND-STMT.

Dieser Abschnitt behandelt folgende Themen:

– Daten einlesen
– Dateiende-Bedingung erzeugen
– Daten-, Anweisungs- und Kommandozeilen mischen

Daten einlesen

Es gibt verschiedene Wege, über die einem Programm Datenzeilen übermittelt werden: So kann im Programm eine Datei eröffnet werden, die die Datenzeilen enthält, oder es können Daten von SYSDTA gelesen werden.

Damit im Programm eine Eingabedatei geöffnet und Datenzeilen gelesen werden können, muss dem Programm diese Datei zugeordnet werden. Dies verläuft in Prozeduren genauso wie auf Systemebene. Detaillierte Beschreibungen dazu enthält das Handbuch „Einführung in das DVS“ [1].

Der Standard-Eingabeweg für Daten ist SYSDTA. SYSDTA gehört wie die Systemdateien SYSCMD, SYSLST, SYSOPT und SYSOUT zur SYSFILE-Umgebung. SYSDTA ist eine logische Datei (Systemdatei) und bezeichnet den Weg, über den Daten vom System an ein Programm weitergereicht werden.

In S-Prozeduren ist (im Gegensatz zu Nicht-S-Prozeduren) die Standardeinstellung, dass SYSDTA SYSCMD zugewiesen wird. SYSDTA kann zum Beispiel mit dem Kommando ASSIGN-SYSDTA einer Datei zugewiesen werden, aus der das Programm dann der Reihe nach die Datensätze lesen kann (zu ASSIGN-SYSDTA: siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

Wird SYSDTA in einer Prozedur einer Datei zugewiesen, ergeben sich keine Unterschiede gegenüber einer Dateizuweisung auf Systemebene. Datensätze können aber auch direkt in die Prozedur geschrieben werden.

Wird SYSDTA mit dem Kommando ASSIGN-SYSDTA TO-FILE=*SYSCMD „umgelenkt“, liest das Programm die Datenzeilen aus der Prozedur.

In S-Prozeduren können Datenzeilen mit dem Kommando SEND-DATA an das geladene Programm übergeben werden. Dabei enthält jeder Kommandoaufruf einen Datensatz oder einen Ausdruck, der ausgewertet den Datensatz ergibt.

Daten können auch ohne den Kommandoaufruf SEND-DATA in eigene Datenzeilen geschrieben werden, dann enthält eine Prozedurzeile genau einen Eingabe-Datensatz. Dabei muss allerdings beachtet werden, wie die Dateiende-Bedingung erzeugt wird (Dateiende-Bedingung: siehe Seite 65). Falls die Datensätze einen ISAM-Schlüssel enthalten, wird dieser ignoriert.

Daten über das Kommando SEND-DATA einzugeben, bietet jedoch einige Vorteile:

– Die Prozedurzeile, die den Datensatz „enthält“, kann mit einer Marke eingeleitet werden.
– Die Prozedurzeile mit dem Datensatz kann einen Kommentar enthalten.
– Der Datensatz kann sich über mehrere Fortsetzungszeilen erstrecken.
– Daten und Dateiende-Bedingung werden über eine einheitliche Oberfläche erzeugt (siehe Abschnitt „Dateiende-Bedingung erzeugen“ auf Seite 65).

Beispiel

```
/ABC = 'Text'
/DEF = 'verarbeitung'
/START-EXE PROGRAM1                          "Programm PROGRAM1 laden"
/SEND-DATA ABC                               "Eingabe: Text"
/SEND-DATA 'EINGABE'                         "Eingabe: EINGABE"
/SEND-DATA 'Dies ist ein ganz langer Eingabe'-
/'satz, der auf jeden Fall mehr als 72 Zei'-
/'chen lang sein soll'
/SEND-DATA ABC // DEF                        "Eingabe: Textverarbeitung"
/SEND-DATA ...
```

In diesem Beispiel werden nacheinander verschiedene Datensätze eingelesen. Der vierte Aufruf von SEND-DATA enthält als Argument einen Ausdruck, der ausgewertet die Eingabe „Textverarbeitung“ ergibt.

SEND-DATA kann auch innerhalb einer Schleife aufgerufen werden, in der Eingabe-Datensätze nacheinander erzeugt werden.

Beispiel

```
/SET-VARIABLE A = 'Text'
/SET-VARIABLE B = 'verarbeitung'
/SET-VARIABLE C = A // B
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/START-EXE PROGRAM1                          "Programm PROGRAM1 laden"
/FOR EINGABE = (A,B,C)                       "FOR-Schleife, in der der"
/                                            "FOR-Schleife, in der der"
/                                            "die Inhalte der Variablen A, B, C
/                                            zugewiesen werden"
/SEND-DATA EINGABE
/END-FOR
/END-BLOCK
```

Man kann Eingabe-Datensätze auch direkt in die Prozedurzeile schreiben, ohne den Kommandoaufruf /SEND-DATA zu benutzen.

Beispiel

```
/SET-PROCEDURE-OPTIONS DATA-ESCAPE-CHAR = *STD
/VAR1 = 'Text'
/VAR2 = 'verarbeitung'
/START-EXE PROGRAM1                              "Programm PROGRAM1 laden"
&VAR1
'EINGABE'
&(VAR1 // VAR2)
...
```

Als Datenzeilen werden nacheinander übergeben:

```
Text
'EINGABE'
Textverarbeitung
```

Daten-, Kommando- und Anweisungszeilen mischen

Für die Weitergabe von Kommandos und Datenzeilen bietet SDF-P das Kommando SEND-DATA. Dabei können Kommandos und Datenzeilen beliebig gemischt werden.

Für die Weitergabe von Anweisungen an ein Programm bietet SDF-P ein eigenes Kommando: SEND-STMT. Die Anweisung wird dabei direkt als Operandenwert im Kommandoaufruf übergeben oder als Ausdruck, der ausgewertet die Anweisung ergibt.

Daten-, Anweisungs- und Kommandozeilen zu mischen, kann auch im Kommando BEGIN-BLOCK mit der Operandeneinstellung PROGRAM-INPUT=*MIXED-WITH-CMD vereinbart werden. Diese Einstellung gilt dabei für den aktuellen Kommandoblock und für alle in diesen Block geschachtelten Blöcke; sie kann in einem untergeordneten Block nicht ausgeschaltet werden.

PROGRAM-INPUT=*MIXED-WITH-CMD bewirkt, dass Daten-, Anweisungs- und Kommandozeilen vom System „gleichartig“ behandelt werden. Das heißt, die Kommandozeile löst keine Dateiende-Bedingung aus. Stattdessen unterbricht das System das Programm:

– Es hebt die „Empfangsbereitschaft“ des Programms auf, was einer Programm-Unterbrechung mit HOLD-PROGRAM entspricht.
– Das System führt die Kommandos aus, bis wieder eine Daten- oder Anweisungszeile eingelesen wird.
– Es stellt die „Empfangsbereitschaft“ des Programms wieder her, was einer Fortsetzung des Programms mit RESUME-PROGRAM entspricht.
– Anschließend werden Daten- oder Anweisungszeilen an das Programm weitergereicht, bis wieder ein Kommando eingelesen wird oder explizit die Dateiende-Bedingung erzeugt wird (zum Beispiel mit /SEND-DATA *EOF).

Beispiel

```
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/                                    "Daten- und Kommandozeilen mischbar"
/START-EXE PROGRAM1                  "PROGRAM1 starten"
/FOR V = *LIST(L)                    "FOR-Schleife, Laufvariable: V wird an"
/                                    "PROGRAM1 weitergegeben"
&V
/END-FOR
/END-BLOCK
```

In diesem Beispiel wird das Programm PROGRAM1 gestartet. Sobald von SYSDTA Daten angefordert werden, wird das Programm unterbrochen; die folgenden Kommandos (BEGIN-BLOCK und FOR) werden ausgeführt.

In der FOR-Schleife wird der Variablen V zunächst der Inhalt des ersten Elements der Liste L zugewiesen.

Die folgende Zeile (&V) ist eine Datenzeile: Das Programm wird fortgesetzt, die Datenzeile, also der Inhalt der Variablen V, an das Programm übergeben.
Sobald wieder Daten von SYSDTA angefordert werden, wird das Programm erneut unterbrochen und die folgenden Kommandos ausgeführt: END-FOR und FOR. Wenn die Listenvariable L noch mehr Elemente enthält, wird die Schleife fortgesetzt und der Variablen V der Inhalt des nächsten Elements zugewiesen. Die Dateneingabe wird beendet, wenn die Listenvariable L „abgearbeitet“ ist und die FOR-Schleife beendet wird.

Mit END-BLOCK wird die Einstellung PROGRAM-INPUT=*MIXED-WITH-CMD aufgehoben, sodass das folgende Kommando die Dateiende-Bedingung erzeugt.

Wenn Daten-, Anweisungs- und Kommandozeilen gemischt werden können, trifft dies auch auf den Aufruf geschachtelter Prozeduren mit CALL-PROCEDURE oder INCLUDE-PROCEDURE zu: Das Programm wird unterbrochen und die aufgerufene Prozedur zunächst im Kommandomodus bearbeitet. Bevor wieder Daten an das anfordernde Programm weitergereicht werden, muss das Kommando RESUME-PROGRAM aufgerufen werden (die „Empfangsbereitschaft“ wird wiederhergestellt). Es darf allerdings kein implizites RESUME-PROGRAM verwendet werden.

Datenprozeduren müssen demnach mit dem Kommando RESUME-PROGRAM beginnen. (Außerdem gilt: S-Prozeduren müssen mit einem Schrägstrich beginnen!)

Mit dem Kommando HOLD-PROGRAM kann die Dateneingabe wieder unterbrochen werden, sodass ein explizites oder implizites EXIT-PROCEDURE ausgeführt werden kann.

Beispiel

```
BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/START-EXE PROGRAM2
/CALL-PROCEDURE DATA.PROC                "Aufruf einer Datenprozedur"
    -----> /RESUME-PROGRAM               "Programm empfangsbereit"
              eingabe-1
              eingabe-2
              ..
           /HOLD-PROGRAM                 "Programm unterbrochen"
           /EXIT-PROCEDURE
    <----
/...                                     "Fortsetzung übergeordnete Prozedur"
```

Wenn die Datenprozedur nicht mit HOLD-PROGRAM beendet wird, wird nach dem Lesen der letzten Prozedurzeile die Dateiende-Bedingung erzeugt.

In der Datenprozedur gilt nicht das in der aufrufenden Prozedur gesetzte PROGRAM-INPUT=*MIXED-WITH-CMD. Der Kommandoaufruf /HOLD-PROGRAM erzeugt jedoch noch keine Dateiende-Bedingung. Die Kommandos der Prozedur werden ausgeführt, die Prozedur also beendet (implizit, wenn EXIT-PROCEDURE fehlt). In der aufrufenden Prozedur ist die Dateiende-Bedingung jedoch aufgehoben. Das Programm läuft also nach dem untergeschobenen RESUME-PROGRAM weiter.

Tritt bei der Ausführung eines Kommandos ein Fehler auf, wird im Kommandomodus zum nächsten Fehlerbehandlungsblock (IF-BLOCK-ERROR) verzweigt.

Einschränkungen

Nicht mit Daten gemischt werden, dürfen die Kommandos des Testhilfe-Dienstprogramms AID. Ein Mischen von Datenzeilen mit diesen Kommandos kann zu einem unerwarteten Verhalten des Programms führen, da nach jedem Kommando das Programm fortgesetzt wird (siehe Handbuch „AID (BS2000)“ [6]).

Der Fehlerstatus von Anweisungen wird nicht immer auf Kommandoebene übertragen. So können Fehler auf Anweisungsebene nicht immer bis zum Programmende durch einen IF-BLOCK-ERROR-Block auf Kommandoebene verarbeitet werden.
Bei den Einstellungen BEGIN-BLOCK PROGRAM-INPUT=*STD / *MIXED-WITH-CMD-passiert im Einzelnen Folgendes:

– Bei PROGRAM-INPUT=*STD verursacht die Eingabe eines Kommandos (nicht HOLD-PROGRAM) eine Dateiende-Bedingung für Anweisungs-Eingaben und bestimmten Programmreaktionen (z.B. bei TERM mit Fehler). Mögliche Fehler werden auf Kommandoebene bei Programmbeendigung übertragen. D.h. sie können ab diesem Zeitpunkt mit IF-BLOCK-ERROR weiter verarbeitet werden.

  Beispiel

```
/BEGIN-BLOCK PROGRAM-INPUT=*STD
/START-EXE MY-UTILITY     "WENN EOF BEI RDSTMT: ABBRUCH"
//...       "FEHLER"
/IF-BLOCK-ERROR; WRITE-TEXT 'FEHLER WEGEN EOF'; END-IF
/...       "FEHLERVERARBEITUNG"
/END-BLOCK
```

– Die Operandeneinstellung PROGRAM-INPUT=*MIXED-WITH-CMD(PROPAGATE-STMT-RC=*STD) unterbricht Kommandoeingaben während Programmanweisungen (und mögliche Fehler) eingegeben werden. Es wird aber kein Fehler auf Kommandoebene übertragen. IF-BLOCK-ERROR verarbeitet in diesem Fall den Fehler auf Anweisungsebene nicht. Der Fehler kann aber durch die vordefinierte Funktion STMT-SPINOFF( ) abgefragt werden: D.h. an Stelle von IF-BLOCK-ERROR muss IF-STMT-SPINOFF( ) angegeben werden.

Beispiel

```
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/START-EXE MY-UTILITY     "WENN EOF BEI RDSTMT: ABBRUCH"
//...      "FEHLER"
/IF-BLOCK-ERROR; WRITE-TEXT 'ES WIRD KEIN TEXT GESCHRIEBEN'; END-IF
/...      "KEINE FEHLERVERARBEITUNG AUF KOMMANDOEBENE"
/END-BLOCK
/ "ES WIRD JETZT EOF GEMELDET, DA KEINE DATENEINGABE MEHR ERFOLGT."
/ "PROGRAM BRICHT AB."
/ "EIN FEHLER WIRD NACH BLOCKENDE ZUR KOMMANDOEBENE UEBERTRAGEN"
```

– Die Operandeneinstellung PROGRAM-INPUT=*MIXED-WITH-CMD(PROPAGATE-STMT-RC=*TO-CMD-RC) unterbricht Kommandoeingaben während Programmanweisungen (und mögliche Fehler) eingegeben werden. Returncodes von Programmanweisungen werden auf die Kommandoebene übertragen. Damit können auf Anweisungsebene aufgetretene Fehler mit dem Kommando IF-BLOCK-ERROR und den vordefinierten Funktionen SUBCODE1( ), SUBCODE2( ) und MAINCODE( ) behandelt werden.

Beispiel

```
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD -
                           /(PROPAGATE-STMT-RC=*TO-CMD-RC)
/START-LMS
//...              "FEHLER AUF ANWEISUNGSEBENE"
//...
/IF-BLOCK-ERROR
/...               "FEHLERBEHANDLUNG AUF KOMMANDOEBENE"
/END-IF
//END
/END-BLOCK
```

Dateiende-Bedingung erzeugen

Ob das System die Dateiende-Bedingung erzeugt oder ein Kommando ausführt, hängt davon ab, wie der Operand PROGRAM-INPUT im BEGIN-BLOCK-Kommando eingestellt ist. Bei der Standardeinstellung BEGIN-BLOCK PROGRAM-INPUT=*STD wird die Dateiende-Bedingung erzeugt, wenn im Datenstrom ein Kommando aufgerufen wird. Das heißt, Daten- und Kommandozeilen können hier - außer man gibt SEND-DATA an - nicht gemischt werden.

Beispiel

```
/PROG: BEGIN-BLOCK
/ABC = 'Text'
/DEF = 'verarbeitung'
/START-EXE PROGRAM1              "Programm PROGRAM1 laden"
&ABC
EINGABE
&(ABC // DEF)
...
/WRITE-TEXT ...
/END-BLOCK PROG
```

In diesem Beispiel beendet das Kommando WRITE-TEXT die Dateneingabe: Das System erkennt am einleitenden Schrägstrich, dass es sich um eine Kommandozeile handelt und erzeugt die Dateiende-Bedingung. Nach Beendigung des Programms wird das Kommando ausgeführt.

Werden bei der Standardeinstellung von BEGIN-BLOCK Daten mit dem Kommando SEND-DATA an das Programm übergeben, muss - um die Dateiende-Bedingung zu erzeugen - im letzten SEND-DATA-Kommando der Operand RECORD = *EOF angegeben werden.

Beispiel

```
/PROG: BEGIN-BLOCK
/VAR1 = 'Text'
/VAR2 = 'verarbeitung'
/START-EXE PROGRAM1                        "Programm PROGRAM1 laden"
/SEND-DATA VAR1                            "Eingabe: Text"
/SEND-DATA 'EINGABE'                       "Eingabe: EINGABE"
/SEND-DATA VAR1 // VAR2                    "Eingabe: Textverarbeitung"
/SEND-DATA RECORD=*EOF                     "Eingabe-Ende: EOF"
```

Das Programm kann nacheinander die Datensätze einlesen.

Beispiel

```
/PROG: BEGIN-BLOCK
/SET-VARIABLE A = 'Text'
/SET-VARIABLE B = 'verarbeitung'
/SET-VARIABLE C = A // B
/BEGIN-BLOCK DATA-INSERT = *YES
/START-EXE PROGRAM1                  "Programm PROGRAM1 laden"
/FOR EINGABE = (A,B,C)               "FOR-Schleife, in der der"
/                                    "Laufvariablen EINGABE nacheinander"
/                                    "die Inhalte der Variablen A, B und C"
/SEND-DATA EINGABE                   "zugewiesen werden"
/END-FOR
/SEND-DATA *EOF                      "Eingabe-Ende: EOF"
/END-BLOCK
```

Bei der Einstellung BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD können - wie im vorhergegangen Abschnitt beschrieben wurde - Daten- und Kommandozeilen ohne zusätzliche Angabe eines SEND-DATA-Kommandos gemischt werden. Allerdings muss auch hier zur Erzeugung der Dateiendebedingung SEND-DATA=*EOF angegeben werden.

Beispiel

Zwei Kommandoblöcke werden ineinander geschachtelt. Im übergeordneten Block ist PROGRAM-INPUT=*MIXED-WITH-CMD eingestellt, was sich auch auf den untergeordneten Block auswirkt.

```
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/...

/COMP: BEGIN-BLOCK
/START-EXE $RZ.FOR1
   PROGRAM STATIST
   ...
   END
/SEND-DATA *EOF
/START-EXE $BINDER
...
/END-BLOCK COMP
...
/END-BLOCK
```

Im untergeordneten Block liest der Fortran-Compiler FOR1 ein Fortran-Programm zeilenweise ein, bis das System einen Kommandoaufruf erkennt.

Das Programm wird unterbrochen und das Kommando ausgeführt: Es wird die Dateiende-Bedingung erzeugt und dadurch im Compiler die Dateiende-Behandlung angestoßen; das Fortran-Programm wird übersetzt.
Anschließend wird der statische Binder BINDER aufgerufen: Der Compiler FOR1 wird entladen und BINDER geladen und gestartet. Es folgen Anweisungen an den BINDER.

Würde in diesem Beispiel der Kommandoaufruf /SEND-DATA *EOF entfallen, ergäbe sich folgender Verlauf:

– Die Dateneingabe wird durch das erste Kommando unterbrochen, ohne dass eine EOF-Bedingung den Compiler startet, da hier PROGRAM-INPUT=*MIXED-WITH-CMD angegeben wurde.
– Dieses Kommando (START-EXE $BINDER) wird ausgeführt, das heißt, der Compiler FOR1 wird entladen (ohne, dass vorher ein Compilations-Ergebnis geschrieben wird) und das Programm BINDER geladen.

Da keine Dateiende-Bedingung erzeugt wurde, wurde das Programm nicht übersetzt. BINDER kann also nicht auf ein neu übersetztes Programm zugreifen.

3.3 Fehlerbehandlung

In S-Prozeduren kann im Gegensatz zu Nicht-S-Prozeduren eine gezielte Fehlerbehandlung für Kommandos und Programmanweisungen durchgeführt werden. Die Returncodes von Anweisungen werden von SDF-P als Kommando-Returncodes interpretiert; im Folgenden wird nicht mehr zwischen Kommando- und Anweisungs-Returncode unterschieden (siehe auch Kommando BEGIN-BLOCK). SDF-P bietet dazu Kommandos, die auf eine Fehlersituation reagieren oder mit deren Hilfe ein Fehler analysiert werden kann. Außerdem gibt es Funktionen, die die Fehlerbehandlung unterstützen (siehe Kapitel „Funktionen“ auf Seite 229).

Die Fehlerbehandlung in S-Prozeduren ist blockorientiert, d.h., sie wird auf Blockebene durchgeführt. Bei ineinander geschachtelten Blöcken kann für jede Blockebene eine Fehlerbehandlung durchgeführt werden. Sie kann jedoch auch nur für übergeordnete, umgebende Blöcke aufgerufen werden. Dorthin werden auch die Fehler, die sich in inneren Blöcken ereigneten, durchgereicht und bearbeitet.

Ein Sonderfall blockorientierter Fehlerbehandlung ist die Fehlerbehandlung von einzelnen Kommandos: Hier wird das Kommando intern in einen BEGIN-Block gesetzt.

Die Fehlerbehandlung selbst wird in so genannten Fehlerbehandlungsblöcken durchgeführt. Zwei solcher Fehlerbehandlungsblöcke sind zu unterscheiden:

– IF-BLOCK-ERROR
– IF-CMD-ERROR

Daneben unterstützt SDF-P Fehleranalysen auf der Ebene der Kommando-Returncodes (mit dem Kommando SAVE-RETURNCODE und mit vordefinierten Funktionen).

Aus Kompatibilität zu Nicht-S-Prozeduren unterstützt SDF-P noch das Kommando SET-JOB-STEP.

Hinweis

Folgende Kommandos werden im Fehlerfall nicht ausgeführt:
ABEND, ABORT, CANCEL-PROCEDURE, END-PROCEDURE, ENDP, EXIT-JOB, EXIT-PROCEDURE, LOGOFF.

Im Fehlerfall können die Komponenten des Returncodes mit eingebauten Funktionen abgefragt werden (siehe Kapitel „Funktionen“ auf Seite 229). Wenn der Returncode auch dann geprüft werden soll, wenn kein Fehler auftrat, muss zuerst das Kommando SAVE-RETURNCODE aufgerufen werden. Nur dann steht der Returncode zur Verfügung und kann mit den vordefinierten Funktionen ausgewertet werden.

Die Fehlerbehandlung wird automatisch angestoßen, wenn eine Kommandoausführung einen Returncode mit einer Fehleranzeige zurückliefert. SDF-P springt dann zum nächsten IF-BLOCK-ERROR- oder IF-CMD-ERROR-Kommando.

SAVE-RETURNCODE

Mit dem Kommando SAVE-RETURNCODE wird der aktuelle Returncode gesichert. Der Returncode kann dann mit den vordefinierten Funktionen ausgewertet werden. Der Programmierer kann so auf Situationen reagieren, bei denen eine Kommandoausführung zwar ohne Fehler beendet wurde (Subcode1 = 0), Subcode2 und Maincode jedoch ausgewertet werden sollen.

3.3.1 Fehlerbehandlungsblöcke

Die Kommandos zur Fehlerbehandlung leiten IF-Blöcke ein, die wie „normale“ IF-Blöcke abgearbeitet werden. Zu diesen IF-Blöcken gehören dementsprechend auch ELSE- und END-IF-Kommandos.

IF-BLOCK-ERROR-Block

Wenn ein Kommando-Returncode mit einer Fehleranzeige zurückgeliefert wird, springt SDF-P automatisch zum nächsten IF-BLOCK-ERROR-Block, wobei andere Blöcke tieferer Schachtelungsebenen übersprungen werden.

Das IF-BLOCK-ERROR-Kommando kann auch aufgerufen werden, ohne dass ein Fehler aufgetreten ist. In diesem Fall wird der ELSE-Zweig des IF-BLOCK-ERROR-Blocks durchlaufen.

Wenn zwischen dem Kommando, das zur Fehlersituation führte, und dem Prozedurende kein IF-BLOCK-ERROR-Block vorhanden ist, wird die Prozedur beendet und der Fehler dem Aufrufer gemeldet: Der Fehlercode wird an den Aufrufer durchgereicht wie bei Prozedurbeendigung mit EXIT-PROCEDURE ERROR = *YES.

Beispiel

```
/BEGIN-BLOCK
/"Kommandos"
/  IF-BLOCK-ERROR
/  "Fehlerbehandlung"
/  ELSE
/  "Kein Fehler"
/  END-IF
/END-BLOCK
```

IF-CMD-ERROR-Block

Mit dem IF-CMD-ERROR-Block kann eine Fehlerbehandlung für das unmittelbar vorhergehende Kommando durchgeführt werden, jedoch nicht für Blockeinleitungs- oder Blockabschlusskommandos. Das heißt, IF-CMD-ERROR kann nicht auf die Kommandos FOR, IF, REPEAT oder WHILE oder die dazugehörenden Abschlusskommandos angewendet werden.

Hiermit ist eine gezielte Fehlerbehandlung für dieses Kommando möglich und eine Blockfehlerbehandlung wird somit verhindert.

Der IF-CMD-ERROR-Block kann wie der IF-BLOCK-ERROR-Block einen ELSE-Zweig enthalten, der durchlaufen wird, wenn das Kommando IF-CMD-ERROR aufgerufen wird und das vorhergehende Kommando fehlerfrei ausgeführt wurde.

Beispiel

```
/kommando
/IF-CMD-ERROR
/     "Fehlerbehandlung"
/ELSE
/     "Kein-Fehler-Kommandofolge"
/END-IF
```

Unabhängig von Fehler/Nicht-Fehler steht der Kommando-Returncode des dem IF-CMD-ERROR vorhergehenden Kommandos immer zur Verfügung. Dadurch können auch im Nicht-Fehlerfall sowohl im ELSE-Zweig als auch nach der Kommandofehlerbehandlung (END-IF) Warnungen ausgewertet werden.

3.3.2 Kommando-Returncodes

Die Fehlerbehandlung in S-Prozeduren basiert darauf, dass Kommandos und Anweisungen einen definierten Returncode liefern. Anhand dieses Returncodes kann SDF-P feststellen, ob bei der Kommandoausführung ein Fehler auftrat und welcher Fehler auftrat.

Der Returncode eines Kommandos besteht immer aus drei Komponenten:

Subcode1	Fehlerklasse
Subcode2	Zusatzinformation zu Subcode1
Maincode	Fehler-/Meldungsschlüssel

Welche Returncodes die SDF-P-Kommandos liefern, ist im Abschnitt „Kommando-Returncode“ auf Seite 563 beschrieben.

Die Komponenten des Returncodes können mit gleichnamigen vordefinierten Funktionen abgefragt werden: SUBCODE2( ), SUBCODE1( ) und MAINCODE( ); der Fehlermeldungstext kann mit der vordefinierten Funktion MSG( ) abgefragt werden (siehe Kapitel „Vordefinierte Funktionen“ auf Seite 353).

Wenn der Returncode auch dann geprüft werden soll, falls kein Fehler auftrat, muss zuerst das Kommando SAVE-RETURNCODE aufgerufen werden (außer bei IF-CMD-ERROR). Nur dann steht der Returncode zur Verfügung und kann mit den vordefinierten Funktionen ausgewertet werden.

Die Fehlerbehandlung wird automatisch angestoßen, wenn eine Kommandoausführung einen Returncode mit einer Fehleranzeige zurückliefert. SDF-P springt dann zum nächsten IF-BLOCK-ERROR- oder IF-CMD-ERROR-Kommando.

SAVE-RETURNCODE

Mit dem Kommando SAVE-RETURNCODE wird der aktuelle Returncode gesichert, sodass er mit den vordefinierten Funktionen ausgewertet werden kann. Der Programmierer kann so zum Beispiel auf Situationen reagieren, bei denen eine Kommandoausführung zwar ohne Fehler beendet wurde (Subcode1 = 0), Subcode2 und Maincode jedoch ausgewertet werden sollen.

Beispiel

Die folgende Kommandofolge bewirkt, dass in einer Prozedur der Returncode des vorhergehenden Kommandos im Nicht-Fehlerfall ausgewertet werden kann.

```
/"Kommando, das eine Warnung, aber keinen Fehler liefert"
/SAVE-RETURNCODE
/WRITE-TEXT &(TO-C-LIT(MSG(MAINCODE())))
```

Fehlerbehandlungsblöcke

Die Fehlerbehandlung in strukturierten Prozeduren ist blockorientiert, d.h. sie bezieht sich auf Kommandoblöcke.

Auch die Fehlerbehandlung selbst erfolgt in Blöcken, den Fehlerbehandlungsblöcken. Diese Blöcke sind ausführlich im Abschnitt „Prozedurrumpf erstellen“ auf Seite 93 beschrieben.

SET-JOB-STEP-Kommando

SDF-P unterstützt aus Kompatibilität zu Nicht-S-Prozeduren das Kommando SET-JOB-STEP (bzw. STEP), das aber in S-Prozeduren nicht verwendet werden sollte. Es hat die gleiche Wirkung wie ein leerer Fehlerbehandlungsblock (IF-BLOCK-ERROR; END-IF), setzt aber zusätzlich einige Auftragsschalter zurück. Außerdem sind mit SET-JOB-STEP Einschränkungen zu beachten (siehe Kapitel „Nicht-S-Prozeduren umstellen“).

3.3.3 Fehlerbedingung beim Einlesen von Datenzeilen

Beim Einlesen von Datenzeilen können folgende Fehler auftreten:

– Eine Prozedurzeile enthält Daten, obwohl ein Kommando erwartet wurde.
– Der Ausdruck kann weder durch eine Variable, noch durch eine Funktion ersetzt werden.
– Eine Prozedurzeile enthält ein einzeln stehendes Escape-Zeichen.

Wenn einer dieser Fehler auftritt, wird im Normalfall die Fehlerbehandlung angestoßen.

Die Kommandos SET-PROCEDURE-OPTIONS und MODIFY-PROCEDURE-OPTIONS bieten jedoch den Operanden DATA-ERROR-HANDLING, mit dem festgelegt werden kann, dass die Fehlerbehandlung nicht angestoßen wird (näheres siehe Abschnitt „Fehlerbehandlung auslösen“ auf Seite 85).

3.3.4 Fehlermeldungen

Wenn im Prozedurlauf Fehler auftreten, werden Fehlermeldungen ausgegeben, die den Fehler identifizieren. Im Dialog kann mit dem Kommando HELP-MSG-INFORMATION nähere Information angefordert werden.

Die von SDF-P ausgegebenen Fehlermeldungen sind im Kapitel „Meldungen“ auf Seite 811 zusammengefasst.

3.3.5 Durchreichen von Fehlern

„Durchreichen von Fehlern“ bedeutet, dass von einer untergeordneten Prozedur Informationen über aufgetretene Fehler an die übergeordnete Prozedur geliefert werden.

Dabei gibt es Unterschiede zwischen Prozeduren, die im Vordergrund aufgerufen werden, und Prozeduren, die im Hintergrund aufgerufen werden. Siehe dazu Kapitel „Prozeduren aufrufen und steuern“ auf Seite 107.

3.4 Prozedur-Compiler

In SDF-P wird ein Prozedur-Compiler angeboten, der die Portabiltät von S-Prozeduren auf eine Anlage garantiert, auf der das Subsystem SDF-P nicht geladen ist. Das heißt, der Prozedur-Compiler konvertiert S-Prozeduren in ein Zwischenformat, sodass sie allein mit SDF-P-BASYS ausgeführt werden können.

Dabei muss Folgendes beachtet werden:

Der Prozedur-Compiler konvertiert bzw. kompiliert nur die SDF-P-Kontrollstrukturen. In dem erzeugten Zwischenformat (auch kompilierte Prozedur oder Objektprozedur genannt) sind nur diese Kontrollstrukturen konvertiert, sonstige Kommandos und die Anweisungen sind identisch mit denen der Quellprozedur. Sie werden genauso ausgeführt wie in der ursprünglichen S-Prozedur.

Der Kompilier-Vorgang wird mit dem Kommando COMPILE-PROCEDURE gestartet. Das Kommando ist Teil des (kostenpflichtigen) Subsystems SDF-P. Das Endprodukt dieses Vorgangs ist das erwähnte Zwischenformat (die kompilierte Prozedur), das als Datei oder als Bibliothekselement abgespeichert werden kann. Die kompilierte Prozedur wird wie eine S-Prozedur mit den Kommandos CALL-PROCEDURE, INCLUDE-PROCEDURE oder ENTER-PROCEDURE aufgerufen. Allerdings kann im Gegensatz zu den anderen beiden Kommandos bei ENTER-PROCEDURE FROM-FILE=*LIBRARY-ELEMENT(...) kein Elementtyp angegeben werden. SDF-P-BASYS erwartet in diesem Fall, dass die kompilierte Prozedur als Element des Typs SYSJ oder J abgespeichert ist.

Hinweis

Auch S-Prozeduren, die keine oder nur bestimmte kostenpflichtige Funktionen von SDF-P nutzen, können mit COMPILE-PROCEDURE konvertiert und auf Anlagen ohne das Subsystem SDF-P ausgeführt werden.

Kompilierte Prozeduren sinnvoll anwenden

Kompilierte Prozeduren sollten nur dort angewendet werden, wo der kostenpflichtige Teil von SDF-P nicht geladen ist. In Installationen, in denen der kostenpflichtige Teil aber vorhanden und geladen ist und die S-Prozeduren auch regelmäßig aktualisiert werden, ist es nicht sinnvoll, kompilierte Prozeduren zu benutzen.

Da nur die SDF-P-Kontrollstrukturen kompiliert werden und der größere Teil der S-Prozedur erst zur Laufzeit interpretiert wird, ergeben sich folgende typische Anwendungsmodelle:

– Prozeduren mit kostenpflichtigen SDF-P-Funktionen können an einen Benutzer übergeben werden, der den kostenpflichtigen Teil von SDF-P nicht installiert hat; z.B. von einem externen Software-Entwickler.

– Ein Rechenzentrum mit verschiedenen BS2000-Rechnern besitzt nur für eine Anlage die Software-Lizenz für den kostenpflichtigen Teil von SDF-P:
  So können an dieser einen Anlage S-Prozeduren entwickelt, getestet und kompiliert werden und an den anderen diese kompilierten Prozeduren eingesetzt werden - obwohl bei letzteren der kostenpflichtige Teil von SDF-P fehlt.

– Ein Rechenzentrum hat nur im Rahmen eines Projekts eine Software-Lizenz für den kostenpflichtigen Teil von SDF-P, z.B. nur für die Zeit der Entwicklung bestimmter Prozeduren:
  So können während dieser Zeit die benötigten S-Prozeduren entwickelt, getestet und kompiliert werden; und auch nachdem die Lizenz für den kostenpflichtigen Teil von SDF-P erloschen ist, können die kompilierten S-Prozeduren benutzt werden.

3.5 SDF-P-Kommandos im Dialog

SDF-P-Kommandos können nicht nur in einer Prozedurumgebung aufgerufen werden, sie können auch im Dialog an der Datensichtstation eingegeben werden.

Generell ist die Anwendung von SDF-P im Dialog immer dann sinnvoll, wenn die Aufgabe in genau dieser Form nur einmal oder nur selten anfällt. Häufig wiederkehrende Aufgaben sollten in Prozeduren gelöst werden.

Beispiele für sinnvolle Anwendung von SDF-P-Kommandos im Dialog:

– Syntax von Kommandoaufrufen prüfen:
  Zum Beispiel beim Erstellen einer Prozedur; auf diese Weise lassen sich Syntaxfehler vermeiden oder die Korrektur von Syntaxfehlern vor dem nächsten Testlauf überprüfen.

– Vor einem Prozeduraufruf Variablen vorbesetzen, die dann innerhalb der aufgerufenen Prozedur verwendet werden:
  Zum Beispiel Variablen, die als Prozedurparameter übergeben werden.

– Im Dialog Schleifen programmieren:
  Wenn zum Beispiel mehrere Dateien nacheinander bearbeitet werden sollen.

– Langdauernde Programmläufe in einem Dialogblock starten, in dem der Programmlauf überwacht wird:
  Zum Beispiel langdauernde Übersetzungsläufe in Programm-Testphasen; die Übersetzung und Auswertung der Ergebnisse erfolgt automatisch; der Programmierer kann die Zeit für andere Aufgaben nutzen; er braucht nicht vor der Datensichtstation sitzen zu bleiben.

3.5.1 Regeln für die Eingabe von SDF-P-Kommandos im Dialog

Aufbau der Eingabezeilen

Es gelten für Kommandos generell die gleichen Syntaxregeln wie in einer Prozedur, Ausnahmen bilden der einleitende Schrägstrich und das Zeilenendezeichen.

Der einleitende Schrägstrich am Beginn der Eingabezeile kann entfallen.

Dies gilt sowohl, wenn einzelne SDF-P-Kommandos eingegeben werden, als auch in Kommandoblöcken.

Zeilenende

Eingabezeilen können mit der DUE-Taste abgeschlossen werden oder mit einem logischen Zeilenende (LZE-Taste). Generell gilt: Sobald die DUE-Taste gedrückt wird, gilt die Eingabe als abgeschlossen.

Wie eine Eingabezeile abgeschlossen werden muss, hängt von verschiedenen Faktoren ab:

– Besteht die Eingabe nur aus einer einzelnen Zeile, wird sie mit DUE abgeschlossen. Das Kommando wird analysiert und sofort ausgeführt.
– Wenn mehrere Eingabezeilen zusammenhängend eingegeben werden sollen, die Kommandos jedoch nicht in einen Kommandoblock eingebettet sind, muss jede Eingabezeile mit dem Zeichen *logisches Zeilenende* abgeschlossen werden (Taste LZE). Erst zum Abschluss der letzten Eingabezeile darf die DUE-Taste gedrückt werden.
– Wenn mehrere Eingabezeilen zusammenhängend eingegeben werden sollen und die Kommandos in einen Kommandoblock eingebettet sind, kann jede Zeile mit DUE abgeschlossen werden.
  Der Inhalt der gerade eingegebenen Zeile wird dann voranalysiert und zwischengespeichert, jedoch noch nicht ausgeführt. Sobald ein Fehler entdeckt wird (zum Beispiel eine falsche Folge von Kontrollstrukturkommandos), wird der Block abgebrochen. Erst wenn der Kommandoblock mit dem entsprechenden Blockabschlusskommando (und Taste DUE) beendet wird, wird der gesamte Kommandoblock ausgeführt.
  Wenn Kommandoblöcke im Dialog eingegeben werden, erscheint die Blockbezeichnung im Betriebssystem-Prompt. Erst wenn der Block beendet oder durch Fehler abgebrochen wird, erscheint wieder das normale Betriebssystem-Prompt.

Daten und Anweisungen

Damit innerhalb von Kommandoblöcken Daten von Kommandos unterschieden werden können, müssen Daten als Argumente des Kommandos SEND-DATA eingegeben werden (siehe Seite 744).

Wenn eine Zeile eine Anweisung enthalten soll, muss sie mit zwei Schrägstrichen „//“ beginnen oder als Argument des Kommandos SEND-STMT eingegeben werden.

Es muss vor dem Programmaufruf /ASSIGN-SYSDTA *SYSCMD gesetzt werden, damit das Programm die Daten oder Anweisungen aus dem Dialogblock liest.

Ausdrucksersetzung

Ausdrucksersetzung ist sowohl im geführten als auch im ungeführten Dialog möglich. Die Ersetzung erfolgt jedoch erst, wenn das Kommando ausgeführt wird.

Im Gegensatz zur &-Ersetzung in S-Prozeduren gilt, dass der zu ersetzende Ausdruck unverändert erhalten bleibt, wenn bei der Auswertung ein Fehler festgestellt wird.

Beispiel

```
/WRITE-TEXT '&(1 // 2)'
```

Der Ausdruck 1 // 2 ist fehlerhaft, da Zahlen nicht verkettet werden dürfen. In einer S-Prozedur wird daher die Fehlerbehandlung angestoßen. Im Dialog wird dagegen die Zeichenkette &(1 // 2) unverändert ausgegeben.

Im Dialog ist standardmäßig auch die Jobvariablenersetzung wirksam.

Unterbrechung/Abbruch

Mit der Taste K1 oder K2 kann die Eingabe innerhalb eines Kommandoblocks jederzeit abgebrochen werden. Es erscheint dann das normale Betriebssystem-Prompt.

Die Ausführung eines Kommandoblocks kann jederzeit mit der Taste K2 unterbrochen und mit RESUME-PROCEDURE fortgesetzt werden.

Geführter/ungeführter Dialog

Werden SDF-P-Kommandos einzeln im Dialog eingegeben, kann der geführte Dialog genutzt werden.

In Kommandoblöcken können Kommandos nur im ungeführten Dialog eingegeben werden; auch die Eingabe „kommando?“ wird zwischengespeichert. Der geführte Dialog wird erst bei Kommandoausführung eingeleitet. Vorwegeingaben werden dabei berücksichtigt.

Der geführte Dialog wird in Dialogblöcken nur wirksam, wenn mit dem Kommando MODIFY-SDF-OPTIONS PROCEDURE-DIALOGUE=*YES eingeschaltet wurde.

Ausdrucksersetzung kann im geführten Dialog genutzt werden.

3.5.2 Beispiel

Programm übersetzen und binden

Ein Programm COBOL.PROG soll übersetzt werden. Es handelt sich um ein sehr komplexes Programm, dessen Übersetzung relativ lange dauert. Der Programmierer möchte diese „Wartezeit“ für andere Aktivitäten nutzen und startet die Übersetzung in einem SDF-P-Kommandoblock im Dialog (er verzichtet darauf, eine Prozedur zu schreiben, weil er sich sicher ist, dass das Programm keine Fehler mehr enthält, und er daher genau diese Kommandofolge nicht wiederholt einzugeben braucht).

Der Programmierer würde folgende Kommandos im Dialog eingeben:

```
/BEGIN-BLOCK
/ASSIGN-SYSDTA *SYSCMD
/START-EXE $COBOL85, MONJV = MJV
/SEND-DATA '*COMOPT ...'
/SEND-DATA '*END'
/IF (SUBSTR(JV('MJV'), 1, 2) EQ '$T')
/START-LMS
//...
//...
//END
/ELSE; PRINT-DOC UEBERS.LISTE
/END-IF
/END-BLOCK
```

Da die gesamte Kommandofolge in einen BEGIN-Block eingebettet wird, kann jede Zeile mit DUE abgeschlossen werden.

Die Übersetzung des Programms soll über die Monitor-JV MJV überwacht werden (MONJV = MJV).

Nach Abschluss der Übersetzung soll der Inhalt der Monitor-JV ausgewertet und in der IF-Abfrage geprüft werden. Wenn der Inhalt der Monitor-JV „$T“ ist, ist die Übersetzung fehlerfrei abgelaufen, das Programm kann in die Programmbibliothek eingebracht werden. Anschließend wird das Dienstprogramm LMS aufgerufen.

Wenn die Übersetzung nicht fehlerfrei war (Inhalt von MJV ist ungleich $T), soll das Übersetzungsprotokoll ausgedruckt werden.

Mit den Kommandos END-IF und END-BLOCK werden die beiden ineinander geschachtelten Kommandoblöcke abgeschlossen. Nach END-BLOCK wird die Ausführung des Kommandoblocks angestoßen.

3.6 Aufruf von Kommandofolgen aus Programmen

Das Kommando INCLUDE-CMD bietet die Möglichkeit, Kommandofolgen oder eine Prozedur aus einem Programm aufzurufen. Die auszuführenden Kommandos werden im Operanden CMD als Wert übergeben. INCLUDE-CMD besitzt dadurch dieselbe Funktionalität wie das Kommando INCLUDE-PROCEDURE (siehe Seite 695), allerdings mit folgenden Einschränkungen:

– Die auszuführenden Kommandos werden nicht von einer Datei eingelesen, sondern direkt im Operanden CMD angegeben, der die Rolle einer virtuellen SYSCMD-Datei übernimmt. Somit bietet sich in Verbindung mit dem CMD-Makroaufruf die Möglichkeit, eine Kommando-Prozedur im Arbeitsspeicher bereitzustellen.

– INCLUDE-CMD unterstützt keinen Operanden von INCLUDE-PROCEDURE. Es werden ausschließlich die Voreinstellungen benutzt.

– Im Gegensatz zu INCLUDE-PROCEDURE beendet INCLUDE-CMD ein Programm nicht, wenn es durch den Makro CMD aufgerufen wurde. Das Programm wird auch nicht beendet, wenn eine Prozedur in der Kommandofolge aufgerufen wird, die im Operanden CMD angegeben ist (siehe Beispiel Seite 694).

– INCLUDE-CMD darf im CMD-Makro (TU-Programm) und in EXECUTE-SYSTEM-CMD-Anweisungen ausgeführt werden.

Während der Ausführung von INCLUDE-CMD werden vom System folgende Operationen zurückgewiesen, um mögliche Inkonsistenzen beim Aufruf im Programm-Modus auszuschalten:

– Programm-Start und Programm-Beendigung: START-Hilfsprogramm, LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM), RESTART-PROGRAM (vgl. CALL-PROCEDURE ...UNLOAD-ALLOWED=*NO).

– Programm-Wiederaufnahme: AID-Kommandos, RESUME-PROGRAM, EXIT-PROCEDURE mit RESUME-PROGRAM=*YES, ENDP-RESUME, INFORM-PROGRAM (bzw. SEND-MSG ...,TO=*PROGRAM)

– Prozedur-Abbruch: CANCEL-PROCEDURE, K2 beim Prompting von Parametern.

– Rekursiver Aufruf von INCLUDE-CMD.

– BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD

– SET-JOB-STEP, wenn ein Programm geladen ist.

4 S-Prozeduren erstellen

In diesem Kapitel wird beschrieben, wie man im Rahmen der Erstellung von S-Prozeduren den Prozedurkopf, den Prozedurrumpf sowie Sprünge richtig definiert.

4.1 Prozedurkopf erstellen

Der Prozedurkopf besteht aus dem Kommando SET-PROCEDURE-OPTIONS, mit dem die Prozedureigenschaften definiert werden, und dem DECLARE-PARAMETER-Block, in dem die Prozedurparameter deklariert werden.

Der Prozedurkopf steht immer am Anfang der Prozedur. Beim Prozeduraufruf werden zunächst die Kommandos des Prozedurkopfs ausgeführt und so die Prozedureigenschaften eingestellt, bevor die Kommandos des Prozedurrumpfs analysiert und verarbeitet werden (siehe Kapitel „Prozeduren aufrufen und steuern“ auf Seite 107).

In den folgenden Abschnitten wird zunächst beschrieben, wie die Prozedurparameter eingestellt werden und welche Standardeigenschaften verwendet werden können. Daran anschließend wird erläutert, wie Prozedurparameter deklariert werden.

4.1.1 Prozedureigenschaften einstellen

Die Prozedureigenschaften werden zunächst im Prozedurkopf mit dem Kommando SET-PROCEDURE-OPTIONS eingestellt. Wenn beim Ablauf der Prozedur andere Eigenschaften gelten sollen, können sie später mit dem Kommando MODIFY-PROCEDURE-OPTIONS geändert werden (Ausnahme: die SYSFILE-Umgebung).

Das Kommando SET-PROCEDURE-OPTIONS muss als erstes Kommando im Prozedurkopf stehen. Wird SET-PROCEDURE-OPTIONS ohne Operanden verwendet, so gelten die durch die aktuelle Syntaxdarstellung eingestellten Defaultwerte. Wenn die im Handbuch beschriebenen Defaultwerte für die Prozedureigenschaften verwendet werden sollen, muss SET-PROCEDURE-OPTIONS entfallen.

Im Kommando SET-PROCEDURE-OPTIONS können folgende Eigenschaften eingestellt werden:

- mit welchem Kommando die Prozedur aufgerufen werden darf
- ob eine eigene SYSFILE-Umgebung eingerichtet werden soll
- in welcher Länge die Sätze der Prozedur ausgewertet werden
- ob und was protokolliert werden darf
- ob die Prozedur unterbrechbar ist
- ob eine Fehlerbehandlung ausgelöst wird, wenn statt eines Kommandos Daten eingelesen werden
- welches Zeichen als Escape-Zeichen in Datensätzen gelten soll
- ob implizite Deklaration von Variablen erlaubt ist
- ob Jobvariablen bei der Ausdrucksersetzung berücksichtigt werden
- ob die Fehlerbehandlung kompatibel zum Spin-Off-Verhalten oder in Abhängigkeit vom Returncode ausgelöst werden soll
- ob bestimmte SDF-P-Meldungen unterdrückt werden sollen

Jeder dieser Prozedureigenschaften entspricht ein Operand im Kommando SET-PROCEDURE-OPTIONS (bzw. MODIFY-PROCEDURE-OPTIONS). In den folgenden Abschnitten werden die Bedeutung dieser Eigenschaften und die Funktion der entsprechenden Operanden kurz erläutert.

4.1.1.1 Aufrufkommando festlegen

Es gibt drei SDF-P-Kommandos für den Aufruf von S-Prozeduren: CALL-PROCEDURE, INCLUDE-PROCEDURE und ENTER-PROCEDURE. Das Kommando für den Prozeduraufruf wird durch den Operanden CALLER des Kommandos SET-PROCEDURE-OPTIONS beeinflusst. CALLER bietet als Operandenwerte *ANY, *CALL und * INCLUDE:

- Ist *ANY gesetzt, darf die Prozedur mit jedem der Aufrufkommandos aufgerufen werden. *ANY ist Standardeinstellung.
- Ist *CALL gesetzt, darf die Prozedur mit CALL-PROCEDURE oder mit ENTER-PROCEDURE aufgerufen werden.
- Ist *INCLUDE gesetzt, darf die Prozedur nur mit INCLUDE-PROCEDURE aufgerufen werden.

Näheres zum Prozeduraufruf siehe Kapitel „Prozeduren aufrufen und steuern“ auf Seite 107.

4.1.1.2 SYSFILE-Umgebung festlegen

Im Operanden SYSTEM-FILE-CONTEXT des Kommandos SET-PROCEDURE-OPTIONS wird die Systemdatei-Umgebung eingestellt, in der die Prozedur ablaufen soll.

Unter dem Begriff SYSFILE-Umgebung werden die Systemdateien des BS2000 zusammengefasst, gleich ob sie ihre Primärzuweisung haben oder einer BS2000-Datei zugewiesen sind.

Prozeduren können in einer eigenen SYSFILE-Umgebung ablaufen oder die Systemdatei-Umgebung des Aufrufers übernehmen. Dies wird eingestellt mit dem Operanden SYSTEM-FILE-CONTEXT.

SYSTEM-FILE-CONTEXT bietet als Operandenwerte *STD, *OWN und *SAME-AS-CALLER:

– Ist *STD gesetzt, erhält die Prozedur eine eigene SYSFILE-Umgebung, in der für alle Systemdateien außer SYSDTA die Zuweisungen des Aufrufers übernommen werden. SYSDTA ist standardmäßig der Prozedur (also *SYSCMD) zugewiesen.

– Ist *SAME-AS-CALLER gesetzt, übernimmt die Prozedur die Systemdatei-Umgebung des Aufrufers, einschließlich der Zuweisung für SYSDTA. Werden Zuweisungen innerhalb der Prozedur verändert, wirkt sich dies entsprechend auf die Systemdatei-Umgebung des Aufrufers aus.

– Ist *OWN gesetzt, wird für die Prozedur eine eigene Systemdatei-Umgebung eingerichtet, in der alle Zuweisungen des Aufrufers gelten, einschließlich der für SYSDTA. Werden Zuweisungen innerhalb der Prozedur geändert, wirkt sich dies jedoch nicht auf die Systemdatei-Umgebung des Aufrufers aus.

Die Einstellung für die Systemdatei-Umgebung gilt für die gesamte Prozedur und kann mit MODIFY-PROCEDURE-OPTIONS nicht geändert werden.

4.1.1.3 Länge der Prozedurzeilen definieren

Im Operanden INPUT-FORMAT des Kommandos SET-PROCEDURE-OPTIONS wird festgelegt, an welcher Stelle der Prozedurzeile das Fortsetzungszeichen stehen muss, wenn die Kommando- bzw. Anweisungsfolge mehrere Zeilen lang ist.

INPUT-FORMAT bietet als Operandenwerte *FREE-RECORD-LENGTH und *BY-SDF-OPTION.

Ist *FREE-RECORD-LENGTH gesetzt, kann das Fortsetzungszeichen an einer beliebigen Stelle in der Prozedurzeile stehen. Das Fortsetzungszeichen muss aber das letzte relevante Zeichen der Zeile sein. *FREE-RECORD-LENGTH ist die Standardeinstellung.

Ist BY-SDF-OPTION gesetzt, hängt die Positionierung des Fortsetzungszeichens von der Einstellung des Operanden CONTINUATION im Kommando MODIFY-SDF-OPTIONS ab. Gilt CONTINUATION =*NEW-MODE, darf das Fortsetzungszeichen zwischen Spalte 2 und 72 stehen. Gilt CONTINUATION = *OLD-MODE, muss das Fortsetzungszeichen auf Spalte 72 stehen.

Die Einstellung *BY-SDF-OPTION gewährleistet die Kompatibilität von Nicht-S- und S-Prozeduren und erleichert so die Umstellung.

Die Einstellung für den Operanden INPUT-FORMAT gilt für die gesamte Prozedur und kann mit MODIFY-PROCEDURE-OPTIONs nicht geändert werden.

4.1.1.4 Protokollierung einstellen

Im Operanden LOGGING-ALLOWED des Kommandos SET-PROCEDURE-OPTIONS wird zum einen eingestellt, ob die Prozedur protokolliert werden darf. Zum anderen kann festgelegt werden, ob Kommandos und/oder Daten protokolliert werden sollen.

LOGGING-ALLOWED bietet als Operandenwerte *YES, *NO und *PARAMETERS:

- Ist LOGGING-ALLOWED=*YES gesetzt, dürfen sowohl Kommandos als auch Daten protokolliert werden.
- Ist *NO gesetzt, darf nicht protokolliert werden.
- Ist *PARAMETERS gesetzt, kann unterschieden werden, ob Kommandos oder Daten protokolliert werden dürfen. *PARAMETERS ist die Standardeinstellung.

Die Einstellungen für die Protokollierung können im Kommando MODIFY-PROCEDURE-OPTIONS geändert werden.

Ausgelöst wird die Protokollierung durch die entsprechende Angabe im Operanden LOGGING des Aufrufkommandos.

4.1.1.5 Unterbrechbarkeit der Prozedur festlegen

Im Operanden INTERRUPT-ALLOWED des Kommandos SET-PROCEDURE-OPTIONS wird festgelegt, ob der Prozedurablauf mit der Funktionstaste K2 unterbrochen werden darf.

INTERRUPT-ALLOWED bietet als Operandenwerte *YES und *NO:

- Ist *YES gesetzt, darf die Prozedur unterbrochen werden. Die Prozedur wird als unterbrechbar bezeichnet. Sie kann dann nach einer Unterbrechung mit dem Kommando RESUME-PROCEDURE fortgesetzt werden. *YES ist die Standardeinstellung.
- Ist *NO gesetzt, ist die Prozedur nicht unterbrechbar. Die K2-Taste hat keine Wirkung.

Wie sich Prozedurunterbrechung auswirkt, ist im Abschnitt „Prozedur unterbrechen“ auf Seite 128 beschrieben.

4.1.1.6 Fehlerbehandlung auslösen

Im Operanden DATA-ERROR-HANDLING wird festgelegt , ob in Datensätzen eine Fehlerbehandlung ausgelöst werden soll, wenn eine Fehlersituation vorliegt.

Die Standardeinstellung ist *YES. Das heißt, eine Fehlerbehandlung wird ausgelöst, wenn eine Prozedur statt Kommandos Daten einliest, oder wenn in Daten eine Ausdrucksersetzung nicht aufgelöst werden kann.

Ist *NO gesetzt, wird keine Fehlerbehandlung ausgelöst. Die Einstellung *NO ist kompatibel zum Spin-Off-Verhalten von Nicht-S-Prozeduren. Sie erleichtert die Umstellung von Nicht-S-Prozeduren auf S-Prozeduren.

4.1.1.7 Art der Fehlerbehandlung einstellen

Die Fehlerbehandlung kann entweder kompatibel zum bisherigen Spin-Off-Verhalten ausgelöst werden oder auf der Basis der Kommando-Returncodes. Sie wird über Fehlerbehandlungsblöcke durchgeführt, die im Prozedurrumpf definiert werden.

Im Operanden ERROR-MECHANISM des Kommandos SET-PROCEDURE-OPTIONS kann die Art der Fehlerbehandlung eingestellt werden. Als Operandenwerte stehen *SPIN-OFF-COMPATIBLE und *BY-RETURNCODE zur Verfügung:

– Ist *SPIN-OFF-COMPATIBLE gesetzt, wird die Fehlerbehandlung kompatibel zum Spin-Off-Verhalten ausgelöst. Subcode1 wird nicht berücksichtigt. Damit wird sichergestellt, dass das Fehlerverhalten von S-Prozeduren, die bereits in BS2000 V10.0 erstellt wurden, kompatibel bleibt. *SPIN-OFF-COMPATIBLE ist die Standardeinstellung.

– Ist *BY-RETURNCODE gesetzt, wird die Fehlerbehandlung ausgelöst, wenn der Subcode1 des letzten Kommando-Returncodes ungleich Null ist. Das Spin-Off-Verhalten wird nicht berücksichtigt. Die Einstellung *BY-RETURNCODE ist geeignet für Prozeduren, in denen Kommandos mit eigenen Kommando-Returncodes verwendet werden. Dadurch wird eine differenzierte Programmierung möglich.

Beispiel 1 (für ERROR-MECHANISM=*SPIN-OFF-COMPATIBLE)

```
/SET-PROCEDURE-OPTIONS ERROR-MECHANISM=*SPIN-OFF-COMPATIBLE
/...
/PRINT-FILE DOES-NOT-EXIST
/...
/
/IF-BLOCK-ERROR
/                  "IF-BLOCK-ERROR wird nicht angestoßen, da"
/                  "/PRINT-FILE (mit korrekter Syntax) keinen Spin-Off"
/                   "erzeugt"
/END-IF
/...
```

Beispiel 2 (für ERROR-MECHANISM=*BY-RETURNCODE)

```
/SET-PROCEDURE-OPTIONS ERROR-MECHANISM=*BY-RETURNCODE
/...
/PRINT-FILE DOES-NOT-EXIST
/...
/IF-BLOCK-ERROR
/                    "IF-BLOCK-ERROR wird gemäß den Returncodes"
/                    "für /PRINT-FILE angestoßen"
/END-IF
/...
```

Fehlerverhalten bei Versionswechsel

Obwohl das Verhalten der meisten Kommandos mit der jeweiligen Version zusammenhängt, ist eine Umstellung eines Kommandos auf definierte Kommando-Returncodes nicht von einer BS2000-Version abhängig, sondern von der Version des Systemprodukts bzw. Subsystems.

Bei ERROR-MECHANISM=*BY-RETURNCODE kann sich durch Umstellung eines Systemprodukts auf Kommando-Returncodes das Fehlerverhalten der betreffenden Kommandos in derselben BS2000-Version ändern.

Fehlerverhalten bei Programmen

Die Fehlerbehandlung betrifft auch Anwenderprogramme; diese können eigene Returncodes liefern. Für Programme, die keinen eigenen Returncode liefern, simuliert SDF einen Returncode analog der Spin-Off-Behandlung.

Standardmäßig erfolgt eine Fehlerbehandlung durch SDF-P erst nach Programmbeendigung. Durch die Voreinstellung *SPIN-OFF-COMPATIBLE erfolgt die Fehlerbehandlung nicht auf Basis des Returncodes (siehe Makro CMDRC im Handbuch „SDF-A“ [16]), sondern durch den Spin-Off auslösenden Parameter UNIT=STEP des Makros TERM (siehe Handbuch „Makroaufrufe an den Ablaufteil“ [7]).

Die Einstellung PROPAGATE-STMT-RC=*TO-CMD-RC in einem BEGIN-Block (siehe Kommando BEGIN-BLOCK) ermöglicht die SDF-P-Fehlerbehandlung und Auswertung von Returncodes während des Programmlaufs. Die SDF-P-Fehlerbehandlung erfolgt dann unabhängig von der Einstellung für ERROR-MECHANISM nur auf Basis des Returncodes.

4.1.1.8 Escape-Zeichen in Datensätzen definieren

Im Operanden DATA-ESCAPE-CHARACTER des Kommandos SET-PROCEDURE-OPTIONS wird festgelegt, welches Zeichen als Escape-Zeichen in Datensätzen für die Ausdrucksersetzung verwendet wird. DATA-ESCAPE-CHARACTER bietet als Operandenwerte *NONE, *STD, &, #, *, @ und $:

- *NONE ist die Standardeinstellung, d.h., es gibt kein Escape-Zeichen. (Allerdings **muss** in der Regel bei Dienstprogrammen ohne SDF-Oberfläche ein Escape-Zeichen vereinbart werden, falls in Datensätzen eine Ersetzung stattfinden soll.)
- Ist *STD gesetzt, gilt das Zeichen & als Escape-Zeichen.
- Mit &, #, *, @ und $ wird das jeweilige Zeichen als Escape-Zeichen definiert.

Näheres siehe Abschnitt „Ausdrucksersetzung“ auf Seite 55.

4.1.1.9 Implizite Deklaration von Variablen einstellen

Im Operanden IMPLICIT-DECLARATION des Kommandos SET-PROCEDURE-OPTIONS wird festgelegt, ob einfache Variablen implizit deklariert werden können. Standardeinstellung ist IMPILICIT-DECLARATION=*YES. Das bedeutet, dass einfache Variablen bei der ersten Zuweisung deklariert werden, ohne dass sie explizit mit dem Kommando DECLARE-VARIABLE deklariert wurden.

Ist IMPLICIT-DECLARATION=*NO gesetzt, müssen einfache Variablen explizit deklariert werden.

Zusammengesetzte Variablen müssen immer explizit deklariert werden. Nähere Informationen hierzu enthält das Kapitel „Variablen in S-Prozeduren verwenden“ auf Seite 139.

4.1.1.10 Jobvariablen-Ersetzung einstellen

Im Operanden JV-REPLACEMENT des Kommandos SET-PROCEDURE-OPTIONS wird festgelegt, ob bei der Ausdrucksersetzung auch Jobvariablen ersetzt werden sollen. JV-REPLACEMENT bietet als Operandenwerte *NONE und *AFTER-BUILTIN-FUNCTION:

- Standardeinstellung ist *NONE. Das bedeutet, dass bei der Ausdrucksersetzung Namen nicht als Jobvariablennamen interpretiert werden.
- Ist *AFTER-BUILTIN-FUNCTION gesetzt, werden bei der Ausdrucksersetzung Namen als Jobvariablennamen interpretiert.

Die Einstellungen können im Kommando MODIFY-PROCEDURE-OPTIONS geändert werden.

4.1.1.11 Ausgewählte SDF-P-Meldungen unterdrücken

Im Operanden SUPPRESS-SDP-MSG des Kommandos SET-PROCEDURE-OPTIONS wird festgelegt, ob die Ausgabe bestimmter SDF-P-Meldungen (aus der Meldungsklasse SDP) unterdrückt werden soll. Die Einstellung gilt nur in der aufrufenden Prozedur (wird nicht weitervererbt). SUPPRESS-SDP-MSG bietet als Operandenwerte *NONE und <structured-name 7..7>:

– Standardeinstellung ist *NONE. Das bedeutet, dass die Meldungsausgabe nicht unterdrückt wird und alle SDF-P-Meldungen ausgegeben werden.

– Bei expliziter Angabe eines Meldungsschlüssels aus der Meldungsklasse SDP wird die Ausgabe dieser Meldung unterdrückt. In einer Liste können auch mehrere Meldungsschlüssel angegeben werden.

Die Einstellungen können im Kommando MODIFY-PROCEDURE-OPTIONS geändert werden:

– SUPPRESS-SDP-MSG=*NONE schaltet die Meldungsunterdrückung aus und es werden wieder alle SDF-P-Meldungen ausgegeben.

– Mit SUPPRESS-SDP-MSG=*ADD(...) kann eine oder mehrere Meldungen in die Liste der zu unterdrückenden Meldungen aufgenommen werden.

– Mit SUPPRESS-SDP-MSG=*REMOVE(...) kann eine oder mehrere Meldungen aus die Liste der zu unterdrückenden Meldungen wieder entfernt werden.

4.1.2 Defaultwerte für Prozedureigenschaften übernehmen

Es gibt zwei verschiedene Arten von Defaultwerten:

– die global in der Syntaxdatei eingestellten Defaultwerte
– die SDF-P-spezifischen Defaultwerte (siehe Kommandobeschreibung des Kommandos SET-PROCEDURE-OPTIONS, Seite 747).

Wenn für die Prozedur die Defaultwerte aus den Syntaxdateien übernommen werden sollen, muss der Prozedurkopf mit dem Kommando SET-PROCEDURE-OPTIONS ohne Operandenwerte beginnen. Wenn Prozedurparameter deklariert werden sollen, folgt dann die Deklaration der Parameter.

Wenn für die Prozedur die SDF-P-spezifischen Defaultwerte übernommen werden sollen, muss das Kommando SET-PROCEDURE-OPTIONS entfallen. Wenn Prozedurparameter deklariert werden sollen, beginnt der Prozedurkopf dann mit der Deklaration der Parameter. Wenn keine Prozedurparameter deklariert werden sollen, beginnt die Prozedur mit dem ersten Kommando des Prozedurrumpfs. Der Prozedurkopf gilt dann als implizit deklariert.

SDF V4.1 ermöglicht die Definition taskspezifischer Default-Werte. Diese Default-Werte können auch In S-Prozeduren definiert werden; sie sind aber nur bei Dialogeingaben wirksam. Im Prozedurmodus werden sie ignoriert, dort gelten die in den Syntaxdateien festgelegten Defaultwerte. Taskspezifische Defaultwerte dürfen nicht in Kontrollflusskommandos und im Prozedurkopf angegeben werden. Zu taskspezifischen Defaultwerten siehe Handbuch „SDF“ [20].

4.1.3 Prozedurparameter deklarieren

Damit beim Aufruf Parameter an die aufgerufene Prozedur übergeben werden können, müssen diese Parameter in der aufgerufenen Prozedur deklariert sein.

Prozedurparameter werden im DECLARE-PARAMETER-Block mit dem Kommando DECLARE-PARAMETER deklariert. Der DECLARE-PARAMETER-Block braucht nur aus einem DECLARE-PARAMETER-Kommando zu bestehen, wenn alle Prozedurparameter in diesem Kommando deklariert werden können.

Wenn für die Parameterdeklaration mehrere DECLARE-PARAMETER-Kommandos benötigt werden, muss der DECLARE-PARAMETER-Block mit dem Kommando BEGIN-PARAMETER-DECLARATION eingeleitet und mit dem Kommando END-PARAMETER-DECLARATION abgeschlossen werden. Zwischen diesen beiden Kommandos können beliebig viele DECLARE-PARAMETER-Kommandos stehen.

Neben Prozedurparametern können innerhalb von DECLARE-PARAMETER-Blöcken auch Variablenbehälter mit OPEN-VARIABLE-CONTAINER geöffnet werden, die innerhalb des DECLARE-PARAMETER-Blocks Variablen zur Verfügung stellen, mit deren Hilfe die Anfangswerte bei Prozedurparametern eingestellt werden können. Die OPEN-VARIABLE-CONTAINER-Kommandos müssen vor dem ersten DECLARE-PARAMETER-Kommando geschrieben werden.

Im DECLARE-PARAMETER-Kommando können folgende Eigenschaften der Prozedurparameter eingestellt werden:

– Parametername
– Anfangswert (falls angegeben)
– Datentyp
– Art der Parameterübergabe

Jede dieser Eigenschaften wird über einen Operanden des DECLARE-PARAMETER-Kommandos definiert.

4.1.3.1 Parametername definieren

Der Parametername wird mit dem Operanden NAME definiert. Da Prozedurparameter Variablen sind gelten für Parameternamen die gleichen Regeln wie für Variablennamen, d.h. es können einfache Namen oder zusammengesetzte Namen verwendet werden (siehe Abschnitt „Variablennamen“ auf Seite 155).

Der Parametername kann dann im Prozeduraufruf als Schlüsselwort für die Parameterübergabe angegeben werden.

4.1.3.2 Datentyp festlegen

Der Datentyp wird im Operanden TYPE des Kommandos DECLARE-PARAMETER definiert. Für Prozedurparameter gelten die gleichen Datentypen wie für Variablen: *ANY, *STRING, *INTEGER und *BOOLEAN. Der hier eingestellte Datentyp muss bei der Parameterübergabe und bei der Zuweisung eines Anfangswertes beachtet werden.
Wird TYPE = *ANY gewählt, wird der Parameterwert als *STRING interpretiert, unabhängig davon, ob er beim Aufruf mit oder ohne Hochkommas angegeben wird.

4.1.3.3 Anfangswert zuweisen

Im Operanden INITIAL-VALUE des Kommandos DECLARE-PARAMETER wird festgelegt, ob dem Prozedurparameter ein Anfangswert zugewiesen werden soll. INITIAL-VALUE bietet als Operandenwerte *NONE, *PROMPT und die direkte Angabe eines Anfangswertes:

– *NONE ist die Standardeinstellung. Das bedeutet, dass dem Parameter kein Anfangswert zugewiesen wird. Dem Parameter muss aber beim Aufruf der Prozedur ein Wert zugewiesen werden.

– Ist *PROMPT gesetzt, wird der Wert bei der Ausführung der Prozedur abgefragt, sofern er nicht beim Prozeduraufruf übergeben wurde. Er kann aber auch gezielt über READ-VARIABLE angefordert werden.

Beispiel

```
/SET-PROCEDURE-OPTIONS
/DECLARE-PARAMETER NAME(INITIAL-VALUE=*PROMPT)
/...
/IF (NOT IS-INITIALIZED('NAME'))
/ WRITE-TEXT 'BITTE GEBEN SIE IHREN NAMEN EIN!'
/ READ-VARIABLE NAME,INPUT=*TERMINAL
/END-IF
```

Als dritte Möglichkeit kann bei INITIAL-VALUE direkt ein Anfangswert angegeben werden. Der angegebene Wert muss zum Datentyp des Prozedurparameters passen. Der Anfangswert gilt, wenn beim Aufruf kein anderer Wert übergeben wird.

4.1.3.4 Art der Parameterübergabe festlegen

Beim Prozeduraufruf kann als Argument des Parameters direkt ein Wert übergeben werden oder der Name einer Variablen, die den entsprechenden Wert enthält. In der aufgerufenen Prozedur muss daher bei der Parameterdeklaration festgelegt werden, wie dieses Argument interpretiert wird. Dies geschieht im Operanden TRANSFER-TYPE des Komamndos SET-PROCEDURE-OPTIONS:

– Gilt TRANSFER-TYPE = *BY-VALUE wird der Wert, der im Prozeduraufruf übergeben wird, direkt dem Prozedurparameter zugewiesen; war der Prozedurparameter zuvor bereits mit einem anderen Wert initialisiert, wird er überschrieben. TRANSFER-TYPE= *BY-VALUE ist die Standardeinstellung. *BY-VALUE muss gesetzt sein, damit bei einer im Hintergrund aufgerufenen Prozedur Parameter übergeben werden können.

– Gilt TRANSFER-TYPE = *BY-REFERENCE wird der übergebene Wert als Name einer Variablen interpretiert, die der Aufrufer deklariert und initialisiert hat. Diese Variable dient dann als Behälter des Prozedurparameters.

Wie sich die Parameterübergabe beim Prozeduraufruf auswirkt, ist im Abschnitt „Prozedurparameter übergeben“ auf Seite 108, beschrieben.

4.1.3.5 Mit permanenten Variablen Prozedurparameter initialisieren

Vor der ersten Deklaration eines Parameters kann mit dem Kommando OPEN-VARIABLE-CONTAINER im Prozedurkopf innerhab des ersten DECLARE-PARAMETER-Blocks ein Variablenbehälter mit permanenten Variablen geöffnet werden. Dadurch können mit den darin deklarierten Variablen die jeweiligen Prozedurparameter initialisiert werden. (Näheres siehe Abschnitt „Variablenbehälter“ auf Seite 168).

4.2 Prozedurrumpf erstellen

Der Prozedurrumpf schließt direkt an den Prozedurkopf an. Er besteht aus einer Folge von Kommandos, Anweisungen und Daten, die beim Ablauf der Prozedur ausgeführt wird. Der Prozedurablauf kann über Kontrollstrukturen und Sprungkommandos gesteuert werden.

Zu den Kontrollstrukturen zählen einfache Kommandoblöcke, Schleifen und Verzweigungen. Zu den Verzweigungen gehören auch die Fehlerbehandlungsblöcke.

Die Kontrollstrukturen werden jeweils von einem Einleitungskommando eingeleitet und von einem Abschlusskommando abgeschlossen.

Kontrollstrukturen können auch geschachtelt werden.

Mit Hilfe von Sprungkommandos kann das sequenzielle Abarbeiten von Kommandos unterbrochen und an eine definierte Prozedurstelle gesprungen werden.

In den folgenden Abschnitten werden zunächst Definition und Einsatz von Kontrollstrukturen und anschließend die Verwendung von Sprungkommandos beschrieben.

4.2.1 Einfache Kommandoblöcke definieren

In einem einfachen Kommandoblock können Kommandos zu einer logischen Einheit zusammengefasst werden. Der Block wird mit dem Kommando BEGIN-BLOCK eingeleitet und mit dem Kommando END-BLOCK abgeschlossen. Dazwischen stehen die Kommandos, die in diesem Block ausgeführt werden sollen. Einfache Kommandoblöcke werden auch als BEGIN-Blöcke bezeichnet.

Die Kommandozeile des Kommandos BEGIN-BLOCK kann - entsprechend den Regeln für S-Marken - mit einer Marke eingeleitet werden. Über diese Marke wird das END-BLOCK-Kommando direkt dem BEGIN-BLOCK-Kommando zugeordnet; die Marke kann auch von anderen Kommandos als Sprungziel verwendet werden.

Wenn die BEGIN-BLOCK-Kommandozeile keine Marke enthält, tritt die implizite Zuordnung zwischen BEGIN- und END-BLOCK-Kommandos in Kraft: Das END-BLOCK-Kommando bezieht sich implizit immer auf das letzte BEGIN-BLOCK-Kommando, das noch nicht durch END-BLOCK abgeschlossen ist.

Aufbau des BEGIN-Blocks ohne Marke

```
/BEGIN-BLOCK
[kommandofolge]
/END-BLOCK
```

Aufbau des BEGIN-Blocks mit Marke

```
/marke: BEGIN-BLOCK
[kommandofolge]
/END-BLOCK [BLOCK = marke]
```

4.2.2 Verzweigungen definieren

Verzweigungen im Prozedurlauf können mit IF-Blöcken erzeugt werden. In IF-Blöcken werden Kommandofolgen abhängig vom Ergebnis einer Bedingung ausgeführt. Ein IF-Block wird mit dem Kommando IF eingeleitet und mit dem Kommando END-IF abgeschlossen. Er kann mit einer Marke versehen werden.

Wenn nur eine Bedingung geprüft werden soll, besteht der IF-Block aus dem IF-Kommando mit der Bedingung, der dazugehörenden Kommandofolge und dem Kommando END-IF. Die Bedingung muss als boolescher Wert angegeben werden.

```
/IF bedingung
  kommandofolge1


/END-IF
```

Wenn die Bedingung im IF-Kommando erfüllt ist, wird $\texttt{kommandofolge}_1$ ausgeführt. Wenn die Bedingung nicht erfüllt ist, wird der IF-Block beendet. Das Kommando, das auf END-IF folgt, wird ausgeführt.

Wie in höheren Programmiersprachen wird in IF-Blöcken zwischen THEN-Zweig und ELSE-Zweig unterschieden. Im obigen Beispiel bildet $\texttt{kommandofolge}_1$ den THEN-Zweig und der Sprung zu END-IF den ELSE-Zweig.

Wenn alternativ zu kommandofolge$_1$ eine andere Kommandofolge ausgeführt werden soll, muss explizit ein ELSE-Zweig angegeben werden. Der ELSE-Zweig enthält die alternative Kommandofolge. Er wird mit dem Kommando ELSE eingeleitet und mit dem Kommando END-IF abgeschlossen.

```
/IF bedingung
  kommandofolge1

/ELSE
  kommandofolge2

/END-IF
```

Wenn die Bedinung im IF-Kommando erfüllt ist, wird kommandofolge$_1$ ausgeführt, und der Block wird beendet. Wenn die Bedingung nicht erfüllt ist, wird kommandofolge$_2$ ausgeführt. In einem IF-Block können auch mehrere Bedinungen nacheinander geprüft werden.

```
/IF bedingung1
  kommandofolge1

/ELSE-IF bedingung2
  kommandofolge2

/ELSE-IF bedingung3
  kommandofolge3
...
/ELSE
  kommandofolgei

/END-IF
```

Ist die Bedingung im IF-Kommando erfüllt, wird kommandofolge$_1$ ausgeführt. Ist sie nicht erfüllt, wird die Bedingung im ersten ELSE-IF-Kommando geprüft. Ist diese Bedingung erfüllt, wird kommandofolge$_2$ ausgeführt, andernfalls wird die Bedingung im nächsten ELSE-IF-Kommando geprüft usw. Ist keine der Bedingungen erfüllt, wird kommandofolge$_i$ ausgeführt, die dem ELSE-Kommando folgt.

Die Kommandofolgen zwischen den Kommandos IF, ELSE-IF, ELSE und END-IF bilden jeweils eigene Blöcke.

Beispiel

Abhängig vom Wert der Variablen A soll eine Datei angelegt werden, deren Dateiname als Teilnamen den Wert der Variablen enthält:

– Wenn der Wert der Variablen A kleiner ist als 11, soll eine Datei DATEI.KLEIN.&A angelegt werden.
– Wenn der Wert der Variablen A zwischen 11 und 100 liegt, soll eine Datei DATEI.MITTEL.&A angelegt werden.
– Ist A größer als 100, soll eine Datei DATEI.GROSS.&A angelegt werden.

```
/DECLARE-PARAMETER A(INITIAL-VALUE=*PROMPT,TYPE=*INTEGER)
/IF (A < 11)
/    CREATE-FILE DATEI.KLEIN.&A,SUPPORT=*PUB-DISK(SPACE=*RELA(PRIM-ALLOC=30))
/ ELSE-IF (A < 101)
/    CREATE-FILE DATEI.MITTEL.&A,SUPPORT=*PUB-DISK(SPACE=*RELA(PRIMALLOC=60))
/ ELSE
/    CREATE-FILE DATEI.GROSS.&A,SUPPORT=*PUB-DISK(SPACE=*RELA(PRIM-ALLOC=90))
/END-IF
```

Die Dateien werden angelegt mit SUPPORT = *PUB-DISK(SPACE = *RELA(PRIM-ALLOC= ...)) im Kommando CREATE-FILE:

– SUPPORT = *PUBLIC-DISK bewirkt, dass die Datei als Plattendatei auf einer gemeinschaftlichen Platte angelegt wird.
– SPACE = *RELATIVE bewirkt relative Speicherplatzzuweisung.
– PRIMARY-ALLOCATION bewirkt, dass der Datei n PAM-Seiten Speicherplatz zugewiesen werden.

Wie Dateien erzeugt werden, ist ausführlich im Handbuch „Einführung in das DVS“ [1] beschrieben. Dort sind auch die Begriffe „Plattendatei“, „gemeinschaftliche Platte“, „(relative) Speicherplatzzuweisung“ und „PAM-Seite“ erklärt.

4.2.3 Schleifen definieren

In einer Schleife kann eine Kommandofolge in Abhängigkeit einer Bedingung mehrmals durchlaufen werden. Schleifen werden über Kommandoblöcke gebildet. Dabei sind drei Schleifenblöcke zu unterscheiden, die nach dem einleitenden Kommando benannt werden: FOR-Block, WHILE-Block und REPEAT-Block.

FOR-Block

Im FOR-Block wird eine Kommandofolge so oft ausgeführt, wie einer Laufvariablen Werte zugewiesen werden. Der FOR-Block wird auch als FOR-Schleife bezeichnet werden.

Der FOR-Block beginnt mit dem Kommando FOR und endet mit dem Kommando END-FOR. Im FOR-Kommando wird einer Laufvariablen zugewiesen:
– ein Zähler
– der Wert einer Listenvariablen
– ein Ausdruck
– die Werte der Elemente einer Liste von Ausdrücken, Listenvariablen und Zähler

```
/FOR variable = counter / listenvariable / ausdruck [,CONDITION = bedingung]
[kommandofolge]
/END-FOR
```

Der Benutzer definiert eine Laufvariable; die Anzahl der Schleifendurchläufe und der Wert der Laufvariablen ist abhängig von der Zuweisung rechts vom Gleichheitszeichen (ausdruck, listenvariable, counter):

– counter
  Die Anzahl der Schleifendurchläufe wird durch den Anfangswert, den Endwert und die Schrittweite festgelegt. Die Laufvariable enthält jeweils den aktuellen Schleifenwert.
– listenvariable
  Die Anzahl der Schleifendurchläufe ist gleich der Anzahl der Variablenelemente. Bei jedem Schleifendurchlauf wird in aufsteigender Reihenfolge der Elemente der Laufvariablen der Wert des Elementes zugewiesen.
– ausdruck
  Die Anzahl der Schleifendurchläufe ist gleich der Anzahl der Elemente der Werteliste. Bei jedem Schleifendurchlauf wird der Laufvariablen - von links beginnend - der jeweilige Ausdruck (String-, arithmetischer -, boolscher Ausdruck, ...) zugewiesen.
– bedingung
  Wahlweise kann eine Bedingung in Form eines logischen Ausdrucks angegeben werden. Diese wird vor jedem Schleifendurchlauf überprüft; ist sie nicht mehr erfüllt, wird die Schleife beendet.

Werden counter, listenvariable und ausdruck in einer Werteliste gemischt, so wird diese Liste - von links beginnend - abgearbeitet, so wie angegeben.

Eine Ausdrucksersetzung (&....) in einem der FOR-Operanden erfolgt nur beim Eintritt in die FOR-Schleife, nicht bei jedem Schleifendurchlauf.

Mit der Laufvariablen können im FOR-Block Kommandos modifiziert oder ersetzt werden, zum Beispiel durch Variablenersetzung.

Der FOR-Block kann mit einer Marke versehen werden.

Beispiel 1

```
/DECLARE-VARIABLE A, MULTIPLE-ELEMENTS=*LIST  ————————————————————————  (1)
/DECLARE-VARIABLE L  ————————————————————————————————————————————————  (2)
/SET-VARIABLE A=1436,WRITE-MODE=*EXTEND  ————————————————————————————  (3)
/A=1455,WRITE-MODE=*EXTEND  ————————————————————————————————————————  (4)
/A=1577,WRITE-MODE=*EXTEND  ————————————————————————————————————————  (5)
/FOR L=*LIST(A)  ———————————————————————————————————————————————————  (6)
/ CANCEL-JOB JOB-ID=TSN(&L)  ———————————————————————————————————————  (7)
/END-FOR
```

Erläuterungen:

(1) Deklaration der Listenvariablen A (DECLARE-VARIABLE siehe Seite 614).

(2) Deklaration der Laufvariablen L.

(3) Der Variablen A wird der Wert 1436 zugewiesen (SET-VARIABLE siehe Seite 753). WRITE-MODE= *EXTEND bewirkt, dass die Liste um ein Element verlängert wird. Diesem neuen letzten Element wird der Wert zugewiesen. Hier ist das letzte Element gleichzeitig das erste Element der Liste.

(4) Der Variablen A wird als zweiter Wert 1455 zugewiesen (der Kommandoname darf bei SET-VARIABLE entfallen).

(5) Der Variablen A wird als dritter Wert 1577 zugewiesen.

(6) Beginn der FOR-Schleife mit der Laufvariablen L und dem Inhalt der Variablen A als Werteliste.

(7) Für jeden Wert in Variable A wird ein CANCEL-JOB-Kommando auf die TSN abgesetzt, die in der Variablen L steht.

Durch die FOR-Schleife werden folgende Kommandos abgesetzt:

```
/CANCEL-JOB JOB-ID=*TSN(1436
/CANCEL-JOB JOB-ID=*TSN(1455)
/CANCEL-JOB JOB-ID=*TSN(1577)
```

Beispiel 2

```
/FOR I = *COUNTER(1,8)
/   ENTER-PROCEDURE ENTERDATEI.&I
/END-FOR
```

Die FOR-Schleife erzeugt acht ENTER-PROCEDURE-Kommandos für die Dateien ENTERDATEI.1 bis ENTERDATEI.8.

WHILE-Block

Im WHILE-Block wird eine Kommandofolge so lange ausgeführt, wie eine vom Anwender definierte Bedingung erfüllt ist. Der WHILE-Block beginnt mit dem Kommando WHILE und endet mit dem Kommando END-WHILE. Der WHILE-Block kann auch als WHILE-Schleife bezeichnet werden.

Das WHILE-Kommando enthält die Bedingung zum Durchlaufen der Schleife; diese Bedingung wird vor jedem Schleifendurchlauf überprüft. Wenn die Bedingung erfüllt ist, wird die Kommandofolge innerhalb des Blocks durchlaufen; ist die Bedingung nicht erfüllt, wird die Schleife beendet und der Prozedurlauf mit dem auf END-WHILE folgenden Kommando fortgesetzt.

Die Bedingung im WHILE-Kommando wird als logischer Ausdruck angegeben (siehe Abschnitt „Typen von Ausdrücken“ auf Seite 271).

```
/[marke:]WHILE bedingung
[kommandofolge]
/END-WHILE [marke]
```

Beispiel

```
/ "Reduzieren der Listenvariablen LIST-A auf die letzten 250 Elemente"
/WHILE (SIZE ('LIST-A') > 250)
/  FREE-VARIABLE LISTE-A#
/END-WHILE
```

Die WHILE-Schleife reduziert die Listenvariable LIST-A auf die letzten 250 Elemente.

REPEAT-Block

Im REPEAT-Block wird eine Kommandofolge so lange ausgeführt, bis eine vom Anwender definierte Bedingung erfüllt ist. Der Block beginnt mit dem Kommando REPEAT und endet mit dem Kommando UNTIL. Der REPEAT-Block kann auch als REPEAT-Schleife bezeichnet werden.

Im Gegensatz zu FOR- und WHILE-Schleifen ist die Bedingung für den Schleifenabbruch nicht im einleitenden Kommando enthalten, sondern im Abschlusskommando des Blocks, dem Kommando UNTIL. Die Schleife wird daher mindestens einmal durchlaufen.

Am Ende eines jeden Schleifendurchlaufs wird die Bedingung im UNTIL-Kommando geprüft. Solange die Bedingung nicht erfüllt ist (Abbruchbedingung), wird die Kommandofolge innerhalb des Blocks erneut durchlaufen; sobald sie erfüllt ist, werden die Kommandos, die auf UNTIL folgen, abgearbeitet.

Die Bedingung im UNTIL-Kommando wird als logischer Ausdruck angegeben (siehe Kapitel „Ausdrücke“ auf Seite 255).

```
/REPEAT
[kommandofolge]
/UNTIL bedingung
```

Beispiel

```
/DECLARE-VARIABLE A
/DECLARE-VARIABLE SCHALTER-1(TYPE=*BOOLEAN)
/SET-VARIABLE A = 5
/SET-VARIABLE SCHALTER-1 = ON
/REPEAT
/  A = A + 10
/  IF (A > 50)
/    SET-VARIABLE SCHALTER-1 = OFF
/  END-IF
/  UNTIL (SCHALTER-1 = OFF)
/SHOW-VARIABLE A
A = 55
```

4.3 Sprünge definieren

Sprünge werden über Sprungkommandos durchgeführt, die das sequenzielle Abarbeiten von Kommandos abbrechen, zu einer definierten Prozedurstelle „springen“ und dort den Prozedurlauf fortsetzen.

In SDF-P stehen folgende Sprungkommandos zur Verfügung:

– EXIT-BLOCK
– CYCLE
– GOTO
– INCLUDE-BLOCK

Mit den Sprungkommandos können nur solche Sprungziele adressiert werden, die im aktuellen Kommandoblock liegen oder in Kommandoblöcken, die den aktuellen Kommandoblock umgeben.

Es ist nicht möglich, von außen in einen Kommandoblock hineinzuspringen, der eine oder mehrere Schachtelungsebenen tiefer liegt.

Als Sprungziele dienen Marken, die in Kommandoaufrufen gesetzt werden. Diese Marken können in den Sprungkommandos EXIT-BLOCK, CYCLE, GOTO und INCLUDE-BLOCK verwendet werden.

Die Sprungkommandos lassen sich in drei Gruppen einteilen:
– Kommandos, deren Sprungziel immer der Blockanfang ist (INCLUDE-BLOCK)
– Kommandos, deren Sprungziel immer das Blockende ist (EXIT-BLOCK, CYCLE)
– Kommandos, die beliebige Sprungziele adressieren (GOTO und kompatibel zu Nicht-S-Prozeduren: SKIP-COMMANDS).

4.3.1 Blockanfang als Sprungziel

Das Kommando INCLUDE-BLOCK springt in einer Prozedur zu der Kommandozeile mit der angegebenen Marke (marke:), wobei die Marke den Anfang eines BEGIN-Blockes kennzeichnen muss (d.h. auf die Marke muss ein BEGIN-BLOCK-Kommando folgen). Der Prozedurlauf wird mit der Abarbeitung des BEGIN-Blockes fortgesetzt. Bei Erreichen des Blockendes wird die Bearbeitung mit dem Kommando fortgesetzt, das auf das INCLUDE-BLOCK-Kommando folgt.

Beispiel

```
/ASSIGN-SYSLST TO=PROT.EINGABE,SYSLST-NUMBER=1
/...
/IF (EING='*START')
/ INCLUDE-BLOCK INFO-1
/END-IF     &* Hier geht es nach Ausführung der Unterprozedur INFO-1 weiter
/...
/IF (EING='*END')
/ INCLUDE-BLOCK INFO-1
/END-IF     &* Hier geht es nach Ausführung der Unterprozedur INFO-1 weiter
/...
/...
/...
/INFO-1: BEGIN-BLOCK    &* Beginn der Unterprozedur INFO-1
/      WRITE-TEXT '&(TIME()): Es wurde &(EING) eingegeben',OUTPUT=*SYSLST(1)
/END-BLOCK &*Ende + Rücksprung zur Kommandozeile, die auf INCLUDE-BLOCK folgt
```

4.3.2 Blockende als Sprungziel

4.3.2.1 Sprung an das Ende eines beliebigen Kommandoblocks

Das Kommando EXIT-BLOCK bricht die Verarbeitung der Kommandofolge des Blocks ab und springt zum Abschlusskommando des Blocks. Anschließend wird der Prozedurlauf mit dem Kommando fortgesetzt, das auf das Blockabschlusskommando folgt.

Im Aufruf des EXIT-BLOCK-Kommandos kann der Block, der abgebrochen werden soll, implizit adressiert werden über den voreingestellten Wert *LAST oder explizit über den Namen der Marke, die einem Blockeinleitungskommando vorangestellt ist.

Beispiel

```
/LOOP: WHILE (BED < 9)
/...
/IF (EING='*END')
/ EXIT-BLOCK LOOP
/END-IF
/...
/END-WHILE
/"Hier geht es nach Ausführung von EXIT-BLOCK weiter"
```

Beim Sprung mit EXIT-BLOCK kann die Marke des aktuellen Blocks angegeben werden (wie im Beispiel oben) oder die Marke eines umgebenden Blocks.

Beispiel

```
/LOOPWH: WHILE (BED<9)
/...
/LOOPFOR: FOR I=*LIST(LISTE)
/...
/             IF (BED=TRUE)
/...
/                EXIT-BLOCK LOOPWH
/...
/              END-IF
/...
/          END-FOR
/...
/END-WHILE
/"Hier geht's weiter nach EXIT-BLOCK"
```

Mit dem Operanden *ALL können alle umgebenden Blöcke abgebrochen werden. Der Prozedurlauf wird dann mit dem Kommando fortgesetzt, das auf der obersten Prozedurebene dem END-Kommando folgt, das den äußersten Block abschließt. Der Einsatz von EXIT-BLOCK *ALL ist zum Beispiel sinnvoll in Fehlerbehandlungsblöcken, wenn die Prozedur nach einem Fehler nicht fortgesetzt, aber korrekt beendet werden soll.

Hinweis

Wird /EXIT-BLOCK ohne Marke in einem IF-Block geschrieben, wird die Prozedur nach dem zugehörigen END-IF fortgesetzt.
Die Adressierung über die Voreinstellung BLOCK=*LAST ist somit nur sinnvoll, wenn die Sprungbedingung im Kommando EXIT-BLOCK selbst angegeben wird.

4.3.2.2 Sprung ans Schleifenende

Das Kommando CYCLE kann nur in Schleifen eingesetzt werden. Es bricht die Verarbeitung der Kommandofolge einer Schleife ab und springt zum entsprechenden Schleifenende-Kommando. Dort wird der Prozedurlauf fortgesetzt.

Bei REPEAT-Schleifen enthält das Schleifenende-Kommando UNTIL die Schleifenbedingung; diese Bedingung wird geprüft und wenn nötig der nächste Schleifendurchlauf gestartet. Bei FOR- und WHILE-Schleifen enthalten die Schleifenende-Kommandos (END-FOR, END-WHILE) den Sprung zurück an den Schleifenanfang. Dort wird ebenfalls die Schleifenbedingung geprüft und wenn nötig der nächste Schleifendurchlauf gestartet.
Im Aufruf des CYCLE-Kommandos kann die Schleife, auf die sich der Kommandoaufruf beziehen soll, implizit adressiert werden über den voreingestellten Wert *LAST oder explizit über den Namen der Marke, die dem Schleifen-Einleitungskommando vorangestellt ist.

Beispiel

```
/LOOP: WHILE (BED < 9)
/...
/IF (EING='*IGN')
/ CYCLE LOOP
/END-IF
/...
/END-WHILE   "Dieses Kommando wird nach CYCLE ausgefuehrt"
```

Hinweis

Wenn - wie in diesem Beispiel - das CYCLE-Kommando in einen IF-Block (oder auch in einen einfachen Block) eingesetzt wird, muss bei CYCLE mit einer Marke auf den betroffenen Schleifen-Block hingewiesen werden.

4.3.3 Beliebige Sprungziele

Sprung an eine S-Marke

Das Kommando GOTO springt in der Prozedur zu der Kommandozeile mit der angegebenen Marke (marke:). Der Prozedurlauf wird mit diesem Kommandoaufruf fortgesetzt.

Das GOTO-Kommando kann nur zum Verlassen eines Kommandoblocks bzw. innerhalb eines Kommandoblocks verwendet werden. Es ist nicht möglich, mit GOTO in einen Kommandoblock von außen hineinzuspringen. Das heißt, GOTO kann Sprungziele in übergeordneten Schachtelungsebenen adressieren, jedoch nicht in untergeordneten.

Beispiel

Die folgenden Zeilen zeigen Beispiele für erlaubte und nicht erlaubte Anwendung des Sprungkommandos GOTO.

```
/LOOP1: WHILE (A < B)
/ADD1: X = X + A
/LOOP2: WHILE (X < Y)
/ADD2: A = A + 1
/  GOTO ADD1                "Erlaubt, weil umgebende Schleife"
/  END-WHILE LOOP2
/  GOTO ADD2                "Verboten, weil innere Schleife!"
/END-WHILE LOOP1
```

Sprung an eine Nicht-S-Marke

Die Kommandos SKIP-COMMANDS und WAIT-EVENT sind SDF-Kommandos des BS2000-Grundausbaus, die aus Kompatibilität mit Nicht-S-Prozeduren unterstützt werden.

Sie erkennen als Sprungziele nur Marken des Nicht-S-Formats (.marke). Nicht-S-Marken dürfen nicht innerhalb von Kommandoblöcken, sondern nur auf der obersten Prozedurebene eingesetzt werden, d.h. mit der Schachtelungstiefe 0.

Mit SKIP-COMMANDS können bedingte und unbedingte Sprünge durchgeführt werden. Sprungziel ist das Kommando mit der angegebenen Marke, dort wird der Prozedurlauf fortgesetzt.

Mit WAIT-EVENT kann der zeitliche Ablauf eines Auftrags von Benutzerschalterstellungen (nur in Stapelaufträgen) oder vom Zustand einer Jobvariablen (JV) abhängig gemacht werden.

Beide Kommandos sind im Handbuch „Kommandos, Bd. 1-5“ [3] beschrieben.

5 Prozeduren aufrufen und steuern

S-Prozeduren können sowohl von der Systemebene als auch aus Prozeduren heraus aufgerufen und gestartet werden. Sie können im Vordergrund ablaufen, d.h. synchron zum aufrufenden Auftrag, oder im Hintergrund, d.h. in einem Stapelauftrag, asynchron zum aufrufenden Auftrag. Dementsprechend muss das Kommando für den Prozeduraufruf gewählt werden. Für den ersten Fall stehen die Kommandos CALL-PROCEDURE bzw. INCLUDE-PROCEDURE zur Verfügung, für den zweiten Fall das Kommando ENTER-PROCEDURE.

5.1 S-Prozeduren im Vordergrund aufrufen

Eine „S-Prozedur im Vordergrund aufrufen“ bedeutet, dass die Prozedur unter Steuerung des Auftrags abläuft, in dem das Prozeduraufruf-Kommando gegeben wurde; es wird kein neuer Auftrag erzeugt. Diese Art von Prozedur wird auch als synchron aufgerufene Prozedur oder als „Dialogprozedur“ bezeichnet, da Interaktion mit dem Benutzer möglich ist. Eine Prozedur im Vordergrund aufrufen kann sowohl von der Systemebene als auch von einer Prozedur heraus erfolgen. S-Prozeduren werden im Vordergrund mit dem Kommando CALL-PROCEDURE oder mit dem Kommando INCLUDE-PROCEDURE aufgerufen.

Prozeduren, die mit CALL-PROCEDURE aufgerufen werden, werden auch als Call-Prozeduren bezeichnet; entsprechend werden Prozeduren, die mit INCLUDE-PROCEDURE aufgerufen werden, als Include-Prozeduren bezeichnet.

Hinweis

Mit CALL-PROCEDURE können sowohl S-Prozeduren in Nicht-S-Prozeduren als auch Nicht-S-Prozeduren in S-Prozeduren aufgerufen werden.

5.1.1 Aufrufkommando wählen

Mit welchem Kommando eine Prozedur aufgerufen werden darf, wird im Prozedurkopf eingestellt: im Kommando SET-PROCEDURE-OPTIONS, Operand CALLER. Bei CALLER = *CALL darf sie nur mit CALL-PROCEDURE aufgerufen werden, bei CALLER = *INCLUDE nur mit INCLUDE-PROCEDURE. Nur wenn CALLER = *ANY gilt, kann die Prozedur mit CALL-PROCEDURE oder mit INCLUDE-PROCEDURE aufgerufen werden.

CALL-PROCEDURE und INCLUDE-PROCEDURE haben exakt die gleichen Operanden. Sie unterscheiden sich aber in ihrer Wirkung bezüglich der Sichtbarkeit von Variablen, wenn die Prozedur aus einer anderen Prozedur heraus aufgerufen wird. In mit INCLUDE-PROCEDURE aufgerufenen Prozeduren sind standardmäßig alle Variablen sichtbar, die auch in der aufrufenden Prozedur sichtbar sind. In Prozeduren, die mit dem Kommando-CALL-PROCEDURE aufgerufenen werden, sind standardmäßig nur die prozedurlokalen Variablen der aktuellen Prozedur sichtbar. (Näheres siehe Abschnitt „Geltungsbereich von Variablen“ auf Seite 162.)

Die Wahl des Aufruf-Kommandos hängt also davon ab, welche Variablenumgebung die aufgerufene Prozedur erhalten soll.

Im Aufruf einer Prozedur muss zunächst der Prozedurbehälter benannt werden. Der Aufrufer kann dann einstellen, wie die Prozedur protokolliert werden soll, ob ein zum Ablaufzeitpunkt geladenes Programm entladen werden darf und ob die Prozedur als Ganzes sofort ausgeführt oder zum Testen unterbrochen werden soll. Außerdem kann er im Prozeduraufruf Prozedurparameter übergeben.

Alle diese Eigenschaften werden über die Operanden der Prozeduraufruf-Kommandos gesteuert und in den folgenden Abschnitten beschrieben. Eine Besonderheit gilt beim Protokollieren: Protokollieren kann zwar auch beim Prozeduraufruf eingestellt werden, ob und was protokolliert wird, hängt aber davon ab, welche Einstellungen innerhalb der Prozedur während des Prozedurablaufs gelten.

5.1.2 Prozedurbehälter angeben

Der Prozedurbehälter wird im Prozeduraufruf im Operanden FROM-FILE angegeben. Prozedurbehälter kann eine BS2000-Datei, ein Bibliothekselement oder eine Listenvariable sein. FROM-FILE = *VARIABLE(VARIABLE-NAME = ...) bezeichnet eine Listenvariable: jedes Element dieser Liste enthält dann eine Prozedurzeile.

5.1.3 Prozedurparameter übergeben

Beim Prozeduraufruf mit CALL-PROCEDURE oder INCLUDE-PROCEDURE können im Operanden PROCEDURE-PARAMETERS Prozedurparameter an die aufgerufene Prozedur übergeben werden. Der Vorgang wird als „Parameterübergabe“ bezeichnet.

Diese Prozedurparameter müssen im Prozedurkopf der aufgerufenen Prozedur mit dem Kommando DECLARE-PARAMETER deklariert sein (siehe Kapitel „S-Prozeduren erstellen“ auf Seite 81).

Die mit DECLARE-PARAMETER deklarierten Prozedurparameter werden von SDF-P wie prozedurlokale Variablen behandelt, also wie Variablen mit dem Geltungsbereich SCOPE = *CURRENT (siehe Kommando DECLARE-VARIABLE auf Seite 614).

Prozedurparameter können als Schlüsselwortparameter oder als Stellungsparameter angegeben werden.

Je nach Art der Parameterübergabe, die für die Prozedur im Kommando DECLARE-PARAMETER festgelegt ist, können im Aufruf Parameterwerte oder Namen von Variablen mit den Parameterwerten angegeben werden.

5.1.3.1 Prozedurparameter als Stellungsparameter übergeben

Stellungsparameter werden ohne den vorangestellten Parameternamen, durch Kommas voneinander getrennt, angegeben.

Wenn Prozedurparameter als Stellungsparameter übergeben werden, muss die Reihenfolge beachtet werden, in der die Parameter in der aufgerufenen Prozedur mit DECLARE-PARAMETER deklariert sind. Sie werden in dieser Reihenfolge zugeordnet.

Das heißt, wenn in jedem DECLARE-PARAMETER-Kommando nur ein Prozedurparameter deklariert ist, wird der erste angegebene Prozedurparameter auf den Parameter(-namen) abgebildet, der im ersten DECLARE-PARAMETER-Kommando deklariert ist, der Zweite auf den Parameternamen im zweiten DECLARE-PARAMETER-Kommando usw.

Werden in einem DECLARE-PARAMETER-Kommando mehrere Prozedurparameter deklariert, werden die übergebenen Prozedurparameter in der Reihenfolge auf sie abgebildet, in der die Parameternamen deklariert sind.

Bei der Prozedurparameter-Deklaration können Prozedurparameter bereits mit einem gültigen Wert vorbelegt, das heißt initialisiert werden. Diese Prozedurparameter können bei der Parameterübergabe im Prozeduraufruf „übergangen“ werden, wenn ihr Anfangswert gültig bleibt. Bei der Übergabe von Stellungsparametern ist zu berücksichtigen, wo die zu „übergehenden“ Prozedurparameter in der Deklarationsfolge stehen.

Im Prozeduraufruf brauchen Prozedurparameter, die bereits initialisiert sind, nicht berücksichtigt zu werden, wenn sie die letzten Prozedurparameter der Deklarationsfolge sind.

Folgen auf diese bereits initialisierten Prozedurparameter aber andere Prozedurparameter, für die ein aktueller Wert übergeben wird, müssen sie als leere Parameter übergeben werden: für den zu übergehenden Parameter muss ein Komma gesetzt werden.

Beispiel

Prozedur PROC1

```
/...
/CALL-PROCEDURE PROC2, PROCEDURE-PARAMETERS = (MUELLER, EDUARD, GOETHE-
/STRASSE, , , 1234567)
```

Prozedur PROC2

```
/SET-PROCEDURE-OPTIONS
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER (NAME (*NONE, *STRING), VORNAME (*NONE, *STRING))
/   DECLARE-PARAMETER STRASSE(*NONE, *STRING)
/   DECLARE-PARAMETER ORT('HAMBURG', *STRING)
/   DECLARE-PARAMETER (VORWAHL('040', *STRING), TEL(*NONE, *STRING))
/END-PARAMETER-DECLARATION
```

Beim Prozeduraufruf werden an PROC2 sechs Prozedurparameter übergeben: Name (MUELLER), Vorname (EDUARD) und Straße (GOETHESTRASSE). Für die beiden Parameter ORT und VORWAHL werden keine Werte übergeben, für sie werden nur Kommas eingesetzt. Als letzter Wert wird die Telefonnummer übergeben. Die Prozedurparameter ORT und VORWAHL sind vorbelegt mit den Werten HAMBURG und 040 (zum Beispiel, weil es sich um Adresslisten eines Hamburger Unternehmens handelt). Da TRANSFER-TYPE = *BY-VALUE voreingestellt ist, werden die angegebenen Prozedurparameter als Werte-Angaben behandelt.

5.1.3.2 Prozedurparameter als Schlüsselwortparameter übergeben

Parameter können auch als Schlüsselwortparameter übergeben werden. Schlüsselwort ist dabei der Name, mit dem der Prozedurparameter im DECLARE-PARAMETER-Kommando deklariert ist. Entsprechend den Regeln für Abkürzbarkeit bei Schlüsselwortoperanden in Kommandos, können auch hier die Schlüsselwörter bis zur Eindeutigkeit abgekürzt werden.

Bei der Übergabe als Schlüsselwortparameter ist die Reihenfolge der Prozedurparameter beliebig, da die Parameternamen eindeutig sein müssen.

Parameter, die in der aufgerufenen Prozedur initialisiert werden, brauchen nur dann berücksichtigt zu werden, wenn ihr Anfangswert überschrieben werden soll.

Beispiel

Werden die Prozedurparameter als Schlüsselwortparameter übergeben, sieht der Prozeduraufruf aus dem vorhergehenden Beispiel folgendermaßen aus:

```
/CALL-PROCEDURE PROC2,-
/PROCEDURE-PARAMETERS = (NAME = MUELLER, VORNAME = EDUARD,-
/STRASSE = GOETHESTRASSE, TEL = 1234567)
```

5.1.3.3 Stellungs- und Schlüsselwortparameter mischen

Im Prozeduraufruf können gleichzeitig Stellungs- und Schlüsselwortparameter übergeben werden. In diesem Fall müssen zunächst die Stellungsparameter angegeben werden, erst dann folgen die Schlüsselwortparameter.

Beispiel

Die Parameterübergabe für die Hamburger Adressliste kann dann so aussehen:

```
/CALL-PROCEDURE PROC2, PROCEDURE-PARAMETERS = (MUELLER, EDUARD, -
/GOETHESTRASSE, TEL = 1234567)
```

Name, Vorname und Straße werden als Stellungsparameter übergeben, die Telefonnummer als Schlüsselwortparameter. Die Prozedurparameter ORT und VORWAHL brauchen nicht berücksichtigt zu werden, da sie bei der Deklaration initialisiert wurden.

5.1.3.4 Art der Parameterübergabe

Bei der Deklaration der Prozedurparameter mit DECLARE-PARAMETER wird auch festgelegt, wie die für den Prozedurparameter übergebene Zeichenfolge zu behandeln ist, und zwar im Operanden TRANSFER-TYPE.

Gilt TRANSFER-TYPE = *BY-VALUE, wird die übergebene Zeichenfolge dem entsprechenden Prozedurparameter als Wert zugewiesen.

Beispiel

```
/CALL-PROCEDURE PROC2, PROCEDURE-PARAMETERS = (MUELLER, EDUARD, -
/GOETHESTRASSE, TEL = 1234567)
```

Hier wird dem Prozedurparameter NAME der Wert MUELLER zugewiesen, dem Prozedurparameter VORNAME der Wert EDUARD usw.

Gilt TRANSFER-TYPE = *BY-REFERENCE, wird die übergebene Zeichenfolge als Variablenbehälter des formalen Prozedurparameters ausgewertet. Dann kann die gerufene Prozedur über diesen Prozedurparameter Ergebnisse an den Aufrufer zurückliefern.

Beispiel

Aufrufende Prozedur:

```
/DECLARE-VARIABLE NAME('MUELLER',*STRING)
/DECLARE-VARIABLE VORNAME('EDUARD',*STRING)
/DECLARE-VARIABLE STR('GOETHESTRASSE',*STRING)
/DECLARE-VARIABLE TELEFON('1234567',*STRING)
/...
/CALL-PROCEDURE PROC2, PROCEDURE-PARAMETERS = (NAME, VORNAME, STR,,, TEL)
```

Aufgerufene Prozedur PROC2:

```
/SET-PROCEDURE-OPTIONS
/BEGIN-PARAMETER-DECLARATION
/DECLARE-PARAMETER NAME (*NONE, *STRING,-
/  TRANSFER-TYPE = *BY-REFERENCE)
/DECLARE-PARAMETER VORNAME (*NONE, *STRING,-
/  TRANSFER-TYPE = *BY-REFERENCE)
/DECLARE-PARAMETER STR(*NONE, *STRING,-
/  TRANSFER-TYPE = *BY-REFERENCE)
/DECLARE-PARAMETER ORT('HAMBURG', *STRING,-
/  TRANSFER-TYPE = *BY-VALUE)
/DECLARE-PARAMETER VORWAHL('040', *STRING,-
/  TRANSFER-TYPE = *BY-VALUE)
/DECLARE-PARAMETER TEL(*NONE, *STRING,-
/  TRANSFER-TYPE = *BY-REFERENCE)
/END-PARAMETER-DECLARATION
```

Als Prozedurparameter werden Variablennamen übergeben. Da in PROC2 für die Prozedurparameter TRANSFER-TYPE = *BY-REFERENCE gilt, wird nicht die übergebene Zeichenfolge als Wert zugewiesen, sondern Variablen werden miteinander verknüpft.

Beispiel

Die Prozedur P enthält folgende Zeilen:

```
/SET-PROCEDURE-OPTIONS
/BEGIN-PARAMETER-DECLARATION
/DECLARE-PARAMETER SUMME (TRANSFER-TYPE=*BY-REFERENCE)
/DECLARE-PARAMETER (P1(TYPE=*INTEGER),P2(TYPE=*INTEGER))
/END-PARAMETER-DECLARATION
/SUMME = P1+P2
```

Im Dialog werden folgende Kommandos eingegeben:

```
/DECLARE-VARIABLE S
/CALL-PROCEDURE P, (S,3,5)
/SHOW-VARIABLE S
```

Dies ergibt die Ausgabe:

```
S = 8
```

Die Übergabe von Variablennamen als Prozedurparameter erfolgt über einen so genannten „Container-“ oder „Behältermechanismus“: Der Prozedurparameter dient als Behälter für die Variable, damit die aufgerufene Prozedur auf diese Variable zugreifen und ihren Inhalt auswerten kann.

Variablennamen können nur dann als Prozedurparameter eingesetzt werden, wenn Variable und Prozedurparameter mit dem gleichen Datentyp deklariert sind; die Variable mit DECLARE-VARIABLE in der aufrufenden Prozedur, der Prozedurparameter mit DECLARE-PARAMETER in der aufgerufenen Prozedur. Das heißt, beide müssen zum Beispiel mit TYPE = *INTEGER deklariert sein. Es darf nicht einer mit TYPE=*INTEGER und der andere mit TYPE = *ANY deklariert sein.

5.1.4 Protokollierung veranlassen

Im Kommando SET-PROCEDURE-OPTIONS wird festgelegt, ob bzw. was während des Prozedurablaufs protokolliert werden darf. Im Rahmen dieser Festlegung kann der Aufrufer in den Kommandos CALL-PROCEDURE / INCLUDE-PROCEDURE die Protokollierung einstellen.

Auf das Protokollieren wirken sich die Einstellungen verschiedener Kommandos aus, die sich bei Vordergrund-Prozeduren hinsichtlich Prozedur- und Job-Ebene folgendermaßen gruppieren lassen:

- SDF-P-Kommandos, mit denen prozedurintern eingestellt wird, ob Protokollieren zulässig ist: SET-PROCEDURE-OPTIONS und MODIFY-PROCEDURE-OPTIONS, Operand LOGGING-ALLOWED.
- SDF-P-Kommandos für den Prozeduraufruf, in denen der Aufrufer ein Protokoll anfordern kann: CALL-PROCEDURE und INCLUDE-PROCEDURE, Operand LOGGING.
- SDF-P-Kommandos für die Testphase der Prozedur: TRACE-PROCEDURE und MODIFY-PROCEDURE-TEST-OPTIONS.
- Kommandos, die das Protokollieren auf Job-Ebene bzw. für SDF beeinflussen: MODIFY-JOB-OPTIONS und MODIFY-SDF-OPTIONS.

Das Kommando MODIFY-JOB-OPTIONS beeinflusst das Protokollieren des Auftragsablaufs, zum Beispiel ob ein zusätzliches SYSLST-Protokoll erstellt werden soll oder ob Hardcopies erzeugt werden. MODIFY-SDF-OPTIONS beeinflusst die Form des Protokolls. Diese beiden Kommandos sind keine SDF-P-Kommandos. Sie sind im Handbuch „Kommandos, Bd. 1-5“ [3] ausführlich beschrieben.
Im Folgenden werden nur die Auswirkungen der SDF-P-Kommandos auf das Protokollieren berücksichtigt. Ob aktuell protokolliert wird, kann mit der vordefinierten Funktion LOGGING-MODE( ) abgefragt werden (siehe Seite 449).

5.1.4.1 Zulässigkeit des Protokollierens

Ob Protokollieren zulässig ist, wird zum einen gesteuert über Schutzmechanismen des Datenverwaltungssystems: Die Prozedur kann mit einem Lesekennwort belegt werden. (Der Leseschutz kann nur aufgehoben werden, wenn das entsprechende Kennwort angegeben wird.) Mit ACL oder BASIC-ACL können Ausführungs- und Leseberechtigung getrennt vergeben werden.

Zum anderen wird dies prozedurintern über die Kommandos SET-PROCEDURE-OPTIONS und MODIFY-PROCEDURE-OPTIONS gesteuert, und zwar jeweils über den Operanden LOGGING-ALLOWED. Dabei wird unterschieden zwischen dem Protokollieren von Kommandos und dem Protokollieren von Daten.

Soll Protokollieren für Kommandos und/oder Daten vom Beginn des Prozedurlaufs an unterbunden werden, muss die Prozedur mit dem Kommando SET-PROCEDURE-OPTIONS beginnen. Dort muss dann der Operand LOGGING-ALLOWED = *PARAMETERS(...) angegeben werden, mit CMD = *NO und/oder DATA= *NO.

Wenn nur für Teile der Prozedur Protokollierung von Kommandos und/oder Daten unterbunden werden soll, muss im Prozedurrumpf das Kommando MODIFY-PROCEDURE-OPTIONS aufgerufen und der entsprechende Wert im Operanden LOGGING-ALLOWED = *PARAMETERS(...) angegeben werden. Die Einstellung (*YES/*NO) bleibt immer so lange erhalten, bis sie erneut mit dem Kommando MODIFY-PROCEDURE-OPTIONS geändert wird.

Wenn für Teile der Prozedur Protokollieren unterbunden ist, kann diese Einstellung nicht „von außen“ verändert werden, das heißt, nicht über die Prozeduraufruf-Kommandos oder Kommandos im Dialog.

Protokollieren sollte zum Beispiel unterbunden werden, wenn der Aufrufer bestimmte Kommandos oder Daten nicht „sehen“ soll.

5.1.4.2 Protokollierung des normalen Prozedurablaufs

Bei Vordergrund-Prozeduren legt der Aufrufer beim Prozeduraufruf fest, ob ein Protokoll des Prozedurablaufs erstellt werden soll, und zwar über den Operanden LOGGING. Es spielt dabei keine Rolle, ob die Prozedur mit CALL- oder INCLUDE-PROCEDURE aufgerufen wird. Bei CALL- und INCLUDE-PROCEDURE hat der Operand LOGGING dieselbe Funktion: über CMD = *YES/*NO stellt der Aufrufer ein, ob Kommandos protokolliert werden, über DATA = *YES/*NO, ob Daten protokolliert werden.

5.1.4.3 Protokollierung des Prozedur-Testlaufs

Mit dem Kommando MODIFY-PROCEDURE-TEST-OPTIONS kann für alle Prozeduren einer Task global eingestellt werden, dass protokolliert werden soll. Diese Einstellung gilt für alle Schachtelungstiefen.

Im Dialog kann der Benutzer mit dem Kommando TRACE-PROCEDURE den Prozedurablauf schrittweise verfolgen. Alle Kommandos, die nach dem Aufruf von TRACE-PROCEDURE bis zur nächsten Unterbrechung ausgeführt werden, werden automatisch protokolliert (sofern Protokollieren zulässig).

5.1.4.4 Einschränkungen bei Prozeduren mit Lesekennwort

Der Versuch, eine Prozedur zu protokollieren, gilt als lesender Zugriff. Ein lesender Zugriff auf eine Datei mit Lesekennwort verlangt, dass das Kennwort mit dem Kommando ADD-PASSWORD in die aktuelle Kennwortliste eingetragen wurde. Ist dies nicht der Fall, so wird im System der Zähler für Fehlversuche erhöht und die Task beim Erreichen der in der Class-2-Option angegebenen Grenze für PWERRORS abgebrochen. Ein solches Verhalten könnte aber für den Benutzer eine unzumutbare Restriktion bedeuten. Nämlich wenn er die Protokollierung mit MODIFY-PROCEDURE-TEST-OPTIONS global einschaltet und sich in seiner Aufrufhierarchie kennwortgeschützte Prozeduren befinden, deren Kennwort er nicht kennt.

Es gilt daher, dass mit Lesekennwort geschützte Prozeduren grundsätzlich nicht über MODIFY-PROCEDURE-TEST-OPTIONS protokolliert werden können, auch dann nicht, wenn das Kennwort in der aktuellen Kennwortliste vorhanden ist. D.h. es wird kein Versuch eines Lesezugriffs unternommen und auch kein Strafzähler erhöht. Das Protokollieren dieser Prozeduren ist also nur direkt über den LOGGING-Operanden in CALL-PROCEDURE oder in INCLUDE-PROCEDURE möglich. Ein Fehlzugriff erhöht in diesem Fall aber auch den Strafzähler. Dasselbe gilt für das Kommando TRACE-PROCEDURE.

5.1.4.5 Inhalt der Protokollierung

Prozedurintern und beim Aufruf von Hintergrund-Prozeduren kann angegeben werden, ob die aktuelle Angabe sich auf Kommandos oder auf Daten (oder beide) beziehen soll. Die Operanden LOGGING-ALLOWED bzw. LOGGING haben bei allen entsprechenden Kommandos die gleiche Form:

Soll sich die Angabe sowohl auf Kommandos als auch auf Daten beziehen, muss der Operand in der Form LOGGING[-ALLOWED] = *YES bzw. *NO angegeben werden.
Soll sich die Angabe nur auf Kommandos beziehen, muss der Operandenwert in der Form LOGGING[-ALLOWED] = *PARAMETERS(CMD = *YES) bzw. (CMD = *NO) angegeben werden.
Entsprechend gilt, wenn sich die Angabe nur auf das Protokollieren von Daten beziehen soll, der Operandenwert *PARAMETERS(DATA = *YES) bzw. (DATA = *NO).

Die nachfolgende Tabelle zeigt die Wechselwirkungen der Einstellungen in den Kommandos SET-PROCEDURE-OPTIONS, MODIFY-PROCEDURE-OPTIONS, CALL-PROCEDURE / INCLUDE-PROCEDURE.

Ob Kommandos oder Daten aktuell protokolliert werden, kann mit der vordefinierten Funktion LOGGING-MODE( ) abgefragt werden, und zwar mit dem Operanden STREAM = *CMD/*DATA.

SET-PROC-OPTION / MOD-PROC-OPTION	**CALL-PROCEDURE bzw. INCLUDE-PROCEDURE Operand LOGGING =**					
Operand LOGGING-ALLOWED=	*YES	*NO	*PARAM (CMD = *Y)	*PARAM (CMD = *N)	*PARAM (DATA = *Y)	*PARAM (DATA = *N)
*YES	K/D	-/-	K/*	-/*	*/D	*/-
*NO	-/-	-/-	-/-	-/-	-/-	-/-
*PAR(CMD= *Y)	K/D	-/-	K/*	-/*	*/D	*/-
*PAR(CMD = *N)	-/D	-/-	-/*	-/*	*/D	-/-
*PAR(DATA = *Y)	K/D	-/-	K/*	-/*	*/D	*/-
*PAR(DATA = *N)	K/-	-/-	K/*	-/-	*/-	*/-

K/D	Kommandos und Daten werden protokolliert
-/-	weder Kommandos noch Daten werden protokolliert
K/-	nur Kommandos werden protokolliert
-/D	nur Daten werden protokolliert
*	ob Kommandos/Daten protokolliert werden, hängt von der Einstellung im Kommando MODIFY-PROCEDURE-TEST-OPTIONS ab.

Hinweise

- Wenn S-Prozeduren protokolliert werden, weist das Protokoll immer Zeilennummer und Prozedurebene aus, so dass im Fehlerfall der Fehler leicht lokalisiert werden kann.
- Wenn Schleifen protokolliert werden, wird das Kommando, das die Schleife einleitet, nur einmal protokolliert. Das Kommando, das die Schleife abschließt, wird bei jedem Schleifendurchlauf protokolliert.
- Wenn vor dem Aufruf eines Kommandos eine S-Marke steht (marke:), werden Kommando und Marke getrennt protokolliert.
- Kommentar-Kommandos werden nicht protokolliert. (Kommentar-Kommandos sind Kommandozeilen, die nur aus einem Kommentar bestehen.)

5.1.5 Programme entladen

Mit dem Operanden UNLOAD-ALLOWED legt der Aufrufer fest, ob ein Programm, das zum Ablaufzeitpunkt geladen ist, entladen werden darf. Darf das Programm nicht entladen werden (UNLOAD-ALLOWED = *NO), ist beim Prozedur-Ablauf kein Kommando erlaubt, das ein Programm entlädt.
Wird versucht, ein Programm zu entladen, obwohl es nicht zulässig ist, führt dies zum Fehler.

Ein solches Kommando ist zum Beispiel START-EXECUTABLE-PROGRAM, das ein Programm startet. Es kann jedoch immer nur ein Programm geladen sein. Wenn noch ein anderes Programm geladen ist, wird dieses erste Programm entladen, bevor das zweite Programm mit START-EXECUTABLE-PROGRAM geladen und gestartet wird.

5.1.6 Ausführungsmodus einstellen

Mit dem Operanden EXECUTION in den Kommandos CALL-PROCEDURE und INCLUDE-PROCEDURE kann der Ausführungsmodus für die Prozedur eingestellt werden.

EXECUTION = *YES ist die Standardeinstellung: die Prozedur wird nach der Voranalyse sofort ausgeführt.

Die Einstellung EXECUTION = *NO ist nützlich, wenn die Prozedur beim Testen nicht vollständig ablaufen soll. Wenn diese Einstellung gilt, werden zunächst die Kommandos des Prozedurkopfs ausgeführt. Anschließend wird der Prozedurrumpf voranalysiert, das heißt, es wird geprüft, ob die Kontrollstrukturen korrekt sind. Ausgeführt wird der Prozedurrumpf jedoch noch nicht.

5.1.7 Fehler durchreichen

„Durchreichen von Fehlern“ bedeutet, dass von einer untergeordneten Prozedur Informationen über aufgetretene Fehler an die übergeordnete Prozedur geliefert werden.

Fehlerinformationen können in Vordergrund-Prozeduren über verschiedene Mechanismen von der untergeordneten aufgerufenen Prozedur an die übergeordnete aufrufende Prozedur durchgereicht werden:

– Kommando EXIT-PROCEDURE
– Fehlerinformationen über Variablen durchreichen.

Durchreichen über EXIT-PROCEDURE

1. Im Kommandoaufruf werden Fehlerinformationen zu den Fehlerklassen Subcode1, Subcode2 und Maincode angegeben (z.B. durch Variablenersetzung; in die Variablen wurde zuvor der Kommando-Returncode gespeichert oder Angabe „eigener“ Fehlerinformationen für Fallunterscheidungen)
2. In der aufrufenden Prozedur werden diese Komponenten dann mit den eingebauten Funktionen SUBCODE1( ), SUBCODE2( ) und MAINCODE( ) ausgewertet.

Über den Operanden ERROR des Kommandos EXIT-PROCEDURE können Informationen zu Fehlern, die während des Prozedurlaufs auftraten und über die interne Fehlerbehandlung abgefangen wurden, an den Aufrufer zurückgegeben werden. Wenn eine Prozedur fehlerhaft abgebrochen wird, wird die Fehlerinformation automatisch weitergereicht, da die Fehlersituation nicht beendet wird.

Durchreichen über Variablen

1. Die Variable muss sichtbar sein: In Call-Prozeduren muss die Variable als taskglobal importiert sein; in Include-Prozeduren kann direkt auf Variablen der aufrufenden Prozedur zugegriffen werden.
2. Im Fehlerfall wird der Variablen ein entsprechender Wert zugewiesen.
3. In einem IF-Block wird entschieden, ob der Prozedurlauf fortgesetzt oder sofort beendet wird.
4. Nach Beendigung der untergeordneten Prozedur wird die Variable in der übergeordneten aufrufenden Prozedur ausgewertet.

5.1.8 Prozedur beenden

Prozeduren können auf verschiedene Weise beendet werden:

– mit dem Kommando EXIT-PROCEDURE
– mit dem Kommando END-PROCEDURE
– mit dem Kommando CANCEL-PROCEDURE

Das Kommando END-PROCEDURE wird nur aus Kompatibilität mit Nicht-S-Prozeduren unterstützt. Wenn eine Prozedur mit END-PROCEDURE abgeschlossen wird, können keine Fehlerinformationen weitergereicht oder ein Programm fortgesetzt werden.

Mit dem Kommando CANCEL-PROCEDURE kann der gesamte Prozedurlauf abgebrochen werden. SYSCMD wird wieder auf die Primärzuweisung gesetzt. Wenn CANCEL-PROCEDURE in einer untergeordneten Prozedur aufgerufen wird, werden auch alle übergeordneten Prozeduren abgebrochen.

Enthält die Prozedur kein Abschluss-Kommando, wird sie automatisch nach Ausführung des letzten Kommandos beendet. Im Fehlerfall wird eine Fehlermeldung an den Aufrufer zurückgegeben.

Beenden mit EXIT-PROCEDURE

S-Prozeduren werden immer mit dem Kommando EXIT-PROCEDURE abgeschlossen. EXIT-PROCEDURE beendet den Prozedurlauf, liefert Fehlerinformationen an den Aufrufer und setzt gegebenenfalls ein Programm fort.

EXIT-PROCEDURE wird nur ausgeführt, wenn die Prozedur bis zu diesem Kommandoaufruf ordnungsgemäß abgelaufen ist, das heißt, wenn sie nicht mit einem Fehler oder mit dem Kommando CANCEL-PROCEDURE abgebrochen wurde.

Mit dem Operanden RESUME-PROGRAM kann ein Programm, das zum Zeitpunkt der Prozedurbeendigung geladen ist, fortgesetzt werden.

Da in Vordergrund-Prozeduren die SYSFILE-Umgebung des Aufrufers verändert werden kann (SYSTEM-FILE-CONTEXT=*SAME-AS-CALLER), muss der Aufrufer nach Prozedur-Beendigung sicherstellen, dass er in der korrekten SYSFILE-Umgebung arbeitet.

(Zur Lieferung von Fehlerinformationen an den Aufrufer siehe Abschnitt „Fehler durchreichen“ auf Seite 118.)

5.2 S-Prozeduren im Hintergrund aufrufen

Eine S-Prozedur im Hintergrund aufrufen (asynchroner Prozeduraufruf) bewirkt, dass die Prozedur unabhängig vom aufrufenden Auftrag läuft; es wird ein neuer Auftrag mit eigener Auftragsnummer (TSN) erzeugt.
In vieler Hinsicht verhalten sich Hintergrund-Prozeduren wie Vordergrund-Prozeduren. Im Folgenden werden nur die Besonderheiten für Hintergrund-Prozeduren dargestellt.

5.2.1 Das Aufrufkommando ENTER-PROCEDURE

Wenn eine S-Prozedur als Hintergrund-Prozedur gestartet werden soll, muss sie mit dem Kommando ENTER-PROCEDURE gestartet werden. Dabei wird intern eine Enter-Datei erzeugt und mit dem Kommando ENTER-JOB gestartet.

Verfahren

1. Die Prozedurdatei wird als Kopie unter dem Namen `S.PROC.tsn.datum.uhrzeit` angelegt, wobei *datum* das Format yyyy-mm-dd und *uhrzeit* das Format hh.mm.ss besitzen.
2. Es wird eine Enter-Datei unter dem Namen `S.E.tsn.datum.uhrzeit` mit folgendem Inhalt angelegt:

   ```
   /SET-LOGON-PARAMETERS
   :
   :
   :
    /CALL-PROCEDURE FROM-FILE=S.PROC.tsn.datum.uhrzeit, -
   /                          PROCEDURE-PARAMETERS=(parameter)
   :
   :
   :
   /EXIT-JOB SYSTEM-OUTPUT=option
   ```

   Der Wert von *parameter* entspricht der Angabe im Operanden PROCEDURE-PARAMETERS. (Prozedurparameter können nur als Werte übergeben werden (*BY-VALUE)).
   Die Kopie der Prozedurdatei wird nach Prozedurablauf gelöscht (wenn die Hintergrund-Prozedur nicht wiederholt werden soll). Der Wert von *option* entspricht der Angabe im Operanden SYSTEM-OUTPUT.
3. Die Enter-Datei wird mit ENTER-JOB gestartet. Die Angaben für die Operanden PROCESSING-ADMISSION, JOB-CLASS, JOB-NAME, MONJV, JV-PASSWORD, JOB-PRIORITY, RERUN-AFTER-CRASH, FLASH-AFTER-SHUTDOWN, START, REPEAT-JOB, RESOURCES, LISTING und JOB-PARAMETER werden entsprechend in das ENTER-JOB-Kommando übernommen.

Die Operanden, die die Eigenschaften der Hintergrund-Prozedur festlegen, sind i Abschnitt „Auftragseigenschaften“ auf Seite 122 beschrieben.

Hinweise

– „Remote Enter“ wird unterstützt, das heißt, die Prozedur kann mit dem Kommando ENTER-PROCEDURE auch an einem entfernten Rechner aufgerufen werden.
– Das Kommando ENTER-PROCEDURE kann nicht von einem Bedienplatz aus aufgerufen werden.

5.2.2 Prozedurbehälter angeben

Der Prozedurbehälter wird im Operanden FROM-FILE benannt. Als Prozedurbehälter kann nur eine BS2000-Datei oder ein Bibliothekselement angegeben werden. Prozeduren, die im Hintergrund aufgerufen werden sollen, können nicht in Listenvariablen übergeben werden, da die Variable nur in der aufrufenden, nicht in der ausführenden Prozedur bekannt wäre.

Ist die Prozedur in einer Datei unter einer fremden Benutzerkennung gespeichert, muss sie ausführend mehrbenutzbar katalogisiert sein, sonst kann auf sie nicht zugegriffen werden.

Wenn die Prozedurdatei durch ein Kennwort geschützt ist, muss dieses Kennwort im Operanden PROCEDURE-PASSWORD oder durch ein Kommando ADD-PASSWORD angegeben werden.

5.2.3 Prozedurparameter übergeben

Im Operanden PROCEDURE-PARAMETERS können Prozedurparameter nur als Werte übergeben werden (im Kommando DECLARE-PARAMETER deklariert mit TRANSFER-TYPE = *BY-VALUE). Variablennamen können nicht übergeben werden, da Variablen nur im aufrufenden Auftrag bekannt sind, nicht im ausführenden.

Ob Aktual- und Formalparameter übereinstimmen, wird erst zum Ablaufzeitpunkt geprüft.

5.2.4 Protokollierung veranlassen

Der Operand LOGGING hat im Aufruf-Kommando von Hintergrund-Prozeduren, also bei ENTER-PROCEDURE, eine andere Form und damit andere Auswirkungen als in den Kommandos CALL-PROCEDURE und INCLUDE-PROCEDURE.

Bei ENTER-PROCEDURE hat der Operand folgende Funktion: über *YES bzw. *NO stellt der Aufrufer die Protokollierung des Auftragablaufs ein bzw. aus, über *STD stellt er sie nur in dem Fall ein, falls die Datei nicht lesegeschützt ist.

Im Hintergrund aufgerufene Prozeduren werden dazu standardmäßig auf SYSOUT protokolliert. Wie viele Sätze nach SYSLST (oder SYSOUT) geschrieben werden, kann im Kommando ENTER-PROCEDURE mit dem Operanden SYSLST-LIMIT (bzw. SYSOUT-LIMIT) eingestellt werden.

5.2.5 Auftragseigenschaften

Das Kommando ENTER-PROCEDURE enthält eine Reihe von Operanden, die sich auf die Hintergrund-Prozedur beziehen. Diese Operanden lassen sich zu folgenden Gruppen zusammenfassen:

- Job-Eigenschaften (Operanden JOB-CLASS, JOB-PRIORITY, JOB-NAME, JOB-PARAMETER)
- überwachende Jobvariablen (Operanden MONJV, JV-PASSWORD)
- Startverhalten (Operanden START, REPEAT-JOB, RERUN-AFTER-CRASH, FLUSH-AFTER-SHUTDOWN)
- Ressourcen (Operand RESOURCES(RUN-PRIORITY, CPU-LIMIT, SYSLST-LIMIT bzw. SYSOPT-LIMIT))

5.2.5.1 Job-Eigenschaften einschließlich überwachende Jobvariablen einstellen

Zu den Job-Eigenschaften der Hintergrund-Prozedur zählen:

- Job-Klasse
- Priorität des Auftrags
- Job-Name
- Job-Parameter
- Überwachende Jovariablen

Die Eigenschaften werden mit den entsprechenden Operanden eingestellt. Welche Werte der Benutzer einsetzen darf, kann er mit dem Kommando SHOW-USER-ATTRIBUTES oder SHOW-JOB-CLASS ermitteln (Beschreibung dieser Kommandos siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

Hinweise

- Den Jobnamen kann der Benutzer frei vergeben, er darf jedoch nur maximal acht Zeichen lang sein.
- Über die Jobparameter werden weitere Jobklassenattribute gekennzeichnet, deren Bedeutung von der Systembetreuung festgelegt wird.

5.2.5.2 Startverhalten einstellen

Das Startverhalten der Hintergrund-Prozedur kann über folgende Operanden eingestellt werden: START, REPEAT-JOB, RERUN-AFTER-CRASH, FLUSH-AFTER-SHUTDOWN.

Mit dem Operanden START wird der Startzeitpunkt der Prozedur festgelegt. Mögliche Werte sind eine absolute Datumsangabe oder eine relative Zeitangabe, z.B. sofort, sobald wie möglich, innerhalb eines bestimmten Zeitraums usw.

Mit dem Operanden REPEAT-JOB kann ein Rhythmus definiert werden, in dem die Prozedur wiederholt gestartet werden soll, z.B. täglich, wöchentlich usw.

Mit dem Operanden RERUN-AFTER-CRASH kann festgelegt werden, ob eine Prozedur automatisch neu gestartet werden soll, wenn sie auf Grund eines Systemfehlers oder beim Shutdown des Systems abgebrochen wurde.

Der Operand FLUSH-AFTER-SHUTDOWN wird ausgewertet, wenn die Hintergrund-Prozedur während des aktuellen Systemlaufs nicht gestartet wurde. Mit FLUSH-AFTER-SHUTDOWN kann festgelegt werden, ob die Hintergrund-Prozedur aus der Auftragsschlange entfernt werden soll, wenn sie bis Systemlaufende nicht gestartet wurde.

5.2.5.3 Nutzung der Ressourcen festlegen

Die Nutzung der System-Ressourcen wird über den Operanden RESOURCES eingeschränkt. Er legt die Run-Priorität fest, die Grenze für CPU-Zeit, die die Hintergrund-Prozedur verbrauchen darf, und die Anzahl der Sätze, die in SYSLST- und SYSOUT-Dateien ausgegeben werden.

5.2.6 Fehler durchreichen

Im Hintergrund aufgerufene Prozeduren laufen in eigenen Aufträgen; Fehlerinformationen können daher nicht über Variablen oder Kommando-Returncodes weitergegeben werden.

Hier bieten sich zur Aufnahme von Fehlerinformationen Jobvariablen an. Kompatibel zu Nicht-S-Prozeduren können auch in S-Prozeduren Benutzerschalter verwendet werden. Sowohl auf Jobvariablen als auch auf Benutzerschalter können andere Aufträge zugreifen.

Über Jobvariablen kann eine differenziertere Fehlerinformation weitergegeben werden, da der Inhalt der Jobvariablen bis zu 256 Zeichen lang sein darf. (Näheres zu Jobvariablen: Siehe Handbuch „Jobvariablen“ [5].)

5.2.7 Prozedur beenden

Auch Hintergrund-Prozeduren werden entweder mit den Kommandos EXIT-PROCEDURE, END-PROCEDURE oder CANCEL-PROCEDURE beendet.

Nach Beendigung einer im Hintergrund aufgerufenen Prozedur wird auch der Stapelauftrag, in dem sie ablief, beendet. Dabei werden die auftragsüberwachenden Jobvariablen bei ordnungsgemäßem Ablauf auf $T, bei Abbruch mit Fehler auf $A gesetzt. Als „Abbruch mit Fehler“ gilt ein Abbruch über CANCEL-JOB oder die explizite Beendigung über EXIT-JOB MODE = *ABNORMAL (bzw. ISP-Kommando ABEND), nicht jedoch der Aufruf des Kommandos EXIT-PROCEDURE ERROR = *YES!

5.3 S-Prozeduren schachteln

Wenn innerhalb einer Prozedur eine weitere Prozedur aufgerufen wird, wird diese Form des Prozeduraufrufs als „geschachtelter Aufruf“ bezeichnet. Die so entstehende Prozedurverknüpfung heißt dementsprechend Prozedurschachtelung.

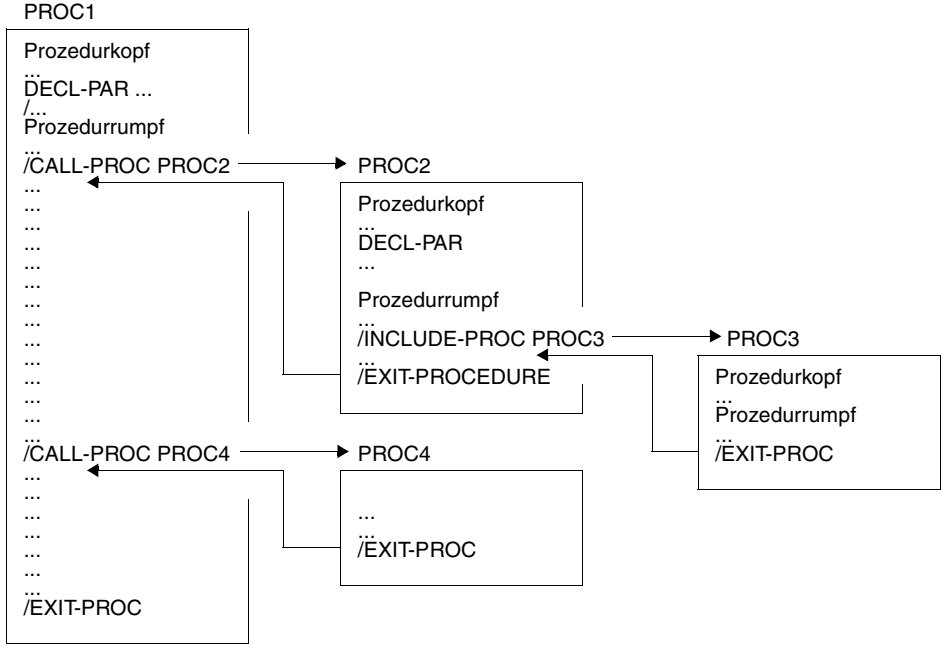
Zu beachten ist, dass Prozeduren zwar geschachtelt aufgerufen werden dürfen; es dürfen jedoch nicht Prozeduren ineinander „geschrieben“ werden. Das heißt, jeder Prozedurbehälter kann immer nur eine Prozedur enthalten.

Werden geschachtelte Prozeduren aufgerufen, wird die innere Prozedur vollständig abgearbeitet, bevor sie die Steuerung an die aufrufende (= übergeordnete) Prozedur zurückgibt. Die gerufene Prozedur ist der rufenden Prozedur untergeordnet.

Für die Prozedur-Schachtelung gibt es drei Kommandos:

– CALL-PROCEDURE und INCLUDE-PROCEDURE für den Prozeduraufruf
– EXIT-PROCEDURE für die Prozedur-Beendigung

Das folgende Schema zeigt ein Beispiel für eine Prozedurschachtelung.



In diesem Beispiel wird PROC1 von der Systemebene aus aufgerufen; PROC1 ist die übergeordnete Prozedur für PROC2, PROC3 und PROC4. PROC2, PROC3 und PROC4 werden aus einer Prozedur heraus aufgerufen, sind also untergeordnete Prozeduren: PROC2 und PROC4 sind nur PROC1 untergeordnet, PROC3 ist PROC2 und PROC1 untergeordnet. PROC2 wiederum ist PROC3 übergeordnet.

Geschachtelte Prozeduren sind immer im Vordergrund aufgerufene Prozeduren. Daher gelten für Aufruf (bzw. Start), Ablauf und Beendigung die gleichen Regeln wie für Vordergrund-Prozeduren, die von der Systemebene aus aufgerufen werden.

Geschachtelte Prozeduren dürfen Hintergrund-Prozeduren aufrufen. Diese Prozeduren werden allerdings nicht in der Aufrufer-Hierarchie geschachtelt, sondern leiten einen neuen Auftrag mit einer eigenen unabhängigen Prozedurschachtelung ein.

Wichtig beim Ablauf geschachtelter Prozeduren ist der gegenseitige Zugriff auf Variablen, das heißt auf Variablen, die in der über- oder in der untergeordneten Prozedur deklariert sind.

Ob Variablen in geschachtelten Prozeduren zugreifbar, das heißt sichtbar sind, hängt zum einen von der Deklaration des Geltungsbereichs der Variablen ab, zum anderen davon, mit welchem Kommando die Prozedur aufgerufen wird.

SDF-P bietet folgende Möglichkeiten, auf Variablen aus übergeordneten Prozeduren zuzugreifen:

– Variablenname beim Prozeduraufruf als Prozedurparameter übergeben.
– Taskglobale Variablen, die nicht sichtbar sind, neu deklarieren. Das bedeutet, dass jede dieser Variablen mit einem DECLARE-VARIABLE-Kommando deklariert werden muss, und zwar mit genau den gleichen Attributen wie in der übergeordneten Prozedur.
– Taskglobale Variablen mit dem Kommando IMPORT-VARIABLE importieren.

Die Zusammenhänge zwischen der Sichtbarkeit von Variablen und dem Prozeduraufruf-Kommando sowie das „Neu-Deklarieren“ und das Importieren von Variablen sind detailliert im Abschnitt „Geltungsbereich von Variablen“ auf Seite 162 beschrieben.

Wie Variablennamen als Prozedurparameter übergeben werden, ist auf Seite 108 beschrieben.

5.4 Interne Unterprozeduren

Ein Kommandoblock, der mit einem BEGIN-BLOCK-Kommando beginnt und mit einem END-BLOCK-Kommando endet (auch BEGIN-Block genannt, siehe Seite 573), kann innerhalb der Prozedur als Unterprozedur genutzt werden. Der Aufruf einer solchen Unterprozedur erfolgt mit dem Kommando INCLUDE-BLOCK (siehe Seite 690). Die Prozedur verzweigt zu der Kommandozeile, die mit der angegebenen Marke beginnt und das entsprechende BEGIN-BLOCK-Kommando enthält. Nach Abbarbeitung des BEGIN-Blocks wird die Prozedur mit dem Kommando fortgesetzt, das auf das INCLUDE-BLOCK-Kommando folgt.

Die Verwendung von Unterprozeduren ist vergleichbar mit anderen höheren Programmiersprachen. Sie fördert die Übersichtlichkeit und Wartungsfreundlichkeit der Prozedur. Der Aufruf einer Unterprozedur ist auch performanter als der Aufruf einer „externen“ Prozedur, bei der erst ein weiterer Prozedurbehälter geöffnet werden muss.

5.5 Prozedur unterbrechen

Der Aspekt Prozedur-Unterbrechung gilt nur für Prozeduren, die im Dialog bzw. im Vordergrund aufgerufen werden (mit CALL- oder INCLUDE-PROCEDURE). Zu unterscheiden sind Unterbrechungen von der Systemebene und prozedurinterne Unterbrechungen.

Prozedurintern kann der Prozedurlauf jederzeit mit dem Kommando HOLD-PROCEDURE unterbrochen werden. Von der Systemebene aus kann eine Prozedur mit der Funktionstaste K2 unterbrochen werden, wenn dies zulässig ist. Ob eine Prozedur unterbrechbar ist, kann bereits im Prozedurkopf eingestellt werden, im Kommando SET-PROCEDURE-OPTIONS, Operand INTERRUPT-ALLOWED. Diese Einstellung kann später im Prozedurrumpf mit dem Kommando MODIFY-PROCEDURE-OPTIONS verändert werden, ebenfalls mit dem Operanden INTERRUPT-ALLOWED. Auf diese Weise kann für einzelne Teile der Prozedur Unterbrechbarkeit ein- oder ausgeschaltet werden. Beim Prozeduraufruf kann die Einstellung für die Unterbrechbarkeit nicht geändert werden.

Bei beiden Arten der Unterbrechung (mit dem Kommando HOLD-PROCEDURE und der Funktionstaste K2) wird der Prozedurlauf fortgesetzt, wenn auf der Systemebene das Kommando RESUME-PROCEDURE eingegeben wird.

Wenn eine Prozedur nicht unterbrechbar ist, während des Prozedurlaufs jedoch mit der Funktionstaste K2 eine Unterbrechung angefordert wird, wird dies ignoriert.

Während der Prozedurunterbrechung kann der Benutzer auf Systemebene im Dialog zum Beispiel Kommandos aufrufen, um die SYSFILE-Umgebung zu prüfen oder zu verändern. Er kann auf alle prozedurlokalen Variablen der unterbrochenen Prozedur zugreifen. Der Unterbrechungsdialog gehört also zum Sichtbarkeitsbereich der unterbrochenen Prozedur.

Während eines Unterbrechungsdialogs können auch Variablen deklariert werden, die anschließend als prozedurlokale Variablen der unterbrochenen Prozedur gelten. Wenn zum Beispiel innerhalb einer Prozedur versucht wird, auf eine Variable zuzugreifen, die nicht deklariert ist, kann im Rahmen der entsprechenden Fehlerbehandlung die Prozedur unterbrochen und die fehlende Variable auf der Systemebene deklariert werden.

5.6 Nichtunterbrechbarkeit

Prozeduren und Programme gegen Unterbrechung sichern

Durch die Operandeneinstellung INTERRUPT-ALLOWED=*NO bei SET-PROCEDURE-OPTIONS bzw. MODIFY-PROCEDURE-OPTIONS wird eine Prozedur gegen Unterbrechungen von Kommando-Eingaben im Dialog-Modus geschützt.

Wenn ein Programm in einer Prozedur aktiviert wird, muss die Prozedur auch vor unkontrollierten Kommando-Eingaben geschützt werden: z.B. wenn das Programm Daten oder Anweisungen verabeitet und einige davon das Programm veranlassen, Kommandos auszuführen (CMD-Makro) oder eine Unterbrechung verursachen (BKPT-Makro), die bei der Prozedur nicht vorgesehen ist.

Beispiel

```
/SET-PROCEDURE-OPTIONS ...,INTERRUPT-ALLOWED=NO
/ASSIGN-SYSDTA TO=*PRIMARY
/START-LMS
                        ----------> Dialog am Terminal:
                                    //
                                    //...
                                    //END
/MODIFY-JOB-SWITCHES ON=(4,5)
/START-EDT
                        ----------> Dialog am Terminal:
                                   *
                                    * ...
                                   *
/EXIT-PROC
```

Ist eine Prozedur vor Unterbrechung geschützt, so wird normalerweise auch ein in dieser Prozedur ablaufendes Programm gegen Unterbrechung durch die Funktionstaste K2 geschützt. Ist im Programm jedoch eine STXIT-Routine für K2-Unterbrechung definiert (ESCPBRK), so wird das Ereignis K2-Unterbrechung an diese STXIT-Routine durchgereicht. Die Verantwortung für die Behandlung dieses Ereignisses liegt dann beim Programm. Allerdings führt das zu Problemen, wenn das Programm mit einem BKPT-Makro reagiert.

Beispiel

```
/SET-PROCEDURE-OPTIONS ...,INTERRUPT-ALLOWED=NO
/START-EXECUTABLE-PROGRAM ...
//
//...                                              <--------- K2
K2-STXIT>  Ausgabe PC
                     BKPT
                     "Zurück zur Kommandoebene"
/EXIT-PROCEDURE
```

5.6.1 Prozeduren implizit schützen

Wenn ein Programm in einer Prozedur gestartet wird, sollte das Programm prüfen, ob die Prozedur gegen Unterbrechungen geschützt ist; d.h. das Programm sollte die Prozedur implizit gegen Unterbrechungen schützen, die von ihm während seines Ablaufs verursacht werden könnten.

Trotzdem können solche Aktionen wie CMD, BKPT, STXIT etc. vom Programm durchgeführt werden: erstens für interne Ziele und zweitens wenn diesbezügliche Benutzeranforderungen vorliegen.
Vor Ausführung der Aktion sollte das Programm mit dem Makroaufruf CLIGET die Prozedureigenschaft INTERRUPT-ALLOWED abgefragen. Ist *NO eingestellt, sollte die Aktion zurückgewiesen werden.

Programme können immer durch Prozedur-Kommandos, wie HOLD-PROGRAM, unterbrochen werden.

5.6.2 Programme explizit sichern

Wenn Benutzerprogramme sicherheitsrelevante Daten verarbeiten, müssen sie gegen unkontrollierte Eingaben von Kommandos gesichert werden. Dieser Schutz muss in allen Eingabe-Modi (bei Vordergrund- und Hintergrund-Prozeduren) gewährleistet sein. Darum muss in den Programmen selbst diese Einstellung vorgenommen werden, wenn sie sicherheitskritische Teile enthalten. Man nennt dies auch expliziten Programmschutz. Dieser wird durch den Makroaufruf CLISET aktiviert. (Siehe dazu auch Kapitel „Programmschnittstellen“ auf Seite 311.) Damit sind die Programme selbst verantwortlich für Aufrufe wie CMD, BKPT, K2-STXIT etc., wenn sicherheitsrelevante Informationen verarbeitet werden.

Annahme und Zurückweisung von Ereignissen

Programm	**nicht unterbrechbar**	**unterbrechbar**	
Prozedur	**beliebig**	**nicht-unterbrechbar**	**unterbrechbar**
Taste K2	z	z	a
K2-STXIT	a,ze	a,zi	a
CMD-Makro	a,ze	a,zi	a
BKPT-Makro	a,ze	a,zi	a
andere Makros	a,ze	a,zi	a
//EXEC-SYS-CMD	z	c	a
//HOLD-PROGRAM	z	c	c
/HOLD-PROGRAM	z*	a*	a*

Erklärung:

a: vom System angenommen

ze: soll vom Programm bei expliziten Programmschutz zurückgewiesen werden (CLISET)

zi: soll vom Programm bei impliziten Programmschutz zurückgewiesen werden (CLIGET)

z: vom System zurückgewiesen

c: vom System zurückgewiesen, wenn SYSSTMT ungleich SYSCMD ist

a*: vom System als Daten betrachtet, wenn SYSSTMT bzw. SYSDTA ungleich SYSCMD ist. Ansonsten wird es vom System angenommen.

z*: vom System als Daten betrachtet, wenn SYSSTMT bzw. SYSDTA ungleich SYSCMD ist. Ansonsten wird es vom System zurückgewiesen.

ze und zi sind unter Programmverantwortung.

Zurückweisungen von Ereignissen

– K2-Unterbrechung wird einfach ignoriert, die Prozesse laufen ohne Beeinträchtigung weiter.
– /HOLD-PROGRAM, //HOLD-PROGRAM und die Einstellung PROGRAM-INPUT= *MIXED-WITH-CMD geben EOF an das Programm zurück.
– //EXECUTE-SYSTEM-COMMAND wird zurückgewiesen und Spin-Off wird für Anweisungen aktiviert.
– //HOLD-PROGRAM wird in jedem Fall zurückgewiesen, wenn SYSSTMT nicht SYSCMD zugewiesen ist.

Koexistenz verschiedener Schutzmodi

Die beiden Schutzmodi (implizit und explizit) sind zwei voneinander zu unterscheidende Funktionen. Sie koexistieren und überschneiden sich bei einigen Ereignissen.

Ein Schutzmodus kann nicht vererbt werden. Trotzdem gibt es dazu äquivalente Effekte, die die Verwendung des Begriffs Vererbung sinnvoll erscheinen lassen.

Denn

- expliziter Programmschutz beinhaltet impliziten Programmschutz,
- impliziter Programmschutz beinhaltet Prozedurschutz
- Prozedurschutz beinhaltet impliziten Programmschutz. Allerdings nur, wenn dieser Schutz vom Programm durch den Makroaufruf CLIGET selbst geleistet wird.

Hinweise

- Da der explizite Programmschutz durch einen SVC realisiert wird, kann das Programm im Kommandomodus mit [K2] unterbrochen werden, bevor der SVC ausgeführt wird: z.B. wenn [K2] während der Ausführung von LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM) gedrückt wird.
  Wenn in einem Programm vermieden werden soll, dass ein SVC durch Drücken von [K2] unterbrochen wird, muss eine STXIT-Routine vorhanden sein, die das [K2]-Unterbrechungsereignis abfängt.
- Um zu vermeiden, dass der SVC durch Testfunktionen (z.B. AID) abgebrochen wird, muss das Programm vor Lesezugriff geschützt werden. In diesem Fall ist nach Drücken der Taste [K2] nur noch das Kommando RESUME-PROGRAM erlaubt.
- Der explizite Programmschutz kann auch im Nicht-Prozedurmodus (bei Vordergrund- wie Hintergrundprozessen) eingestellt werden.
- Der implizite Programmschutz ist nur im Prozedurmodus relevant. Im Nicht-Prozedurmodus kann diese Option nicht eingestellt werden.
- Wenn Programme impliziten Programmschutz unterstützen, sind mehrere unterbrechungsrelevante Aktionen - im Einklang mit der Prozedureinstellung bei INTERRUPT-ALLOWED - untersagt. Kompatibles Verhalten muss darum in den Programmen selbst richtig eingestellt sein.
- Impliziter Programmschutz kann in der Prozedur durch einen Schalter, eine Programmanweisung oder eine Parameterdatei aktiviert werden.
- Um das Programmschutzverhalten zu sichern, sollten Programme vor jeder unterbrechungsrelevanten Aktion (CMD, STXIT, BKPT etc.) die Prozedureinstellung INTERRUPT-ALLOWED abfragen.
- Eine Prozedur kann bei Vordergrundprozessen beendet werden, während ein Programm durch /HOLD-PROGRAM bzw. //HOLD-PROGRAM unterbrochen ist. In diesem Fall kann das Programm den impliziten Unterbrechungsschutz wieder zurückstellen, indem es fortwährend die CLIGET-Schnittstelle abfragt, bevor es eine unterbrechungsrelevante Aktion veranlasst.

– Der implizite Programmschutz gegen Unterbrechungen muss in der Programmbeschreibung dokumentiert werden. Wenn dies nicht der Fall ist, soll die Verwendung des Programms in nicht-unterbrechbaren Prozeduren erfasst werden.
– Eine Prozedur kann gegen Unterbrechung durch ein prozedurinternes Programm geschützt werden, das in einer K2-STXIT-Routine den Makro BKPT aufruft. Es können SDF-P Programmfunktionen benutzt werden, um die Unterbrechung abzufangen und das Programm erneut zu starten.

Beispiel

```
/SET-PROCEDURE-OPTIONS INTERRUPT-ALLOWED=*NO
/ASSIGN-SYSOUT TO=*DUMMY        "Kein Einfluss auf EDT bei write-read"
/DECLARE-VARIABLE OPS(TYPE=*STRUCTURE),-
                 /MULTIPLE-ELEMENTS=*LIST
/LOAD-EXE FROM-FILE=*LIB(LIB=&(INSTALLATION-PATH( -
                                /LOGICAL-ID='EDT', -
                                /INSTALLATION-UNIT='SYSLNK', -
                                /VERSION=*STD, -
                               /DEFAULT-PATH-NAME='EDT')), -
                        /ELEM=EDTSTRT,TYPE=L)
/EXECUTE-CMD CMD=(SHOW-JOB-STATUS),TEXT-OUTPUT=*NONE,-
            /STRUCTURE-OUTPUT=OPS,RETURNCODE=*NONE
/SHV OPS#.PROG-FILE;SHV OPS#.PROG-NAME
/WHILE (OPS#.PROG-FILE NE '')
/   RESUME-PROGRAM
/   FREE-VARIABLE OPS
/   EXECUTE-CMD CMD=(SHOW-JOB-STATUS),TEXT-OUTPUT=*NONE,-
/   STRUCTURE-OUTPUT=OPS,RETURNCODE=*NONE
/END-WHILE
```

5.7 Interne Verarbeitung von S-Prozeduren

5.7.1 Prozedur analysieren

Bevor eine Prozedur ausgeführt wird, übergibt SDF-P sie zunächst an den SDF-P-Prozedur-Interpreter. Der Prozedurkopf wird ausgeführt, der Prozedurrumpf voranalysiert und anschließend ausgeführt.

5.7.1.1 Prozedur-Interpreter

Der Prozedur-Interpreter prüft folgende Merkmale:

– Ist das erste Zeichen der Prozedur ein Schrägstrich (/)?
Prozeduren müssen mit einem Kommandoaufruf, also mit einem Schrägstrich (/) beginnen. Wenn das erste Zeichen kein Schrägstrich ist, handelt es sich nicht um eine Prozedur. Die weitere Ausführung wird abgebrochen.

– Folgt dem Schrägstrich ein SET-LOGON-PARAMETERS (bzw. LOGON)-Kommando?
Wenn ja, handelt es sich um einen Stapelauftrag. Die Ausführung wird abgewiesen.

– Folgt dem Schrägstrich das Kommando BEGIN-PROCEDURE oder PROCEDURE?
BEGIN-PROCEDURE (bzw. PROCEDURE) leitet Nicht-S-Prozeduren ein, die aufgerufene Prozedur ist demnach keine S-Prozedur. Sie wird kompatibel bearbeitet (siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

5.7.1.2 Voranalyse

Hat die Analyse der ersten Prozedurzeile ergeben, dass die aufgerufene Prozedur eine S-Prozedur ist, beginnt der Prozedur-Interpreter mit der Verarbeitung der Prozedur.

S-Prozeduren werden in folgenden Schritten verarbeitet:

1. Wenn vorhanden, wird das Kommando SET-PROCEDURE-OPTIONS ausgeführt.
(Das Kommando SET-PROCEDURE-OPTIONS darf nur als erstes Kommando einer S-Prozedur aufgerufen werden).

2. Wenn vorhanden, werden die Kommandos des DECLARE-PARAMETER-Blocks ausgeführt.
(Mehrfache Aufrufe des Kommandos DECLARE-PARAMETER müssen mit dem Kommando BEGIN-PARAMETER-DECLARATION eingeleitet und mit END-PARAMETER-DECLARATION abgeschlossen werden. Ein OPEN-VARIABLE-CONTAINER-Kommando im DECLARE-PARAMETER-Block muss auch zwischen diesen beiden Kommandos stehen).

3. Der Prozedurrumpf wird eingelesen und verarbeitet:

   a) Es werden alle Prozedurzeilen eingelesen. Die Zeilenende-Kommentare werden ausgeblendet und alle Folgezeilen ausgewertet: d.h. das Fortsetzungszeichen wird durch die nachfolgende Zeile ersetzt und der Schrägstrich (/) am Beginn der Fortsetzungszeile gelöscht.

   b) Kommandofolgen mit Kommandos, die durch Semikolon (;) voneinander getrennt sind, werden in einzelne Kommandos zerlegt.
   Ausnahme: Kommandofolgen in einem AID-Kommandoblock. Diese werden erst dann verarbeitet, wenn die Dialogtesthilfe AID den Kommandoblock ausführt (nähere Informationen zum Dienstprogramm AID enthält das Handbuch „AID“ [6]).

   c) Es wird geprüft, ob die Blockstrukturen syntaktisch korrekt sind. Außerdem wird geprüft, ob die Zielmarken der Sprung-Kommandos GOTO, EXIT-BLOCK und CYCLE existieren.

5.7.2 Prozedur verarbeiten und ausführen

Nachdem die gesamte Prozedur eingelesen und analysiert ist, wird die Prozedur gestartet und die einzelnen Kommandos ausgeführt.

Dabei werden zunächst die Ausdrücke innerhalb der Kommandos ersetzt (= Ausdrucksersetzung), dann erst wird das Kommando analysiert und anschließend ausgeführt.

Bei der Ausdrucksersetzung sind einige Einschränkungen zu beachten, die sich aus der hier beschriebenen Reihenfolge der Bearbeitungsschritte ergeben. Sie sind im Abschnitt „Ausdrucksersetzung“ auf Seite 55 beschrieben.

Prozedurablauf

Der Ablauf von S-Prozeduren wird prozedurintern über Kontrollstrukturen gesteuert. Zur internen Steuerung des Prozedurablaufs gehört auch, dass Fehler abgefangen und ausgewertet werden, so dass der Prozedurlauf anschließend ordnungsgemäß beendet oder fortgesetzt werden kann. Weitere wichtige Aspekte im Prozedurablauf sind Protokollierung und Unterbrechung der Prozedur. Während Kontrollstrukturen, Protokollierung und Fehlerbehandlung sowohl in Vordergrund- als auch in Hintergrund-Prozeduren eine wichtige Rolle spielen, können nur solche Prozeduren unterbrochen werden, die im Vordergrund aufgerufen wurden.

Kontrollstrukturen sind Schleifen oder Verzweigungen, die als Kommandoblöcke realisiert sind. Sie werden jeweils durch paarweise zusammengehörende Kontrollflusskommandos eingeleitet und abgeschlossen. Zu den Kontrollflusskommandos gehören außerdem die Sprungkommandos. Das Konzept der Kontrollstrukturen bzw. Kontrollflusskommandos gilt gleichermaßen für im Vordergrund und für im Hintergrund aufgerufene Prozeduren. Es ist im Kapitel „S-Prozeduren erstellen“ auf Seite 81 beschrieben. Eine detaillierte Beschreibung der Kommandos enthält das Kapitel „SDF-P-Kommandos“ auf Seite 545.

5.8 Kompilierte Prozeduren

Kompilierte Prozedur identifizieren

Das Kommando COMPILE-PROCEDURE schreibt in die erste Zeile einer kompilierten Prozedur die folgende Identifizierung in Textform:

```
/ ///COMPILED-PROCEDURE///
```

Der Code des Zwischenformats wird direkt nach diesem Satz generiert. Diese Identifizierung wird von SDF-P-BASYS benutzt, um die kompilierte Prozedur von der S-Prozedur zu unterscheiden.

Darüberhinaus dient sie jeder anderen Systemkomponente, jedem Dienstprogramm oder jeder Benutzeranwendung zur Kenntlichmachung, dass das gelesene Objekt als eine Prozedur aufgefasst werden muss, die mit CALL-PROCEDURE aufgerufen werden kann. Dazu benutzen S-Prozeduren und kompilierte Prozeduren dieselben Erkennungszeichen (siehe auch Seite 134): Sie müssen mit `/<kommando>` beginnen und das Kommando darf kein SET-LOGON-PARAMETERS (bzw. LOGON) oder BEGIN-PROCEDURE (bzw. PROCEDURE) sein.

Kompilierte Prozedur variabel gestalten

Im Gegensatz zu Quellprozeduren können kompilierte Prozeduren nicht mit Text-Editoren geändert bzw. modifiziert werden. Der Endbenutzer hat so keine Möglichkeit, kompilierte Prozeduren für seine Zwecke anzupassen. Darum sollte der Entwickler der Text-Prozedur unabhängige Anpassungsmechanismen in die Prozedur einbauen, d.h. z.B:

- Prozedur-Parameter benutzen
- Möglichkeiten zum Einlesen von Variablen durch Bildschirmeingabe schaffen (durch READ-VARIABLE)
- Möglichkeiten zum Einlesen von Variablenmengen aus einer Textdatei bereitstellen (durch READ-VARIABLE)
- Variablenbehälter mit einfachen Variablen einbauen
- Prozedurausgänge für S-Prozeduren durch nicht kostenpflichtige Funktionen in die kompilierte Prozedur einbauen (z.B. mit CALL-PROCEDURE oder INCLUDE-PROCEDURE)
- Job-Variable benutzen usw.

Hinweise

– S-Prozeduren, die durch kompilierte Prozeduren in Listenvariablen oder mit einem Texteditor zur Laufzeit erzeugt werden, sind nicht selbst wiederum kompilierte Prozeduren. D.h. wenn kompilierte Prozeduren ohne das Subsystem SDF-P ausgeführt werden sollen, können sie zwar andere S-Prozeduren erzeugen, die kostenpflichtige SDF-P-Funktionen benutzen, aber sie können diese zur Laufzeit nicht ausführen lassen. Desgleichen können keine kostenpflichtigen SDF-P-Funktionen benutzt werden, wenn eine kompilierte Prozedur während der Laufzeit durch die Taste K2 oder HOLD-PROCEDURE unterbrochen wird und das Subsystem SDF-P nicht geladen ist.

– Es werden nur Fehler aufgedeckt, die während der Analyse von SDF-P-Kontrollstrukturen nach SYSOUT gemeldet werden. Es werden also nicht die Eingabe-Kommandos überprüft, sondern nur die Strukturen der Prozedur.

– Wenn die Syntaxdateien der SDF-P-Kontrollflusskommandos zwischen der Kompilations-Zeit und der Laufzeit verändert werden, können inkompatible Änderungen zum Absturz der kompilierten Prozedur führen. Die Konsequenzen sind hier dieselben wie bei Text-Prozeduren, jedoch wird der Fehler nicht zur Aufrufzeit gemeldet, sondern erst zur Laufzeit der Kommandos der kompilierten Prozedur.

– Kompilierte Prozeduren werden vom Prozedur-Compiler in Behälter abgelegt. Sie können auch ohne Konsequenzen für ihre Ausführung in andere Behälter kopiert werden, allerdings nicht in Variablenbehälter.

– Nicht-S-Prozeduren können nicht kompiliert werden.

6 Variablen in S-Prozeduren verwenden

In diesem Kapitel wird zuerst das Variablenkonzept von SDF-P beschrieben. Anschließend wird dargestellt, wie Variablen in S-Prozeduren deklariert werden (dazu gehört auch eine Beschreibung der Eigenschaften von Variablen) und wie Variablen verarbeitet werden.

6.1 Variablenkonzept

Für das Programmieren von Prozeduren bietet SDF-P ein komplexes Variablenkonzept, das die Prozedurparameter des BS2000 und die Jobvariablen des Jobvariablensystems berücksichtigt.

Dieses Variablenkonzept wird in diesem Abschnitt kurz vorgestellt: zunächst die Grundlagen des Variablenkonzepts, dann die Abgrenzung von Prozedurparametern und S-Variablen und schließlich die Gegenüberstellung von S-Variablen und Jobvariablen.

6.1.1 Grundlagen des Variablenkonzepts

Variablen sind als Platzhalter für gespeicherte bzw. zu speichernde Daten wichtige Bestandteile von Prozeduren. Sie können verschiedene Werte annehmen.

Variablen in SDF-P, die auch als S-Variablen bezeichnet werden, sind eindeutig identifiziert durch ihren Variablennamen und ihren Geltungsbereich. Der Geltungsbereich bestimmt, wo auf eine Variable zugegriffen werden darf: nur in der aktuellen Prozedur oder in der gesamten Task. Ein weiteres wichtiges Merkmal von Variablen ist der Datentyp, der festlegt, welche Werte eine Variable annehmen kann.

Zu unterscheiden sind noch einfache und zusammengesetzte Variablen: Einfache Variablen können nicht weiter „unterteilt“ werden, zusammengesetzte Variablen bestehen aus Variablenelementen (Variablenelemente selbst können auch einfache oder zusammengesetzte Variablen sein). Bei den zusammengesetzten Variablen sind drei Typen zu unterscheiden: Listen, Arrays und Strukturen.

Wichtige Merkmale komfortablen Programmierens sind:

– implizite Deklaration von Variablen
– dynamisch wechselnder Datentyp
– dynamische Erweiterung zusammengesetzter Variablen

Ablaufsicherheit wird im Gegensatz dazu gewährleistet durch:

– explizite Deklaration von Variablen
– festen Datentyp
– statische Strukturen

Alle diese Punkte berücksichtigt das Variablenkonzept von SDF-P. Sie werden in den folgenden Abschnitten kurz beschrieben.

6.1.1.1 Einfaches Erstellen von Prozeduren

Standardmäßig sind S-Prozeduren so eingestellt, dass dem Programmierer das Arbeiten mit Variablen möglichst erleichtert wird: durch implizite Variablendeklaration, dynamisch wechselnden Datentyp und dynamische Erweiterung zusammengesetzter Variablen.

Implizite Deklaration von Variablen

„Implizite Deklaration“ bedeutet, dass Variablen nicht explizit mit einem Kommando deklariert werden müssen. Sie werden automatisch deklariert, wenn ihnen ein Wert zugewiesen wird.

In S-Prozeduren gilt standardmäßig, dass Variablen implizit deklariert werden können, sie werden dann mit den Standardmerkmalen angelegt: als einfache Variablen, mit dynamisch wechselndem Datentyp (s.u.) und dem Geltungsbereich „aktuelle Prozedur“. Das heißt, auf die Variable kann nur innerhalb der aktuellen Prozedur zugegriffen werden und in den Prozeduren, die in dieser Prozedur mit dem Kommando INCLUDE-PROCEDURE aufgerufen werden (Näheres zum Geltungsbereich siehe Seite 162).

Dynamisch wechselnder Datentyp

Der Datentyp bestimmt, welche Werte eine Variable annehmen kann, zum Beispiel einen Integer-Wert (Datentyp Integer) oder eine beliebige Zeichenfolge (Datentyp String; Zeichenfolgen werden daher im Folgenden auch als Strings bezeichnet).
Standardmäßig werden Variablen in S-Prozeduren mit dynamisch wechselndem Datentyp angelegt.

„Dynamisch wechselnder Datentyp“ bedeutet, dass der Datentyp nicht bei der Deklaration der Variablen festgelegt wird, sondern erst bei der Zuweisung. Er kann dann bei jeder Zuweisung wechseln, das heißt, einer Variablen könnten zum Beispiel nacheinander ein Integer-Wert, ein String und wieder ein Integer-Wert zugewiesen werden.

Der aktuelle Datentyp kann über die vordefinierte Funktion CURRENT-TYPE( ) abgefragt werden (siehe Seite 370).

Dynamische Erweiterung zusammengesetzter Variablen

Zusammengesetzte Variablen sind Variablen, die aus mehreren Variablenelementen bestehen. Zusammengesetzte Variablen müssen explizit deklariert werden. Bei dieser expliziten Deklaration brauchen die Variablenelemente nicht bekannt zu sein, weder die Anzahl der Elemente noch ihre Namen oder ihr Datentyp.

Zusammengesetzte Variablen werden in S-Prozeduren standardmäßig als dynamisch erweiterbar deklariert. „Dynamisch erweiterbar“ bedeutet: Wenn bei der Verarbeitung solcher zusammengesetzter Variablen mehr Elemente benötigt werden, als aktuell zur Verfügung stehen, werden automatisch Elemente hinzugefügt (die zusammengesetzte Variable wird erweitert).

6.1.1.2 Ablaufsicherheit in Prozeduren

Bei umfangreichen, komplexen Prozeduren steht weniger schnelles Erstellen im Vordergrund, sondern eher Ablaufsicherheit, Übersichtlichkeit und auch Wartbarkeit. SDF-P bietet dem Programmierer daher die Möglichkeit, Variablen explizit zu deklarieren, und zwar mit festem Datentyp. Im Falle von zusammengesetzten Variablen vom Typ Struktur bietet SDF-P die Möglichkeit, Variablen explizit zu deklarieren, die eine feste Anzahl von Variablenelementen mit festem Datentyp haben (= statische Strukturen).

Explizite Variablendeklaration

„Explizite Variablendeklaration“ bedeutet, dass Variablen, einfache und zusammengesetzte Variablen, mit den entsprechenden Kommandos deklariert werden (Kommando DECLARE-VARIABLE (siehe Seite 595), DECLARE-ELEMENT (siehe Seite 600), DECLARE-CONSTANT (siehe Seite 614). Bei der expliziten Deklaration können dann auch die Variablenmerkmale eindeutig festgelegt werden.

In S-Prozeduren kann implizite Deklaration von Variablen unterbunden werden: mit dem Kommando SET-PROCEDURE-OPTIONS im Prozedurkopf (siehe Abschnitt „Prozedurkopf erstellen“ auf Seite 81).

Fester Datentyp

Bei der expliziten Variablendeklaration kann der Variablen ein bestimmter Datentyp zugewiesen werden. Dieser Datentyp kann später nicht dynamisch verändert werden.

„Fester Datentyp“ bedeutet, dass der Variablen nur Werte eines bestimmten Datentyps zugewiesen werden können (zum Beispiel nur Integer-Werte). Zuweisungen mit falschem Datentyp werden zurückgewiesen, und es wird eine Fehlerbehandlung angestoßen.

Statische Strukturen

Zusammengesetzte Variablen vom Typ Struktur können als statisch deklariert werden. Das bedeutet, dass die Zahl der Variablenelemente bereits bei der Variablendeklaration festgelegt wird; sie kann später nicht verändert werden.

6.2 Variablendeklaration

Nach der ersten allgemeinen Einführung in das Arbeiten mit Variablen im vorangegangenen Abschnitt wird jetzt im Einzelnen erklärt, wie das Deklarieren von Variablen in SDF-P vonstatten geht. So beschreiben die folgenden Unterabschnitte jeweils einen Aspekt, der dabei zu beachten ist:

- Zunächst werden die unterschiedlichen „Variablentypen“ in SDF-P vorgestellt, das heißt, es werden die Begriffe erläutert, die sich auf S-Variablen als Ganze beziehen.
- Der folgende Abschnitt, „Variablennamen“, beschreibt die Syntax der Variablennamen.
- Die Abschnitte drei und vier beschreiben die möglichen „Datentypen“ der S-Variablen und wie S-Variablen initialisiert werden („Initialisierung“).
- Der Abschnitt „Geltungsbereich“ beschreibt die verschiedenen Geltungsbereiche und ihre Auswirkung auf die „Sichtbarkeit“ von S-Variablen.
- Die letzten beiden Abschnitte beschreiben den Behältermechanismus für S-Variablen („Variablenbehälter“) und die Möglichkeit der Mehrfachdeklaration.

6.2.1 Variablentypen

Variablen in S-Prozeduren sind benannte Datenobjekte, denen ein Wert zugewiesen werden kann. Sie werden über ihren Variablennamen angesprochen. Variablen in S-Prozeduren werden auch als S-Variablen bezeichnet.

Es gibt zwei Typen von S-Variablen:

- einfache Variablen
- zusammengesetzte Variablen

Einfache Variablen können nicht weiter unterteilt werden. Zusammengesetzte Variablen bestehen aus einem oder mehreren Elementen. Die Elemente zusammengesetzter Variablen (= Variablenelemente) können wieder einfache oder zusammengesetzte Variablen sein.

Als Variablenelemente gelten immer nur die Elemente der obersten Ebene einer zusammengesetzten Variablen. Sind die Variablenelemente selbst zusammengesetzte Variablen, werden ihre Elemente nicht als Elemente der übergeordneten zusammengesetzten Variablen betrachtet.

Drei Typen von zusammengesetzten Variablen sind zu unterscheiden:

- Listen (= Listenvariablen)
- Arrays
- Strukturen

Diese zusammengesetzten Variablen unterscheiden sich im Variablennamen, in ihrem inneren Aufbau und in der Art, wie die Variablenelemente angesprochen werden.

In den folgenden Abschnitten werden zunächst einfache Variablen beschrieben, dann allgemein die zusammengesetzten Variablen und schließlich die verschiedenen Typen zusammengesetzter Variablen (Listen, Arrays, Strukturen).

6.2.1.1 Einfache Variablen

Einfache Variablen sind eindeutig benannte Datenobjekte, denen ein Wert zugewiesen und deren Wert verändert werden kann. Sie werden über einen Variablennamen angesprochen. Einfache Variablen enthalten keine Variablenelemente.

Auch Variablenelemente, die nicht selbst zusammengesetzte Variablen sind, werden als einfache Variablen bezeichnet.

Deklaration einfacher Variablen

Einfache Variablen, die nicht Variablenelemente sind, können folgendermaßen deklariert werden:

– implizit bei der ersten Wertzuweisung mit dem Kommando SET-VARIABLE, wenn implizite Deklaration erlaubt ist
– explizit mit dem Kommando DECLARE-VARIABLE ... TYPE = *ANY / *INTEGER / *BOOLEAN / *STRING, MULTIPLE-ELEMENTS = *NO

Wie einfache Variablen, die Variablenelemente sind, deklariert werden, hängt davon ab, ob es sich um Elemente einer Liste, eines Arrays oder einer Struktur handelt:

– Elemente von Listen werden implizit bei der Deklaration der Liste oder später bei dynamischer Erweiterung der Liste angelegt. Sie können über einen Namen angesprochen werden, der sich aus der Position des Listenelements ergibt.
– Elemente von Arrays werden ebenfalls implizit bei der Deklaration des Arrays oder später bei dynamischer Erweiterung des Arrays angelegt. Sie können über einen eigenen Namen angesprochen werden, der bei der Deklaration des Arrays bereits vorgegeben wird (siehe Seite 148).
– Elemente von Strukturen können implizit deklariert werden, wenn dies bei der Deklaration der Struktur festgelegt wird. Sie können aber auch explizit deklariert werden (siehe Seite 149).

6.2.1.2 Zusammengesetzte Variablen

Zusammengesetzte Variablen sind Variablen, die aus mehreren Elementen bestehen. Diese Elemente können über den gemeinsamen Variablennamen, eben den Namen der zusammengesetzten Variablen, angesprochen werden.

Zusammengesetzte Variablen müssen in der Regel explizit mit dem Kommando DECLARE-VARIABLE deklariert werden. Welche Operanden des Kommandos DECLARE-VARIABLE bei der Deklaration verwendet werden müssen, hängt davon ab, ob es sich bei der zusammengesetzten Variablen um eine Liste, ein Array oder eine Struktur handelt.

Arrays und Listen werden über den Operanden MULTIPLE-ELEMENTS = *ARRAY(...) / *LIST(...) deklariert.

Es gibt aber auch Kommandos, wie z.B. SHOW-VARIABLE oder EXECUTE-CMD, die Listen implizit anlegen. Strukturen werden über den Operanden TYPE = *STRUCTURE des Kommandos DECLARE-VARIABLE deklariert (daher werden Strukturen auch als Variablen mit dem Datentyp Struktur bezeichnet). Diese drei Typen zusammengesetzter Variablen werden in den folgenden Abschnitten beschrieben.

Listen

Listen werden auch als Listenvariablen bezeichnet; dieser Begriff wird vor allem dann gewählt, wenn Verwechslungsgefahr mit SDF-Listen besteht (eine SDF-Liste ist ein String, der nach den syntaktischen Regeln für Operandenlisten in Kommandos interpretiert wird).

Listen in SDF-P sind zusammengesetzte Variablen, deren Elemente alle den gleichen Datentyp haben. Auf die Elemente einer Liste kann sequenziell oder direkt zugegriffen werden.

Als Listenelemente sind einfache Variablen zugelassen oder zusammengesetzte Variablen vom Typ Struktur.

Listen haben an der Benutzerschnittstelle nur einen „relativen“ Elementnamen. Dem Listennamen wird dazu das Zeichen # angehängt, gefolgt von der Nummer des Elements. Die Elementnummer ergibt sich aus der Reihenfolge der Elemente in der Liste. Das erste Listenelement, der so genannte Listenkopf, hat immer die Elementnummer 1 (<name>#1). Von diesem Listenkopf ausgehend werden die weiteren Listenelemente durchnummeriert. Beim Ansprechen des Listenkopfs kann die Elementnummer auch weggelassen werden (<name>#).

Listen können in FOR-Schleifen sequenziell abgearbeitet werden: Wenn das aktuelle Listenelement einen gültigen Wert enthält, wird die FOR-Schleife durchlaufen. Dies wird so oft wiederholt, bis die Liste abgearbeitet ist (zu FOR-Schleifen siehe Abschnitt „Schleifen definieren“ auf Seite 97).

Listen deklarieren

Listen werden explizit mit dem Kommando DECLARE-VARIABLE im Operanden MULTIPLE-ELEMENTS = *LIST(...) deklariert. Die hier festgelegten Variableneigenschaften gelten für alle Elemente der Liste.

Listen sind im Kommando SET-VARIABLE implizit dynamisch erweiterbar durch Anfügen eines neuen Elements am Anfang oder am Ende. Jedoch muss berücksichtigt werden, dass die Folge der Listenelemente lückenlos ist.

Soll die Zahl der Listenelemente eingeschränkt werden, muss das über den Operanden LIMIT im Kommando DECLARE-VARIABLE geschehen.

Listen freigeben

Es kann jeweils ein einzelnes Listenelement oder ein zusammenhängernder Bereich von Listenelementen freigegeben werden. Nach der Freigabe wird die Liste neu nummeriert, sodass immer eine lückenlose Nummerierung ab 1 vorhanden ist.

Beispiel

```
/DECLARE-VARIABLE L,MULTIPLE-ELEMENTS=*LIST
/L=*STRING-TO-VAR('(1,2,3,4,5,6,7,8)')
/SHOW-VARIABLE L,LIST-INDEX-NUMBER=*YES
L#1 = 1
L#2 = 2
L#3 = 3
L#4 = 4
L#5 = 5
L#6 = 6
L#7 = 7
L#8 = 8
/FREE-VARIABLE L#3
/SHOW-VARIABLE L,LIST-INDEX-NUMBER=*YES
L#1 = 1
L#2 = 2
L#3 = 4
L#4 = 5
L#5 = 6
L#6 = 7
L#7 = 8
/FREE-VARIABLE *LIST(LIST-NAME=L,FROM-INDEX=4),NUMBER-OF-ELEMENTS=3)
/SHOW-VARIABLE L,LIST-INDEX-NUMBER=*YES
L#1 = 1
L#2 = 2
L#3 = 4
L#4 = 8
```

Listen ausgeben

Einzelne Listenelemente können mit dem Kommando
`/SHOW-VARIABLE <name>#<nummer>` oder in einer FOR-Schleife auszugegeben werden.

Arrays

Arrays sind zusammengesetzte Variablen, deren Elemente alle mit dem gleichen Datentyp deklariert sind.

Arrayelemente können selbst einfache Variablen sein oder zusammengesetzte Variablen vom Typ Struktur. Sie können direkt angesprochen werden über ihren Arrayelementnamen, der sich zusammensetzt aus dem Arraynamen und einem ganzzahligen Arrayindex (siehe Abschnitt „Variablennamen“ auf Seite 155).

Beispiel 1

KONTO ist ein einfaches Array und besteht aus folgenden Elementen:

```
KONTO#-12
KONTO#-1
KONTO#0
KONTO#1
KONTO#2
KONTO#123
KONTO#1234
```

Beispiel 2

KUNDE ist ein zusammengesetztes Array, als Index dient die Kundennummer, die Arrayelemente sind Strukturen, die jeweils die Adresse des Kunden enthalten.

```
KUNDE#123.NAME
KUNDE#123.VORNAME
KUNDE#123.STRASSE
...
KUNDE#358.NAME
KUNDE#358.VORNAME
KUNDE#358.STRASSE
...
```

Arrays deklarieren

Arrays werden deklariert mit dem Kommando DECLARE-VARIABLE, Operand MULTIPLE-ELEMENTS = *ARRAY(...). Die übrigen hier deklarierten Variableneigenschaften gelten für alle Elemente des Arrays.

Arrayelemente können nur implizit deklariert werden. Explizite Deklaration ist für einzelne Elemente nicht möglich.

Arrays sind immer dynamisch erweiterbar. Über die Operanden UPPER-BOUND und LOWER-BOUND kann ein Wertebereich für den Arrayindex definiert und so die Zahl der Arrayelemente eingeschränkt werden.

Strukturen

Strukturen sind zusammengesetzte Variablen, deren Elemente unterschiedliche Datentypen haben können und direkt über alphanumerische Elementnamen angesprochen werden können.

Als Strukturelemente sind einfache Variablen oder zusammengesetzte Variablen vom Typ Liste, Array oder Struktur zugelassen.

Der Strukturelementname setzt sich zusammen aus dem Variablennamen der Struktur und einem Teilnamen vom SDF-Datentyp <structured-name>, die durch einen Punkt voneinander getrennt werden (siehe Abschnitt „Variablennamen“ auf Seite 155).

Strukturen können statisch oder dynamisch erweiterbar sein. Dies wird bei der Deklaration festgelegt.

Hinweis

Strukturen spielen eine wichtige Rolle bei strukturierten Ausgaben mittels strukturierter Variablenströme. Detail-Informationen hierzu enthält das Kapitel „S-Variablenströme“ auf Seite 193ff.

Beispiel

ANSCHRIFT ist eine Struktur, die aus den folgenden einfachen Variablen als Elementen besteht:

```
ANSCHRIFT.NAME
ANSCHRIFT.VORNAME
ANSCHRIFT.TITEL
ANSCHRIFT.STRASSE
ANSCHRIFT.HAUSNR
ANSCHRIFT.PLZ
ANSCHRIFT.ORT
```

Die Strukturelemente NAME, VORNAME, TITEL, STRASSE und ORT können mit dem Datentyp STRING deklariert werden, die Strukturelemente HAUSNR und PLZ mit dem Datentyp INTEGER.

Strukturen deklarieren

Zusammengesetzte Variablen vom Typ Struktur werden mit dem Kommando DECLARE-VARIABLE deklariert, und zwar im Operanden NAME mit TYPE = *STRUCTURE (DEFINITION=). Dort wird auch festgelegt, ob die Struktur dynamisch erweiterbar oder statisch ist.

Dynamische Strukturen

Eine Struktur ist „dynamisch“ oder „dynamisch erweiterbar“, wenn die Elemente nicht explizit über ein Strukturlayout oder mit *BY-SYSCMD deklariert werden.

Dynamische Strukturen werden explizit deklariert mit dem Kommando DECLARE-VARIABLE, Operand NAME = ...(TYPE = *STRUCTURE(DEFINITION = *DYNAMIC)).

Die Elemente dynamischer Strukturen können explizit oder implizit deklariert werden. Die Einstellung des Operanden IMPLICIT-DECLARATION im Kommando SET-PROCEDURE-OPTIONS im Prozedurkopf hat keinen Einfluss; implizite Deklaration von Elementen dynamischer Strukturen ist immer möglich.

Beispiel 1

Es wird eine dynamische Struktur DYN-STR deklariert:

```
/DECLARE-VARIABLE DYN-STR (TYPE = *STRUCTURE(*DYNAMIC))
```

Später wird über eine Zuweisung ein Element dieser Struktur initialisiert:

```
/DYN-STR.STR2.ARR#123 = 'ABC'
```

Diese Zuweisung bewirkt Folgendes:

- Das Strukturelement DYN-STR.STR2 wird als dynamische Struktur angelegt.
- Das Strukturelement DYN-STR.STR2.ARR wird als Array angelegt, mit dem Datentyp TYPE = *ANY.
- Das Arrayelement DYN-STR.STR2.ARR#123 wird als einfache Variable angelegt, mit dem Datentyp TYPE = *ANY.

Beispiel 2

```
/DECLARE-VARIABLE S1 (TYPE = *STRUCTURE(*DYNAMIC))
```

Über eine Zuweisung wird ein Element dieser Struktur initialisiert:

```
/S1.S2.ARR#1.S3 = 'ABC'
```

Diese Zuweisung bewirkt Folgendes:

- Das Strukturelement S1.S2 wird als dynamische Struktur angelegt.
- Das Strukturelement S1.S2.ARR wird als Array angelegt, mit dem Datentyp TYPE = *STRUCTURE(*DYNAMIC).
- Das Arrayelement S1.S2.ARR#1 wird als dynamische Struktur angelegt, mit dem Datentyp TYPE = *STRUCTURE(*DYNAMIC).
- Das Strukturelement S1.S2.ARR#1.S3 wird als einfache Variable angelegt, mit dem Datentyp TYPE = *ANY.

Beispiel 3

```
/DECLARE-VARIABLE DYN-STR (TYPE = *STRUCTURE(*DYNAMIC))
/DECLARE-ELEMENT DYN-STR.S.NUMBER(TYPE=*INTEGER)
```

In diesem Beispiel sind die Elemente explizit deklariert, und zwar DYN-STR.S als dynamische Struktur und DYN-STR.S.NUMBER als einfache Variable mit TYPE = *INTEGER.

Beispiel 4

```
/DECLARE-VARIABLE S1 (TYPE = *STRUCTURE(*DYNAMIC))

/S1#123 = 'ABC' → Fehler
```

Die Zuweisung wird mit Fehler abgewiesen, da S1 nicht als Array deklariert ist.

Statische Strukturen

Statische Strukturen sind Strukturen, die nicht dynamisch erweiterbar sind. Zu unterscheiden ist hier, ob die statische Struktur durch *BY-SYSCMD erzeugt oder über ein so genanntes Strukturlayout definiert wird. Im folgenden Abschnitt wird zunächst die Deklaration über das Strukturlayout vorgestellt, im Anschluss daran die Deklaration von Strukturen, die für die Ausgabe von BS2000-Kommandos verwendet werden.

Wenn die Strukturdeklaration abgeschlossen ist, können der Struktur keine weiteren Elemente mehr hinzugefügt werden.

Regeln für statische Strukturen

Strukturdeklarationen müssen innerhalb einer Prozedur vollständig sein. Das heißt, sie müssen in der Prozedur, in der sie beginnen, abgeschlossen werden. Dies gilt sowohl für CALL- als auch für INCLUDE-Prozeduren.

Deklarationsblöcke dürfen Kontrollflusskommandos oder Prozeduraufrufe enthalten.

Bei einem Prozeduraufruf innerhalb einer Strukturdeklaration ist Folgendes zu beachten:

– Innerhalb eines Deklarationsblocks können Prozeduren beliebig aufgerufen werden.
– Wird eine Prozedur mit INCLUDE-PROCEDURE aufgerufen, sind die Elemente, die vor dem Aufruf deklariert wurden, nicht sichtbar. Auch können in der INCLUDE-Prozedur keine weiteren Elemente der unterbrochenen Struktur deklariert werden.
– Wenn in einer INCLUDE-Prozedur auf eine unvollständige Struktur oder ein unvollständiges Strukturlayout zugegriffen wird, kommt es zum Fehler.

Abbruch der Strukturdeklaration durch Prozedurende:

– Wenn eine Prozedur während einer Strukturdeklaration abgebrochen wird, werden unvollständige Strukturen mit dem Geltungsbereich Prozedur oder Include automatisch gelöscht. Eine entsprechende Warnung wird ausgegeben.
– Unvollständige Strukturen und Strukturlayouts mit dem Geltungsbereich Task bleiben erhalten. Die Strukturdeklaration wird implizit abgeschlossen, sodass später auf die Struktur bzw. die zuvor deklarierten Strukturelemente zugegriffen werden kann. Es können jedoch keine weiteren Elemente deklariert werden.

Strukturen mit *BY-SYSCMD deklarieren

Wird eine statische Struktur mit DEFINITION = *BY-SYSCMD deklariert, so muss dem Deklarationskommando für die Struktur direkt die Deklaration der Strukturelemente folgen.

Die Strukturelemente müssen in einem Strukturdeklarationsblock deklariert werden. Dieser Block wird mit dem Kommando BEGIN-STRUCTURE eingeleitet und mit END-STRUCTURE abgeschlossen. Im Kommando BEGIN-STRUCTURE darf kein Strukturlayoutname angegeben werden.

Es ergeben sich damit für die Deklaration folgende Schritte:

1. Struktur explizit deklarieren

   Das Kommando DECLARE-VARIABLE muss folgende Angaben enthalten:

   NAME = variablenname
   TYPE = *STRUCTURE(DEFINITION = *BY-SYSCMD)

2. Strukturdeklarationsblock einleiten

   Dem Kommando DECLARE-VARIABLE muss direkt das Kommando BEGIN-STRUCTURE folgen. Das Kommando darf weder einen Strukturlayoutnamen (NAME =) noch einen Geltungsbereich (SCOPE =) enthalten.

3. Strukturelemente deklarieren

Die Strukturelemente müssen einzeln über separate Aufrufe des Kommandos DECLARE-ELEMENT deklariert werden. Die Strukturelemente können als einfache oder zusammengesetzte Variablen deklariert werden, mit beliebigem Datentyp. Sind die Strukturelemente einfache Variablen, können sie mit INITIAL-VALUE initialisiert werden. Alle Eigenschaften, die nicht explizit im Kommando DECLARE-ELEMENT deklariert werden, werden von der übergeordneten Struktur übernommen. Im Operanden NAME wird nur der Elementname angegeben, nicht der Strukturname.

4. Strukturdeklarationsblock abschließen

   Mit dem Kommando END-STRUCTURE wird der Deklarationsblock abgeschlossen.

Beispiel 1

Deklaration einer Struktur mit einfachen Variablen als Strukturelementen:

```
/DECLARE-VARIABLE M(TYPE = *STRUCTURE(*BY-SYSCMD))
/BEGIN-STRUCTURE
/   DECLARE-ELEMENT A
/   DECLARE-ELEMENT B
/END-STRUCTURE
```

Beispiel 2

Deklaration einer Struktur mit einer zusammengesetzten Variablen als Strukturelement:

```
/DECLARE-VARIABLE M(TYPE = *STRUCTURE(*BY-SYSCMD))
/BEGIN-STRUCTURE
/   DECLARE-ELEMENT A(TYPE = *STRUCTURE(*BY-SYSCMD))
/   BEGIN-STRUCTURE
/      DECLARE-ELEMENT B
/   END-STRUCTURE
/END-STRUCTURE
```

Es existiert jetzt die statische Struktur M mit dem Element M.A.B. Da A wiederum als Struktur mit *BY-SYSCMD deklariert wurde, musste BEGIN-STRUCTURE geschachtelt werden.

Regeln für Strukturen mit *BY-SYSCMD

Die Kommandos DECLARE-VARIABLE und BEGIN-STRUCTURE müssen in dieser Reihenfolge direkt aufeinander folgen.

BEGIN-STRUCTURE darf weder einen Layoutnamen noch einen Geltungsbereich enthalten.

Auf die Strukturelemente kann erst dann zugegriffen werden, wenn die Deklaration aller Elemente abgeschlossen ist. (Ausnahme: mit dem Kommando SHOW-VARIABLE kann der Inhalt der Strukturelemente jederzeit angezeigt werden.)

Strukturen mit benanntem Strukturlayout deklarieren

Das Strukturlayout entspricht einer Schablone, die es erlaubt, gleichartige Strukturen einfach zu erzeugen. Existiert ein Strukturlayout, dann kann es in beliebig vielen DECLARE-VARIABLE Kommandos verwendet werden, die so beliebig viele Strukturen mit den gleichen Elementen erzeugen.

Als Folge von SDF-P-Kommandos beginnt ein Strukturlayout mit einem Kommando BEGIN-STRUCTURE, gefolgt von DECLARE-ELEMENT-Kommandos, und abgeschlossen wird es mit dem Kommando END-STRUCTURE.

In diesen Strukturlayouts werden die einzelnen Elemente von Strukturen explizit deklariert. Strukturlayouts sind eindeutig durch einen Namen gekennzeichnet. Dieser Name muss bei der Deklaration der Struktur im Kommando DECLARE-VARIABLE wiederholt werden.

Es ergeben sich damit im Einzelnen folgende Schritte:

1. Strukturlayout (= Deklarationsblock für die Strukturelemente) einleiten

   Mit dem Kommando BEGIN-STRUCTURE wird das Strukturlayout eingeleitet.
   Im Operanden NAME muss ein Name angegeben werden (NAME = layoutname). Über diesen Namen wird später eine Verbindung hergestellt zwischen der Deklaration der Variablen „variablenname“ vom Typ Struktur im Kommando DECLARE-VARIABLE und der Deklaration der Elemente dieses Strukturlayouts.
   Wenn das Kommando DECLARE-VARIABLE aufgerufen wird, muss das Strukturlayout existieren.

2. Strukturelemente deklarieren

   Alle Layoutelemente werden über separate Aufrufe des Kommandos DECLARE-ELEMENT deklariert.
   Sie können als einfache Variablen oder als zusammengesetzte Variablen vom Typ Liste, Array oder Struktur deklariert werden.
   Alle Variablenmerkmale, die im DECLARE-ELEMENT-Kommando nicht explizit deklariert werden, werden von der übergeordneten Struktur übernommen.
   Layoutelemente dürfen nicht initialisiert werden.

3. Strukturlayout/Deklarationsblock abschließen

   Mit dem Kommando END-STRUCTURE wird das Strukturlayout abgeschlossen.

4. Struktur explizit deklarieren

   Im Kommando DECLARE-VARIABLE, im Operanden NAME = variablenname (TYPE = *STRUCTURE(...)) wird in der Klammer ein Layoutname angegeben (DEFINITION = layoutname) und so die Verbindung zu dem bereits deklarierten Strukturlayout hergestellt.

Beispiel

Zunächst wird ein Strukturlayout A deklariert:

```
/BEGIN-STRUCTURE A
/   DECLARE-ELEMENT B
/   DECLARE-ELEMENT C
/END-STRUCTURE
```

Es existiert noch keine Struktur, Wertzuweisungen sind nicht möglich. Später werden Strukturen deklariert, für die das Strukturlayout A gelten soll:

```
/DECLARE-VARIABLE KK (TYPE = *STRUCTURE (DEFINITION = A))
/...
/DECLARE-VARIABLE CC (TYPE = *STRUCTURE (DEFINITION = A))
```

Jetzt existieren zwei statische Strukturen, KK und CC, mit den Elementen KK.B und KK.C bzw. CC.B und CC.C. Diesen Elementen können jederzeit Werte zugewiesen werden.

Regeln für Strukturlayouts

Bei der Deklaration muss im einleitenden Kommando BEGIN-STRUCTURE ein Name angegeben werden.

Strukturlayouts müssen vor den Strukturen deklariert werden.

Jedes Strukturlayout steht für beliebig viele Deklarationen von statischen Strukturen zur Verfügung.

Strukturlayouts haben einen eigenen Namensraum, das heißt, Variablen und Strukturlayouts dürfen gleiche Namen haben.

Einem Strukturlayout dürfen keine Werte zugewiesen werden. Es dient nur als Schablone für spätere Deklarationen.

Strukturlayouts dürfen - im Gegensatz zu anderen Strukturdeklarationen - nicht geschachtelt werden. Das heißt, jeder Deklarationsblock muss mit END-STRUCTURE abgeschlossen werden, bevor der nächste Deklarationsblock mit BEGIN-STRUCTURE geöffnet werden kann.

Hinweis

Beispiele für die Deklaration von Strukturen, Strukturlayouts und Strukturelementen enthält auch die Beschreibung des Kommandos DECLARE-ELEMENT auf Seite 600.

6.2.2 Variablennamen

Dieser Abschnitt beschreibt zunächst die Syntaxregeln für Variablennamen in S-Prozeduren. Anschließend sind die in SDF-P reservierten Wörter und Variablennamen aufgelistet, die der Benutzer nicht für eigene Variablennamen verwenden darf.

6.2.2.1 Syntax der Variablennamen

Es gilt folgende Metasyntax:

Symbol	Bedeutung
:=	Definition
< >	SDF-Datentyp
m..n	Wertebereich
...	Wiederholung
/	Alternativangaben
[ ]	optionale Angabe

Für Variablennamen und Variablenteilnamen in SDF-P werden in der Kommandosyntax die SDF-Datentypen <composed-name> und <structured-name> verwendet. Die tatsächlich von SDF-P überprüfte Syntax ist aber gegenüber diesen Datentypen eingeschränkt und soll deshalb hier näher beschrieben werden.

Ein Variablenname (composed-variable-name) in SDF-P setzt sich zusammen aus einem oder mehreren Strukturnamen (structured-variable-name), die durch Punkte voneinander getrennt sind. Durch den Punkt wird jeweils der Name einer Unterstruktur gekennzeichnet. Leerzeichen innerhalb eines Variablennamens sind nicht zugelassen.

```
composed-variable-name:=structured-variable-name[.structured-variable-name]
```

Länge mindestens 1 Zeichen, höchstens 255 Zeichen

Beispiel

```
ABCDEFGHIJKLMNO#1
A#1
A123#1
BCD.XYZ
```

Ein Strukturname besteht aus einem Variablenteilnamen (vtn), der von einem # und einem Indexnamen gefolgt sein kann, falls er ein Listenelement oder ein Arrayelement bezeichnet.

```
structured-variable-name := vtn [# [indexname]]
```

Länge	mindestens 1 Zeichen, höchstens 20 Zeichen
Variablenteilname	(vtn):
Zeichenvorrat	alle Buchstaben (A, ... Z) alle Ziffern (0, ... 9) @ und Bindestrich (-)
Erstes Zeichen	Buchstabe
Konventionen	Zeichenfolge SYS am Beginn des Variablennamens ist reserviert für Systemvariablen und sollte vom Benutzer nicht verwendet werden. Es dürfen nicht zwei Bindestriche hintereinander stehen. Der Bindestrich darf nicht das letzte Zeichen des Variablenteilnamens sein.

Beispiel

```
TELEFON-VORWAHL#1
KUNDE#NUMMER
```

Falls der Strukturname ein Listenelement oder ein Arrayelement bezeichnet, so folgt ihm ein # mit einem Indexnamen.

```
indexname = <integer -2147483648..2147483647>  / vtn
```

Beispiel

```
TELEFON#223366
TELEFON#INDEX
```

Elementnamen

Bei Elementnamen muss unterschieden werden zwischen Arrayelementen, Listenelementen und Strukturelementen.
Generell gilt für Elementnamen die Syntax, wie unter <composed-variable-name> beschrieben.

Listenelementnamen

Listenelementnamen bestehen an der Benutzerschnittstelle aus folgenden Komponenten:

– Name der Liste (<composed-variable-name 1..253>)
– #
– optional: Elementnummer (<integer 1..2147483647>)

Arrayelementnamen

Arrayelementnamen setzen sich zusammen aus folgenden Komponenten:

– Name des Arrays (<composed-variable-name 1..253>)
– # (Kennzeichen für Arrayelemente, wenn weitere Zeichen folgen)
– Arrayindex (<integer -2147483648..2147483647>)

Der Listenindex sowie der Arrayindex können auch über eine einfache Variable bezeichnet werden, die einen Integer-Wert im gültigen Wertebereich enthält.

SDF-P analysiert die Zeichenfolge, die im Elementnamen auf das Aggregatsymbol # folgt: Ist das erste Zeichen eine Ziffer oder ein Bindestrich (= Minuszeichen), wird die Zeichenfolge als Integer-Wert interpretiert, also als direkte Angabe des Index.

Ist das erste Zeichen ein Buchstabe, wird die Zeichenfolge als Variablenname interpretiert. SDF-P sucht dann eine Variable mit dem angegeben Namen. Dieser Name muss eine einfache Variable bezeichnen, die mit einem gültigen Integer-Wert initialisiert sein muss. Ist das nicht der Fall, wird eine Fehlermeldung ausgegeben. Wenn die Variable einen gültigen Integer-Wert enthält, wird dieser Wert im Arrayindex bzw. im Listenindex eingesetzt und auf das so bezeichnete Aggregatelement zugegriffen.

Beispiel 1

KONTO ist ein einfaches Array. Der Arrayindex kann im Arrayelementnamen direkt angegeben werden:

```
KONTO#123
```

Der Arrayindex kann aber auch über eine Variable, INDEX, geliefert werden:

```
/INDEX = 236
/SHOW-VAR KONTO#INDEX
```

Die Zeichenfolge INDEX wird als Variablenname erkannt und der Inhalt der Variablen ausgewertet. Dementsprechend gibt das Kommando SHOW-VARIABLE den Inhalt des Arrayelements KONTO#236 aus. Allerdings muss das Arrayelement KONTO#236 schon vorher deklariert und initialisiert sein.

Beispiel 2

Die Angabe des Arrayindex wird immer als Integer-Wert ausgewertet. Daher bezeichnen die folgenden Angaben das gleiche Element des Arrays KONTO:

```
KONTO#123
KONTO#0123
KONTO#00123
KONTO#NUMMER
```

Voraussetzung für die Angabe KONTO#NUMMER ist, dass eine Variable NUMMER deklariert und mit dem Wert 123 initialisiert ist.

Strukturelementnamen

Strukturelementnamen setzen sich aus folgenden Komponenten zusammen:

– Name der Struktur (<composed-variable-name 1..253>)
– „.“ (als Kennzeichen für Strukturen)
– Teilname des Elements (<structured-variable-name 1..20>)

Beispiel

Folgende Variablen sind Elemente der Struktur ANSCHRIFT:

```
ANSCHRIFT.NAME
ANSCHRIFT.VORNAME
ANSCHRIFT.TITEL
ANSCHRIFT.STRASSE
```

6.2.2.2 Reservierte Wörter

„Reservierte Wörter“ sind Schlüsselwörter, die für Operatoren in Ausdrücken oder boolesche Konstanten verwendet werden. Sie dürfen nicht als Variablennamen deklariert werden.

Wenn sie dennoch in einer Variablen-Deklaration eingesetzt werden, wird keine Variable angelegt, es wird eine Fehlermeldung ausgegeben.

Folgende Schlüsselwörter sind reservierte Wörter:

AND	LE	OFF
DIV	LT	ON
EQ	MOD	OR
FALSE	NE	TRUE
GE	NO	YES
GT	NOT	

6.2.2.3 Reservierte Variablennamen

Variablennamen, die mit dem Präfix SYS beginnen sind reserviert für den Datentransfer von und zu Systemkomponenten. Dabei bildet der Variablenname SYSSWITCH eine besondere Form.

SYSSWITCH

Der Variablenname SYSSWITCH bezeichnet eine zusammengesetzte Variable vom Typ Array, über die die Auftragsschalter angesprochen werden können.

Das Array SYSSWITCH ist folgendermaßen definiert:

Datentyp	BOOLEAN
Geltungsbereich	TASK (in Dialog- und Prozedurumgebung)
Anzahl Elemente	32
Arrayindex	0,..., 31
Werte	TRUE = Schalter steht auf ON, FALSE = Schalter steht auf OFF

Beispiel

Auftragsschalter setzen:

```
/MODIFY-JOB-SWITCHES ON = 1
/SET-VARIABLE SYSSWITCH#1 = TRUE
```

Die beiden Kommandos haben die gleiche Wirkung.

Beispiel

```
/DECLARE-VARIABLE SAVED-SYSSWITCH,MULTIPLE-ELEMENTS=*ARRAY

/SET-VARIABLE SAVED-SYSSWITCH=SYSSWITCH
/MODIFY-JOB-SWITCHES ON =( 1,4,5)
/...
/SET-VARIABLE SYSSWITCH=SAVED-SYSSWITCH
```

Die Arrayelemente können über die Teilnamen SYSSWITCH#0 bis SYSSWITCH#31 angesprochen werden. Es ist sowohl Schreib- als auch Lesemodus zulässig.

Das Array SYSSWITCH oder die Arrayelemente können nicht gelöscht werden, das heißt, sie können nicht in einem FREE-VARIABLE- bzw. DELETE-VARIABLE-Kommando angegeben werden.

Das Array SYSSWITCH ist immer implizit in jeder Prozedur deklariert.

SYSPARAM

Der Variablenname SYSPARAM bezeichnet eine Variable vom Typ String, über die auf die mit den Kommandos START-/LOAD-EXECUTABLE-PROGRAM übergebenen Programmparameter zugegriffen werden kann. In C-Programmen erfolgt der Zugriff auf die Programmparameter mit der Funktion `getopt`; Assemblerprogramme müssen die Variable SYSPARAM über den Makroaufruf GETVAR (siehe Seite 333) einlesen und selbst auswerten.

Beispiel

```
/LOAD-EXE FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=ASS.PLAMLIB,
                                     ELEMENT-OR-SYMBOL=NEUWORT4),
          PROGRAM-PARAMETERS='INPUT=DATEI-1,OUTPUT=OUT.ERGEBNIS'
%  BLS0517 MODULE 'NEUWORT4' LOADED
/SHOW-VARIABLE SYSPARAM,INF=*PAR(VALUE=*C-LIT)
SYSPARAM = 'INPUT=DATEI-1,OUTPUT=OUT.ERGEBNIS'
```

6.2.3 Datentyp

SDF-P unterscheidet drei Datentypen: INTEGER, BOOLEAN und STRING.

INTEGER ist ein numerischer Datentyp und bezeichnet ganze Zahlen im Wertebereich von -2^{31} bis $2^{31}-1$.

BOOLEAN ist ein boolescher Datentyp. Der Wertebereich umfasst die Werte 0 und 1. Diese Werte können über die reservierten Wörter TRUE, ON, YES (für 1) und FALSE, OFF und NO (für 0) angesprochen werden.

STRING ist ein alphanumerischer Datentyp und bezeichnet Zeichenketten, die bis zu 4096 Zeichen lang sein dürfen.

Der Datentyp wird im Kommando DECLARE-VARIABLE über den Operanden TYPE deklariert. Außer den drei oben vorgestellten Bezeichnungen für die Datentypen können dem Operanden die Werte ANY und STRUCTURE zugewiesen werden.

ANY ist der Defaultwert; der Datentyp der Variablen wird bei der Zuweisung festgelegt, kann sich also mit jeder erneuten Zuweisung ändern (dynamischer Datentyp).

STRUCTURE legt fest, dass eine Variable vom Typ Struktur angelegt wird.

In der SDF-Struktur, die durch STRUCTURE eingeleitet wird, wird festgelegt, wie das Variablenaggregat „Struktur“ aufgebaut werden soll, ob es dynamisch (*DYNAMIC) oder statisch ist und ob der Strukturdeklaration die Elementdeklarationen folgen (*BY-SYSCMD) oder ob sie über ein Strukturlayout definiert werden.

6.2.4 Anfangswert

Einfache Variablen können bereits bei der Deklaration initialisiert werden, indem ihnen mit dem Operanden INITIAL-VALUE ein Anfangswert zugewiesen wird.

Wird die Variable mit einem festen Datentyp deklariert, das heißt mit INTEGER, BOOLEAN oder STRING, muss der Anfangswert den gleichen Datentyp haben. Andernfalls kommt es zum Fehler.

Wird für den Datentyp ANY angegeben, bestimmt der Wert, der bei INITIAL-VALUE angegeben wird, den aktuellen Datentyp der Variablen.

Standardmäßig erhalten Variablen keinen Anfangswert; der Benutzer muss ihn explizit zuweisen.

Wenn der Benutzer mit INITIAL-VALUE einen Anfangswert explizit zuweist, muss unterschieden werden, ob eine Variable erstmals deklariert wird oder ob eine bereits bestehende Variable neu deklariert wird. Nur Variablen, die erstmals deklariert werden, können initialisiert werden. Wird eine bereits bestehende Variable neu deklariert, wird dieser Wert nicht überschrieben, die Angabe des neuen Anfangswertes wird ignoriert.

Auf Variablen, die nicht initialisiert sind, also keinen gültigen Wert enthalten, kann nur „schreibend“ zugegriffen werden. Schreibzugriff auf Variablen heißt, dem Variablennamen wird explizit oder implizit mit dem Kommando SET-VARIABLE oder READ-VARIABLE ein Wert zugewiesen.

Lesezugriff auf nichtinitialisierte Variablen führt zum Fehler. „Lesezugriff auf Variablen“ bedeutet, dass der Inhalt der Variablen ausgewertet wird, zum Beispiel über eine Funktion, in einem Ausdruck usw.

Konstanter (Anfangs-)Wert

Das Kommando DECLARE-CONSTANT deklariert eine schreibgeschützte Variable mit konstantem Anfangswert.

Bei der Deklaration der Variablen muss im Unteroperanden VALUE ein Wert angegeben werden, der durch nachfolgende Zuweisungen nicht mehr überschrieben werden kann. Das bedeutet: Der Wert kann nicht mit SET-VARIABLE geändert und auch nicht mit FREE-VARIABLE gelöscht werden.

6.2.5 Geltungsbereich von Variablen

Der Geltungsbereich einer Variablen legt fest, wie lange und in welchem Kontext eine Variablendefinition gültig ist. Er bestimmt damit die Lebensdauer und die Sichtbarkeit der Variablen.

Die „Lebensdauer“ der Variablen kann entweder an den Ablauf einer Prozedur oder an die laufende Task gebunden werden. Im ersten Fall wird mit dem Ende der betreffenden Prozedur die Variablendefinition ungültig; sie steht auch dann nicht mehr zur Verfügung, wenn dieselbe Prozedur zu einem späteren Zeitpunkt erneut aufgerufen wird.

Durch Wahl des Geltungsbereichs CURRENT wird die Lebensdauer durch die aktuelle Prozedur begrenzt, in der die Deklaration geschieht; der Geltungsbereich PROCEDURE bindet die Lebensdauer der Variablen an die letzte in der aktuellen Aufrufverschachtelung mit CALL-PROCEDURE gerufene Prozedur.

Variablen mit dem Geltungsbereich TASK bleiben hingegen bis zum Taskende erhalten und überdauern somit alle Prozeduraufrufe in dieser Task.

Die „Sichtbarkeit“ einer Variablen legt fest, ob aus einer Prozedur heraus auf die Variable zugegriffen werden kann. „Sichtbar“ sind alle Variablen, die in der aktuellen Prozedur – gleich mit welchem Geltungsbereich – deklariert worden sind. Darüber hinaus ist eine Variable nur sichtbar, wenn

– die aktuelle Prozedur mit dem Kommando INCLUDE-PROCEDURE aufgerufen wurde,
– die Variable in der rufenden Prozedur sichtbar war und
– in der aktuellen Prozedur keine andere Variable gleichen Namens deklariert wurde.

Dies hat zur Folge, dass Variablen über (auch mehrere, geschachtelte) Include-Aufrufe hinweg von der gerufenen Prozedur „gesehen“ und damit auch benutzt werden können. Eine mit CALL-PROCEDURE gerufene Prozedur hingegen „sieht“ keine Variable, die sie nicht selbst deklariert hat. (Eine Ausnahme bildet die SYSSWITCH-Variable, siehe Seite 159.)

6.2.5.1 Geltungsbereich TASK

Taskvariablen können aus einer Prozedur heraus nur angesprochen werden, wenn dies explizit vorgesehen wird. Es gibt drei verschiedene Möglichkeiten hierzu:

a) Deklaration einer Taskvariablen

Mit einem Kommando /DECLARE-VARIABLE ...,SCOPE = *TASK kann eine Taskvariable erzeugt werden; sie ist anschließend unter dem angegebenen Namen in der deklarierenden Prozedur ansprechbar.

Wenn eine Taskvariable gleichen Namens bereits existierte, wird sie durch das DECLARE-VARIABLE-Kommando ebenfalls zugänglich gemacht; die weiteren Operanden des Kommandos dürfen allerdings nicht den Attributen der bestehenden Variablen widersprechen.

b) Import einer Taskvariablen

Soll von einer Prozedur aus auf eine Taskvariable zugegriffen werden, von der bekannt ist, dass sie bereits existiert, so genügt es, ein IMPORT-VARIABLE-Kommando für diese Variable in die Prozedur aufzunehmen.

Das Importieren einer Variablen ist gleichwertig zur Deklaration mittels eines DECLARE-VARIABLE-Kommandos mit der Angabe SCOPE = *TASK, jedoch kann dabei keine neue Variable erzeugt werden. Dafür erspart es eine erneute, widerspruchsfreie Angabe der Attribute TYPE und MULTIPLE-ELEMENTS (mit ggf. daranhängenden Unterstrukturen).

Beispiel

Eine Taskvariable ARBEITSBIBLIOTHEK soll nur dann neu angelegt und initialisiert werden, wenn sie noch nicht vorhanden ist. Andernfalls soll die aktuelle Prozedur die bestehende Definition verwenden:

```
/IF IS-DECLARED('ARBEITSBIBLIOTHEK',SCOPE=*TASK)
/ IMPORT-VARIABLE ARBEITSBIBLIOTHEK "Die Variable kann jeden beliebigen
                                    Datentyp haben"
/ELSE
/ DECLARE-VARIABLE ARBEITSBIBLIOTHEK(INITIAL-VALUE='#WORKLIB',-
/ TYPE=*STRING)
/END-IF
```

c) Nutzung einer Taskvariable als Behälter

Die beiden vorgenannten Möglichkeiten erlauben die Verwendung einer Taskvariablen innerhalb einer Prozedur nur unter ihrem Originalnamen. Wenn dies nicht erwünscht ist, weil z.B. eine prozedurlokale Variable gleichen Namens vorhanden ist oder auch der Name der Taskvariablen (wegen der notwendigen taskweiten Eindeutigkeit) zu lang und zu umständlich ist, kann eine Taskvariable als Behälter (Container) für eine in der Prozedur eingerichtete Variable benutzt werden.

Beispiel

```
/DECLARE-VARIABLE I(INIT-VALUE=0,TYPE=*INTEGER), -
/ CONTAINER=*VAR(ZAEHLER-FUER-PROZ-A,SCOPE=*TASK)
```

Nach dieser Deklaration kann mit Kommandos wie

```
/I = I + 1
```

auf die Taskvariable zugegriffen werden, deren tatsächlicher Name ZAEHLER-FUER-PROZ-A Konflikte mit den Namen anderer Taskvariablen vermeidet.

Die als Behälter benutzte Taskvariable muss schon vorher vorhanden sein. Darum dürfen auch keine Attribute für die Variable, die auf den Variablenbehälter Bezug nimmt, vereinbart werden, die der Deklaration des Variablenbehälters widersprechen (TYPE, DEFINITION, MULTIPLE-ELEMENTS).

Prozedur-Unterbrechung

Bei Prozedur-Unterbrechung sind nur die Variablen sichtbar, die in der gerade unterbrochenen Prozedur sichtbar sind.

6.2.5.2 Geltungsbereiche PROCEDURE und CURRENT

Neben den Taskvariablen gibt es „prozedurlokale“ Variablen mit den Geltungsbereichen PROCEDURE und CURRENT. Diese sind für eine Prozedur, die mit CALL-PROCEDURE aufgerufen wurde, identisch. Eine „Includeprozedur“ hingegen, die auf die Variablen der rufenden Prozedur zugreifen kann, ohne sie erneut zu deklarieren oder zu importieren, kann wählen, ob eine Deklaration im Geltungsbereich des Aufrufers (SCOPE = *PROCEDURE) wirksam werden soll oder nur „privat“, für die Includeprozedur selbst gelten soll (SCOPE= *CURRENT).
Wenn eine Variable implizit deklariert wird, erhält sie immer den Geltungsbereich *CURRENT.

Import einer prozedurlokalen Variablen

Eine prozedurlokale Variable kann von einer gerufenen Prozedur importiert werden, wenn sie in der rufenden Prozedur mit dem Kommando

```
DECLARE-VARIABLE VAR-NAME=...,SCOPE=*CURRENT/*PROCEDURE(IMPORT-ALLOWED=*YES)
```

deklariert wurde und von der gerufenen Prozedur mit dem Kommando

```
IMPORT-VARIABLE NAME= ..., FROM=*SCOPE(SCOPE=*CALLING-PROCEDURES)
```

importiert wird.

Beispiel

Includeprozeduren, die mit den Variablen des Aufrufers arbeiten, können sehr einfach geschrieben werden: Die Deklaration von Parametern und lokalen Variablen ist oft nicht erforderlich. Die nachfolgende Beispielprozedur ruft mehrmals eine Includeprozedur auf, um zu verschiedenen Geldbeträgen die Mehrwertsteuer zu berechnen und nach SYSOUT auszugeben.

Rufende Prozedur:

```
/"Berechnung der Mehrwertsteuer"
/
/   "Vorbesetzungen"
/ DEZIMAL-ZEICHEN = ','; MWST = 16 "%"
/
/   "1. Berechnung"
/ BETRAG = 5730
/ INCLUDE-PROCEDURE I.MWST
/
/   "2. Berechnung"
/ BETRAG = 9820 * 3
/ INCLUDE-PROCEDURE I.MWST
```

Gerufene Prozedur I.MWST:

```
/   "Betrag ausrechnen"
/ STEUER = (BETRAG * MWST + 50) / 100
/
/   "Betrag ausgeben"
/ CT-AUSGABE = STRING(STEUER)
/ CT-LAENGE  = LENGTH(CT-AUSGABE)
/ EUR-AUSGABE = SUBSTR(CT-AUSGABE,1,CT-LAENGE - 2) -
/     // DEZIMAL-ZEICHEN -
/     // SUBSTR(CT-AUSGABE,CT-LAENGE - 1)
/ WRITE-TEXT 'Die Steuer betraegt &EUR-AUSGABE EUR.'
```

Durch die Auslagerung der obigen Kommandos in eine Includeprozedur erspart man es sich hier, die Berechnung des Steuerbetrags an mehreren Stellen der ersten Prozedur vorzunehmen. Andererseits kann die Includeprozedur auf die Variablen DEZIMAL-ZEICHEN, BETRAG und MWST zugreifen, ohne dass diese jeweils als Parameter übergeben werden müssten.

Wenn eine Includeprozedur selbst Variablen deklariert, kann sie wählen, ob diese in der rufenden Prozedur sichtbar sein sollen. Die Sichtbarkeit ist insbesondere dann sinnvoll, wenn der Hauptzweck des Include-Aufrufs darin besteht, Deklarationen vorzunehmen.

Beispiel

Eine Prozedur

```
/DECLARE-VARIABLE NAME(TYPE=*STRING),SCOPE=*PROCEDURE
/DECLARE-VARIABLE LAND(TYPE=*STRING,INIT='D'),SCOPE=*PROCEDURE
/DECLARE-VARIABLE PLZ(TYPE=*INTEGER),SCOPE=*PROCEDURE
/DECLARE-VARIABLE ORT(TYPE=*STRING),SCOPE=*PROCEDURE
```

kann von mehreren Prozeduren mit /INCLUDE-PROCEDURE aufgerufen werden und sorgt jeweils für übereinstimmende Deklarationen der Variablen NAME, LAND, PLZ und ORT. Sollen später Änderungen an den Deklarationen notwendig sein, so genügt es, diese zentral an der Includeprozedur vorzunehmen.

In diesem Fall wird SCOPE = *PROCEDURE spezifiziert, damit die Deklarationen im Prozedur-Scope des Aufrufers wirksam werden und nach Abarbeitung des Include-Aufrufs erhalten bleiben.

Deklarationen mit SCOPE = *CURRENT hingegen sind in einer Includeprozedur „aufruflokal“; damit können Konflikte mit Variablendefinitionen in der aufrufenden Prozedur vermieden werden.

Beispiel

Wird in die Beispielprozedur I.MWST auf Seite 165 zu Beginn die Deklaration

```
/DECLARE-VARIABLE (STEUER, CT-AUSGABE, CT-LAENGE, EUR-AUSGABE)
```

aufgenommen, so kann auch die rufende Prozedur eine Variable mit einem der angegebenen Namen verwenden, ohne dass diese durch den Include-Aufruf verändert würde (SCOPE = *CURRENT ist voreingestellt und braucht daher im DECLARE-VARIABLE-Kommando nicht angegeben zu werden).

Prozedur-Unterbrechung

Wird eine Prozedur mit dem Kommando HOLD-PROCEDURE oder mit der K2-Taste unterbrochen, kann im Dialog auf alle prozedurlokalen Variablen der unterbrochenen Prozedur zugegriffen werden. Der Unterbrechungsdialog gehört also zum Sichtbarkeitsbereich der unterbrochenen Prozedur.

Variablen, die während eines Unterbrechungsdialogs deklariert werden, gelten anschließend als prozedurlokale Variablen der unterbrochenen Prozedur, gleich ob CALL- oder INCLUDE-Prozedur.

6.2.6 Variablenbehälter

Bei der Variablendeklaration können Variablen mit einem so genannten „Behälter“ oder auch „Container“ verknüpft werden. In diesem Behälter wird dann der Inhalt der Variablen abgelegt. Die Verknüpfung erfolgt über den Operanden CONTAINER des Kommandos DECLARE-VARIABLE.

Vorteil des „Behälter-Mechanismus“ ist, dass verschiedene Variablen mit demselben Behälter verknüpft werden können und so garantiert ist, dass alle diese Variablen den gleichen Inhalt haben.

Variablenbehälter können S-Variablen, Jobvariablen oder Behälter, die in PLAM-Bibliotheken abgespeichert werden, sein.

Hinweis

Ein Container muss immer schon vor der Variablen, die auf ihn Bezug nimmt, deklariert sein.

6.2.6.1 S-Variable als Behälter

Eine S-Variable kann Variablenbehälter für eine andere S-Variable sein. Allerdings müssen beide (Variablenbehälter und Variable) mit dem gleichen Variablen- und Datentyp deklariert werden und der Behälter muss die gleiche oder eine längere Lebensdauer als die Variable haben (die Lebensdauer ist durch den Geltungsbereich festgelegt).

Es gibt zwei verschiedene Möglichkeiten, S-Variablen als Variablenbehälter zu bestimmen, je nach dem wie dauerhaft die Variablen gespeichert werden sollen. So können Variablen nicht-permanent oder permanent durch Variablenbehälter gespeichert werden.

Hinweis

In Kommandobausteinen oder System-Ausgaberoutinen sind oft Variablennamen vorgegeben. Diese reservierten Variablennamen können im Operanden CONTAINER als Name der Behältervariablen eingesetzt werden. Der Benutzer erspart sich so das Umspeichern der vom System gelieferten Informationen in andere Variablen, die in der Prozedur gelten. Reservierte Wörter und Variablennamen sind im Abschnitt „Variablennamen“ auf Seite 155, aufgelistet.

Variablenbehälter für nicht-permanente Variablen

Soll eine Variable in einem Variablenbehälter nicht permanent gespeichert werden (d.h. sie ist höchstens bis zum Taskende vorhanden), muss der Variablenbehälter nur eine standardmäßig deklarierte Variable sein. Der Inhalt der zu speichernden Variablen wird in diesem Fall mit dem Variablenbehälter im Klasse-5-Speicher abgelegt.

Über den Behälter-Mechanismus können so auch taskglobale Variablen unter einem anderen Namen zugreifbar gemacht werden, wodurch eventuelle Namenskonflikte leichter umgangen werden können.

Variablenbehälter für permanente Variablen

Permanente Variablenbehälter werden in PLAM-Bibliotheken abgespeichert.

Geöffnet (bzw. wenn noch nicht vorhanden, auch erzeugt) werden diese Behälter durch das Kommando OPEN-VARIABLE-CONTAINER. Geschlossen werden sie explizit durch das Kommando CLOSE-VARIABLE-CONTAINER oder implizit durch Prozedur- bzw. Taskende (EXIT-PROCEDURE bzw. EXIT-JOB). Gesichert werden diese Variablenbehälter in PLAM-Bibliothekselemente mit dem Kommando SAVE-VARIABLE-CONTAINER. Informationen über offene Variablenbehälter können zudem mit SHOW-VARIABLE-CONTAINER-ATTR in strukturierte Ausgaben umgelenkt werden.

Wenn der Variablenbehälter geöffnet ist, kann auf die darin enthaltenen S-Variablen zugegriffen werden.

Eine Anwendungsmöglichkeit von permanenten Variablen ist, dass damit Prozedurparameter initialisiert werden können. Denn OPEN-VARIABLE-CONTAINER kann im Prozedurkopf zwischen BEGIN-PARAMETER-DECLARATION und dem ersten DECLARE-PARAMETER angegeben werden; d.h. ein Variablenbehälter kann vor der ersten Deklaration eines Parameters geöffnet werden, und somit können die darin deklarierten Variablen den jeweiligen Parameter initialisieren. (Mit der Voreinstellung *ALL in OPEN-VARIABLE-CONTAINER beim Operanden AUTOMATIC-DECLARE ist auch sichergestellt, dass standardmäßig alle im Variablenbehälter enthaltenen Variablen schon (vor-)deklariert sind. Der Geltungsbereich der Variablen ist dann der des Variablenbehälters.)

Wenn ein Parameter denselben Namen hat wie eine durch das Öffnen des Variablenbehälters deklarierte Variable, wird er zurückgewiesen und der Prozeduraufruf abgebrochen.

Hinweise

- Zur OPEN-VARIABLE-CONTAINER-Zeit ist das Bibliothekselement entweder nach dem Lesen geschlossen (LOCK-ELEMENT = *NO) oder nicht (LOCK-ELEMENT = *YES). D.h. Bibliothekselemente können daher entweder für mehrere Tasks oder nur für eine (exklusiver Zugriff) benützt werden.
- Wenn ein Variablenbehälter mit LOCK-ELEMENT = *YES geöffnet ist, werden alle nachfolgenden Öffnungsversuche in derselben oder in anderen Tasks abgewiesen.
- Alle Variablenattribute mit Ausnahme des Geltungsbereichs werden im Variablenbehälter gesichert. Das sind im einzelnen: Typ, Name der Struktur, Array- und Listen-Grenzen.
- Der Variablenbehälter wird automatisch ohne Angabe von CLOSE-VARIABLE-CONTAINER beim Prozedurende geschlossen, wenn beim Öffnen SCOPE = *CURRENT eingestellt wurde oder bei einem untergeordneten Aufruf SCOPE = *PROCEDURE. Auch bei Taskende wird der Variablenbehälter automatisch geschlossen, wenn SCOPE = *TASK angegeben wurde.

– Allerdings wird dabei mit Ausnahme des letzten Falls der Variablenbehälter nicht gesichert; und im letzten Fall erfolgt auch nur eine Sicherung, wenn SAVE-AT-TERMINATION = *YES eingestellt wurde.

PLAM-Bibliothekselemente als Ablage für Variablenbehälter werden folgendermaßen definiert (siehe auch Handbuch „LMS“ [11]):

– Variablenbehälter werden als PLAM-Bibliothekselemente mit TYPE = SYSVCONT in LMS aufgenommen. Sie können so von SDF-P und FHS benützt werden.
– Wenn das angegebene Bibliothekselement beim Öffnen noch nicht existiert, wird es bei der Einstellung LOCK-ELEMENT = *YES mit dem Öffnen erzeugt, bei der Einstellung LOCK-ELEMENT = *NO mit dem Sichern.
– Wenn das Element beim Öffnen gesperrt ist, kann es nicht mit /SAVE-VARIABLE-CONTAINER CONTAINER-NAME = <composed-name 1..64>(ELEMENT-VERSION = *INCREMENT) gesichert werden.
– Wenn die Element-Version beim Öffnen nicht angegeben wird und das Element noch nicht existiert, wird das Element mit der höchst möglichen Version X'FF' erzeugt.
– Wenn die Element-Version beim Öffnen nicht angegeben wird, aber das Element schon existiert, wird es in der höchsten Version eröffnet.
– Wenn eine Element-Version außer *UPPER-LIMIT (in LMS) beim Öffnen eingestellt ist und das Element nicht existiert, kann es nach dem Öffnen und Erzeugen mit /SAVE-VARIABLE-CONTAINER CONTAINER-NAME = <composed-name 1..64>(ELEMENT-VERSION = *INCREMENT) gesichert werden.
– Wenn ein Element mit /SAVE-VARIABLE-CONTAINER CONTAINER-NAME = <composed-name 1..64>(ELEMENT-VERSION = *SAME) gesichert wird, wird es auf Grund der Öffnen-Einstellung LOCK-ELEMENT = *YES in derselben Version zurückgeschrieben, auf Grund der Öffnen-Einstellung LOCK-ELEMENT = *NO in der höchsten Version.
– Wenn ein Element mit /SAVE-VARIABLE-CONTAINER CONTAINER-NAME = <composed-name 1..64>(ELEMENT-VERSION = *INCREMENT) gesichert wird, wird der Zuwachs beim Sichern und nicht beim nächsten Öffnen angehängt.

Beispiel

Eingabe

```
/OPEN-VARIABLE-CONTAINER CONT, *LIBRARY-ELEMENT(#MY-CONT-LIB)
/SHOW-VARIABLE-CONTAINER-ATTR CONT
```

Ausgabe

```
CONTAINER-NAME = CONT
   FROM-FILE = *LIBRARY-ELEMENT
      LIBRARY = :1OSC:$QM123.S.152.OCG6.MY-CONT-LIB
      ELEMENT = CONT
      VERSION = *HIGHEST-EXISTING
   LOCK      = *NO
   SCOPE     = *PROCEDURE
```

Eingabe

```
SAVE-VARIABLE-CONTAINER CONT
CLOSE-VARIABLE-CONTAINER CONT
```

Weitere Hinweise zu „permanenten Variablen“ siehe im Kapitel „SDF-P-Kommandos“, vor allem in der Beschreibung zum Kommando OPEN-VARIABLE-CONTAINER, Seite 710.

6.2.6.2 Jobvariable als Behälter

Wenn eine Variable mit einer Jobvariablen verknüpft wird, wird der Inhalt der deklarierten Variablen in der Jobvariablen abgelegt. Dabei spielt es keine Rolle, mit welchem Geltungsbereich die Variable deklariert ist.

Die Verknüpfung erfolgt über den Operanden CONTAINER = *JV(jvname), mit jvname als Name der Jobvariablen.

Jobvariablen werden wie Dateien katalogisiert. Für den Zugriff auf Jobvariablen gelten daher die gleichen Bedingungen wie für den Zugriff auf Dateien, einschließlich Kennwortschutz. Informationen zu Jobvariablen enthält das Handbuch „Jobvariablen“ [5].

Der Kennwortschutz für Jobvariablen bleibt voll erhalten. Auf kennwortgeschützte Jobvariablen kann auch über mit ihr verbundene S-Variablen nur dann zugegriffen werden, wenn zuvor das entsprechende Kennwort mit dem Kommando ADD-PASSWORD in die Kennworttabelle der Task eingetragen wurde.

Die Definition einer Jobvariablen als Variablen-Behälter hat keinen Einfluss auf die Lebensdauer der Variablen. Die mit ihr verknüpfte Jobvariable wird jedoch nicht automatisch gelöscht. Sie muss explizit mit dem Kommando FREE-VARIABLE, Operand DESTROY-CONTAINER gelöscht werden.

Bei der Deklaration von Variablen, die mit Jobvariablen verknüpft werden, gelten folgende Einschränkungen:

– Die Variable muss den Datentyp STRING haben.
– Der Variableninhalt darf höchstens 256 Byte lang sein.

Begründung: In Jobvariablen wird kein Datentyp-Attribut geführt, und für den Jobvariablen-Wert sind maximal 256 Bytes vorgesehen.

Die Variable wird so angelegt, wie sie im Kommando DECLARE-VARIABLE deklariert ist. Der Inhalt der Variablen wird jedoch nicht im Klasse-5-Speicher abgelegt, sondern in der Jobvariablen. Zugriff auf die Variable ist also gleichzeitig Zugriff auf die Jobvariable.

Bei Jobvariablen kann nicht unterschieden werden zwischen nicht-initialisierten Jobvariablen (= „kein Wert zugewiesen“) und Jobvariablen mit dem Inhalt „Leerstring“ (= String der Länge 0, C''). Der Zustand „nicht initialisiert“ wird derzeit folgendermaßen definiert: Eine Jobvariable ist nicht initialisiert, wenn sie den String 256 X'EE' enthält. Diese Zeichenkette darf daher Variablen, die mit Jobvariablen verknüpft sind, nicht als normaler Textstring zugewiesen werden.

6.2.7 Mehrfachdeklaration

„Mehrfachdeklaration“ bedeutet, dass eine Variable, die bereits an anderer Stelle der Prozedur deklariert ist, noch einmal deklariert wird. Das Neu-Deklarieren taskglobaler Variablen in untergeordneten Prozeduren ist zum Beispiel solch eine Mehrfachdeklaration.

Mehrfachdeklarationen werden nur dann akzeptiert, wenn die Variable mit genau denselben Attributen (außer CONTAINER und INIT) deklariert wird wie die bereits existierende Variable. In diesem Fall wird die Deklaration ignoriert. Es wird keine neue Variable angelegt.

Existiert die Variable bereits mit einem Container, dann genügt bei einer Mehrfachdeklaration die Angabe CONTAINER = *STD. Dies hat die gleiche Wirkung wie die korrekte Angabe des Containers.

Existiert die Variable bereits mit einem Wert, dann genügt bei einer Mehrfachdeklaration die Angabe INIT = *NONE. Die Variable wird nicht entwertet, sondern behält ihren Wert. Dies hat die gleiche Wirkung wie die direkte Angabe des Anfangswertes.

Stimmen die Deklarationen nicht überein, erfolgt die Fehlerbehandlung.

Dies gilt auch, wenn zunächst implizit deklarierte Variablen später explizit neu deklariert werden. Sie müssen dann mit den Defaultwerten deklariert werden.

Auf diese Weise können zum Beispiel Prozedurparameter als normale S-Variablen deklariert werden; die Merkmale der Prozedurparameter müssen jedoch erhalten bleiben.

6.3 Variablenverarbeitung

Dieses Kapitel beschreibt die Wertzuweisung an Variablen, das Löschen von Variablen und die Ausgabe von Variablen und Strukturlayouts.

6.3.1 Wertzuweisung an Variablen

„Wertzuweisung“ bedeutet, dass einer Variablen oder einem Variablenaggregat ein Inhalt, das heißt ein Wert, zugewiesen wird.

Es kann unterschieden werden zwischen direkter Wertzuweisung mit dem Kommando SET-VARIABLE und der Wertzuweisung über TERMINAL mit dem Kommando READ-VARIABLE. In diesem Abschnitt ist nur die direkte Wertzuweisung mit SET-VARIABLE beschrieben. Die Wertzuweisung mit READ-VARIABLE wird im Abschnitt „Variableninhalte einlesen“ auf Seite 181, beschrieben.

Mit dem Kommando SET-VARIABLE werden Variablen Werte zugewiesen. Die zugewiesenen Werte können Inhalte anderer Variablen sein oder Ergebnisse von Funktionen oder zusammengesetzten Ausdrücken. SET-VARIABLE kann sowohl auf einfache als auch auf zusammengesetzte Variablen angewendet werden. Die entsprechenden Regeln sind weiter unten beschrieben.

Der Kommandoname SET-VARIABLE braucht in Zuweisungen nicht genannt zu sein, er kann entfallen. Die Zeichenfolge auf der linken Seite des Gleichheitszeichens wird als Variablenname interpretiert.

Wenn implizite Deklaration für die Prozedur zulässig ist, braucht die Variable nicht deklariert zu sein (implizite Deklaration ist nur für einfache Variablen zulässig und wenn IMPLICIT-DECLARATION = *YES gilt; siehe Kommando SET-PROCEDURE-OPTIONS auf Seite 747 oder MODIFY-PROCEDURE-OPTIONS auf Seite 701).

Ist implizite Deklaration unterbunden, muss die Variable vor der ersten Zuweisung deklariert werden. Zuweisung an eine nicht deklarierte Variable führt dann zu einem Fehler.

Voraussetzung für die Zuweisung ist immer, dass der Datentyp des zugewiesenen Wertes dem Datentyp der Variablen auf der linken Seite der Zuweisung entspricht. Nur wenn die Variable auf der linken Seite der Zuweisung mit dem Datentyp ANY deklariert ist, können ihr beliebige Werte zugewiesen werden.

Mit dem Kommando DECLARE-CONSTANT initialisierten Variablen kann mit SET-VARIABLE kein neuer Wert zugewiesen werden.

6.3.1.1 Einfache Variablen

Wird einer einfachen Variablen ein Wert zugewiesen, wird immer der bisherige Wert überschrieben: Der Inhalt der Variablen wird implizit mit FREE-VARIABLE gelöscht; anschließend wird der Variablen der Wert, der sich aus dem Ausdruck rechts vom Gleichheitszeichen ergibt, zugewiesen. Dieser Zuweisungsmodus ist im Kommando SET-VARIABLE voreingestellt (WRITE-MODE = *REPLACE).

6.3.1.2 Zusammengesetzte Variablen

Elementen von zusammengesetzten Variablen aller Art können genauso wie einfachen Variablen einfache Werte zugewiesen werden.

In Zuweisungen können auch zusammengesetzte Variablen eingesetzt werden. Dann müssen die zusammengesetzten Variablen auf beiden Seiten der Zuweisung vom gleichen Typ sein.

Gilt die Zuweisung für zusammengesetzte Variablen, können außer dem Zuweisungsmodus überschreiben (= *REPLACE) auch andere Zuweisungsmodi verwendet werden: Mischen (= *MERGE) oder Erweitern (= *EXTEND oder *PREFIX). Dabei ist allerdings zu beachten, über welche Modi zusammengesetzte Variablen bei der Zuweisung verknüpft werden können und wie diese Modi sich auf den Inhalt der Variablenelemente und den Aufbau der zusammengesetzten Variablen auswirken.

Arrays

Wenn einem Array als Ganzem Werte zugewiesen werden, muss auch auf der rechten Seite der Zuweisung ein Array angegeben sein, der vom selben Typ ist:

`/SET-VARIABLE array`$_1$ `= array`$_2$

Für Arrays gelten die Modi Überschreiben (WRITE-MODE = *REPLACE) und Mischen (WRITE-MODE = *MERGE).

*Überschreiben (WRITE-MODE = *REPLACE)*

Zuerst werden alle Elemente von $array_1$ gelöscht. Dann werden die Elemente von $array_2$ unter dem Namen $array_1$ angelegt mit den Werten von $array_2$.

Beispiel

```
/DECLARE-VARIABLE array1, MULTIPLE-ELEMENTS=*ARRAY
/DECLARE-VARIABLE array2, MULTIPLE-ELEMENTS=*ARRAY
/array1#1=1
/array2#2=2
/SET-VARIABLE array1 = array2, WRITE-MODE=*REPLACE
/SHOW-VARIABLE array1
array1#2 = 2
```

*Mischen (MODE = *MERGE)*

Wenn die Arrays $array_1$ und $array_2$ in ihren Elementnamen identisch sind, sind auch Mischen und Überschreiben identisch, d.h. $array_1\#index_i = array_2\#index_i$ (WRITE-MODE = *MERGE hat dieselbe Wirkung wie WRITE-MODE = *REPLACE).

Wenn $array_1$ und $array_2$ nicht identisch sind, ist Mischen nur sinnvoll, wenn die Elemente von $array_1$ und $array_2$ unterschiedlichen Arrayindex haben.

Elemente von $array_1$, für die es im Array $array_2$ kein Pendant mit gleichem Arrayindex gibt, bleiben unbeeinflusst. Sie werden nicht implizit gelöscht.

Enthält $array_2$ Elemente, für die es kein Pendant in $array_1$ gibt, wird $array_1$ um diese Elemente erweitert, d.h. die Elemente werden entsprechend ihrem Index in $array_1$ einsortiert.

Beispiel

```
/DECLARE-VARIABLE array1, MULTIPLE-ELEMENTS=*ARRAY
/DECLARE-VARIABLE array2, MULTIPLE-ELEMENTS=*ARRAY
/array1#1 = 1
/array1#3 = 4
/array2#2 = 2
/array2#4 = 8
/SET-VARIABLE array1 = array2, WRITE-MODE=*MERGE
/SHOW-VARIABLE array1
array1#1 = 1
array1#2 = 2
array1#3 = 4
array1#4 = 8
```

Listen

Wenn einer Liste Werte zugewiesen werden, hängt es vom Zuweisungsmodus ab, was auf der rechten Seite der Zuweisung stehen darf.

Für Listen gelten die Modi „Überschreiben“ (WRITE-MODE = *REPLACE) und „Erweitern“ (WRITE-MODE = *EXTEND, WRITE-MODE = *PREFIX).

*Überschreiben (WRITE-MODE = *REPLACE)*

Auf der rechten Seite der Zuweisung muss eine Liste angegeben sein:

```
/SET-VARIABLE liste1 = liste2, WRITE-MODE = *REPLACE
```

Zuerst werden alle Elemente von $liste_1$ gelöscht. Dann werden die Elemente von $liste_2$ unter dem Namen $liste_1$ angelegt mit den Werten von $liste_2$.

*Erweitern (WRITE-MODE = *EXTEND)*

Wenn an eine Liste ein oder mehrere Elemente angehängt werden sollen, muss im SET-VARIABLE-Kommando der Operand WRITE-MODE = *EXTEND eingesetzt werden. In Zusammenhang mit WRITE-MODE = *EXTEND kann einer Liste ein Ausdruck oder eine andere Liste zugewiesen werden:

```
/SET-VARIABLE liste1 = ausdruck, WRITE-MODE = *EXTEND
/SET-VARIABLE liste1 = liste2, WRITE-MODE = *EXTEND
```

Mit /SET-VARIABLE $liste_1$ = ausdruck, WRITE-MODE = *EXTEND wird $liste_1$ um ein Element erweitert, d.h., an $liste_1$ wird ein Element angehängt. Diesem Element wird dann das Ergebnis von „ausdruck“ zugewiesen.

Gilt /SET-VARIABLE $liste_1$ = $liste_2$, WRITE-MODE = *EXTEND, wird $liste_2$ an $liste_1$ angehängt. Das heißt, an $liste_1$ werden so viele Elemente angehängt, wie $liste_2$ besitzt. Diesen neuen Elementen werden der Reihe nach die Inhalte der Elemente von $liste_2$ zugewiesen.

*Erweitern (WRITE-MODE = *PREFIX)*

Wenn eine Liste nach vorn um ein oder mehrere Elemente erweitert werden soll, muss im SET-VARIABLE-Kommando der Operand WRITE-MODE = *PREFIX eingesetzt werden. In Zusammenhang mit WRITE-MODE = *PREFIX kann einer Liste ein Ausdruck oder eine andere Liste zugewiesen werden:

```
/SET-VARIABLE liste1 = ausdruck, WRITE-MODE = *PREFIX
/SET-VARIABLE liste1 = liste2, WRITE-MODE = *PREFIX
```

Mit /SET-VARIABLE $liste_1$ = ausdruck, WRITE-MODE = *PREFIX wird $liste_1$ um ein Element erweitert: vor das bisher erste Element von $liste_1$ wird ein neues Element eingefügt ($liste_1$ erhält einen neuen Listenkopf). Diesem Element wird dann das Ergebnis von „ausdruck“ zugewiesen.

Gilt /SET-VARIABLE $liste_1$ = $liste_2$, WRITE-MODE = *PREFIX, wird $liste_2$ vor das bisher erste Element von $liste_1$ (den Listenkopf) eingefügt: vor den Listenkopf von $liste_1$ werden so viele Elemente eingefügt, wie $liste_2$ besitzt. Diesen neuen Elementen werden der Reihe nach die Inhalte der Elemente von $liste_2$ zugewiesen.

Strukturen

Wenn einer Struktur als Ganzer Werte zugewiesen werden, muss auch auf der rechten Seite der Zuweisung eine Struktur angegeben sein:

/SET-VARIABLE $struktur_1$ = $struktur_2$

Für Strukturen gelten die Modi Überschreiben (WRITE-MODE = *REPLACE) und Mischen (WRITE-MODE = *MERGE).

*Überschreiben (WRITE-MODE = *REPLACE)*

Zu unterscheiden ist, ob $struktur_1$ eine statische oder dynamische Struktur ist:

Wenn $struktur_1$ eine statische Struktur ist, gilt die Zuweisung nur für solche Elemente von $struktur_1$, für die es in $struktur_2$ Elemente mit gleichem Elementnamen gibt. Alle Inhalte der Elemente von $struktur_1$ werden zunächst implizit gelöscht (implizites FREE-VARIABLE). Dann wird den Elementen von $struktur_1$ jeweils der Inhalt des entsprechenden Elements von $struktur_2$ zugewiesen. Elemente, die in der jeweils anderen Struktur kein Pendant haben, bleiben unberücksichtigt.

Beispiel

```
/DECLARE-VARIABLE struktur1(TYPE=*STRUCTURE(*BY-SYSCMD))
/BEGIN-STRUCTURE
/   DECLARE-ELEMENT A(INITIAL-VALUE='A')
/   DECLARE-ELEMENT B(INITIAL-VALUE='B')
/END-STRUCTURE
/DECLARE-VARIABLE struktur2(TYPE=*STRUCTURE)
/STRUKTUR2.A='C'
/STRUKTUR2.B='D'
/STRUKTUR2.C='E'
/SET-VARIABLE struktur1=struktur2,WRITE-MODE=*REPLACE
/SHOW-VARIABLE struktur1
STRUKTUR1.A = C
STRUKTUR1.B = D
```

Wenn $struktur_1$ eine dynamische Struktur ist, wird analog zu Arrays zunächst ein implizites FREE-VARIABLE auf $struktur_1$ gemacht, dann wird implizit für jedes Element von $struktur_2$ ein Element in $struktur_1$ deklariert. Diesen neuen Elementen von $struktur_1$ werden dann die Inhalte der Elemente von $struktur_2$ zugewiesen.

*Mischen (MODE = *MERGE)*

Wenn die Strukturen $struktur_1$ und $struktur_2$ identisch sind, sind „Mischen“ und „Überschreiben“ identisch (WRITE-MODE = *MERGE hat die gleiche Wirkung wie WRITE-MODE = *REPLACE).

Wenn $struktur_1$ und $struktur_2$ nicht identisch sind, muss wieder unterschieden werden, ob $struktur_1$ eine statische oder eine dynamische Struktur ist.

Wenn $struktur_1$ eine statische Struktur ist, wirkt sich die Zuweisung wie beim Überschreiben zunächst auf alle die Elemente von $struktur_1$ aus, die den gleichen Elementnamen haben wie Elemente von $struktur_2$. Diesen Elementen von $struktur_1$ wird der Inhalt des jeweiligen Elements von $struktur_2$ zugewiesen. Enthält $struktur_2$ noch weitere Elemente, wird $struktur_1$ dennoch nicht erweitert.

Wenn $struktur_1$ eine dynamische Struktur ist, wird implizit für jedes Element von $struktur_2$ ein Element in $struktur_1$ deklariert. Diesen neuen Elementen von struktur1 werden dann die Inhalte der Elemente von $struktur_2$ zugewiesen. (Die Wirkung von WRITE-MODE = *MERGE entspricht nicht der von WRITE-MODE = *REPLACE). Elemente von $struktur_1$, die kein Pendant in $struktur_2$ haben, bleiben unbeeinflusst. Den Elementen von $struktur_1$, die ein Pendant in $struktur_2$ haben, werden die Inhalte der Elemente von $struktur_2$ zugewiesen.

Beispiel

```
/DECLARE-VARIABLE STRUCTUR1(TYPE=*STRUCTURE(*DYNAMIC))
/STRUCTUR1.A='A'
/STRUCTUR1.B='B'
/STRUCTUR1.D='X'
/DECLARE-VARIABLE STRUCTUR2(TYPE=*STRUCTURE)
/STRUCTUR2.A='C'
/STRUCTUR2.B='D'
/STRUCTUR2.C='E'
/SET-VARIABLE STRUCTUR1=STRUCTUR2,WRITE-MODE=*MERGE
/SHOW-VARIABLE STRUCTUR1
STRUCTUR1.A = C
STRUCTUR1.B = D
STRUCTUR1.D = X
STRUCTUR1.C = E
/SHOW-VARIABLE STRUCTUR2
STRUCTUR2.A = C
STRUCTUR2.B = D
STRUCTUR2.C = E
```

6.3.2 Variablen und Variablendeklarationen löschen und entfernen

Es muss hier zuerst beachtet werden, dass es sowohl einen Unterschied gibt zwischen dem Löschen und Entfernen als auch zwischen dem Löschen von Variableninhalten und dem Löschen von Variablendeklarationen.

So werden Variableninhalte oder Variablendeklarationen gelöscht. Dagegen werden Elemente zusammengesetzter Variablen innerhalb der Deklaration einschließlich ihres Inhalts entfernt. Dabei können Variableninhalte implizit und explizit gelöscht werden: explizit mit dem Kommando FREE-VARIABLE, implizit bei jeder neuen Wertzuweisung. (Denn bevor einer einfachen Variablen ein neuer Wert zugewiesen wird, wird der alte Variableninhalt gelöscht.)

Variablendeklarationen werden mit dem Kommando DELETE-VARIABLE gelöscht.

Elemente zusammengesetzter Variablen werden einschließlich ihrer Werte wiederum mit dem Kommando FREE-VARIABLE entfernt. Anschließend kann auf ihren Inhalt nicht mehr zugegriffen werden, auch eine erneute Deklaration ist nicht möglich.

Hinweis

In diesem Zusammenhang muss auch berücksichtigt werden, dass Variablendeklaration und Variableninhalt auch von der Geltungsdauer der Variablen abhängig sind (siehe dazu den Abschnitt „Geltungsbereich von Variablen“ auf Seite 162).

6.3.2.1 Variableninhalt löschen und Elemente entfernen

Für zusammengesetzte Variablen kann in Zuweisungen „Überschreiben“ oder „Erweitern“ vereinbart werden. „Überschreiben“ bedeutet dann, dass wie bei einfachen Variablen der alte Inhalt der betroffenen Elemente gelöscht wird, bevor ihnen ein neuer Inhalt zugewiesen wird. „Erweitern“ hat keinen Einfluss auf den Inhalt der bestehenden Elemente, sie werden nicht gelöscht.

Beim expliziten Löschen von Variableninhalten mit dem Kommando FREE-VARIABLE müssen die Auswirkungen auf zusammengesetzte Variablen und Variablenbehälter beachtet werden.

Grundsätzlich muss berücksichtigt werden, dass ein FREE-VARIABLE auf ein Array, auf eine Liste oder auf eine dynamische Struktur ein Löschen und Entfernen aller jeweiliger Elemente bewirkt. Dagegen bewirkt ein FREE-VARIABLE auf ein Element eines Arrays, einer Liste oder einer dynamischen Struktur lediglich das Löschen und Entfernen dieses einen Elements.

Mit dem Kommando DECLARE-CONSTANT initialisierte Variablen können mit FREE-VARIABLE nicht freigegeben werden.

Listen

Bezieht sich FREE-VARIABLE auf eine Liste, werden alle Listenelemente gelöscht und entfernt. Die Deklaration bleibt erhalten.

Bezieht sich FREE-VARIABLE auf ein Listenelement oder einen Bereich von Listenelementen, so wird nach dem Löschen und Entfernen die Liste neu durchnummeriert, da in Listen keine Lücken entstehen dürfen.

Arrays

Bezieht sich FREE-VARIABLE auf ein Array, werden alle Arrayelemente gelöscht und entfernt; die Deklaration bleibt erhalten.

Bezieht sich FREE-VARIABLE auf ein Arrayelement, wird dieses Element gelöscht und entfernt.

Sind die Arrayelemente selbst Strukturen, werden die Elemente dieser Strukturen gelöscht, die Strukturdeklaration bleibt jedoch erhalten. Für dynamische Strukturen bedeutet dies, dass die Deklaration TYPE = *STRUCTURE(*DYNAMIC) erhalten bleibt, für statische Strukturen, dass die Deklaration der Elemente erhalten bleibt (dasselbe gilt bei Listen).

Strukturen

Bezieht sich FREE-VARIABLE auf eine statische Struktur, werden die Elemente gelöscht, die Strukturdeklaration, das heißt die Deklaration der Elemente bleibt erhalten. Entsprechend gilt, dass der Inhalt des Elements gelöscht wird, wenn sich FREE-VARIABLE auf das Element einer statischen Struktur bezieht.

Bezieht sich FREE-VARIABLE auf eine dynamische Struktur, werden die Inhalte der Elemente dieser Struktur gelöscht und alle Elemente entfernt. Wenn sich FREE-VARIABLE auf ein Element einer dynamischen Struktur bezieht, wird dieses Element entfernt. Wenn das Strukturelement selbst eine zusammengesetzte Variable ist, werden auch alle Elemente dieser Variablen (und alle weiteren evtl. daran hängenden Unterelemente) entfernt.

Variablenbehälter

Variablen können mit anderen Variablen oder Jobvariablen als Variablenbehälter verknüpft sein. Ist bei dieser Verknüpfung der Variablenbehälter eine Variable, wird der Inhalt der Behältervariablen analog zur Angabe des Kommandos FREE-VARIABLE gelöscht.

Ist der Variablenbehälter eine Jobvariable, kann der Programmierer festlegen, ob die Jobvariable zusammen mit der Variablen bei der Angabe von FREE-VARIABLE gelöscht wird. „Jobvariable löschen“ bedeutet hier, dass der Katalogeintrag der Jobvariablen gelöscht wird.

6.3.2.2 Variablendeklarationen löschen

Mit DELETE-VARIABLE kann die Variablendeklaration einer Variablen im aktuellen Geltungsbereich gelöscht werden, d.h. auch Deklarationen von importierten Task-Variablen. Der Name der Variablen kann danach nicht länger benützt werden, dazu ist der Inhalt auch nicht mehr vorhanden.

Folgende Variablendeklarationen können mit DELETE-VARIABLE nicht gelöscht werden:

- Prozedurparameter
- Elemente von zusammengesetzten Variablen
- Systemvariablen (z.B. SYSSWITCH)
- Behälter-JVs
- Variablenbehälter für nicht-permanente Variablen
- Strukturlayouts

6.3.3 Variableninhalte einlesen

Zum Einlesen von Variableninhalten wird das Kommando READ-VARIABLE benutzt. Es liest standardmäßig Daten von *TERMINAL und weist diese dann einer Variablen zu.

6.3.3.1 Eingabeziel

Der Operand VARIABLE-NAME des Kommandos bezeichnet die Variable, der ein neuer Inhalt zugewiesen werden soll. Diese Variable kann eine einfache oder zusammengesetzte Variable sein. Bei jeder Zuweisung wird standardmäßig zunächst der alte Variableninhalt gelöscht und dann der neue Inhalt zugewiesen.

Variablennamen direkt angeben

Das Eingabeziel kann im Operanden VARIABLE-NAME direkt benannt werden, als Variablenname einer einfachen Variablen oder als Variablenname einer Struktur.

Bezeichnet NAME eine einfache Variable, muss diese Variable überschreibbar sein. Ist die angegebene Variable kein Variablenelement und noch nicht deklariert, wird sie implizit deklariert mit dem Geltungsbereich SCOPE = *CURRENT und dem Datentyp *ANY, wenn implizite Deklaration von Variablen zulässig ist.

Wenn NAME ein Variablenelement bezeichnet, das noch nicht existiert, wird es ebenfalls implizit deklariert, falls die übergeordnete zusammengesetzte Variable existiert und erweitert werden kann.

Es können mehrere Variablennamen (bis zu 2000) in Form einer Liste angegeben werden, sie sind dann in runde Klammern einzuschließen: (varname$_1$, varname$_2$, ...). Für jeden dieser Variablennamen gelten die vorher beschriebenen Regeln.

Werden Variablen in einer solchen Liste angegeben, muss jeder dieser Variablen ein Wert zugewiesen werden können. Wenn die Eingabe kürzer ist als diese Liste, das heißt, wenn nicht allen Variablen Werte zugewiesen werden können, wird die Fehlerbehandlung angestoßen.

Variablen implizit erzeugen

Im Operanden NAME des Kommandos READ-VARIABLE können implizit Variablen erzeugt werden, abhängig von der Eingabe (Operand VARIABLE-NAME = *BY-INPUT(.:.)). Diese Angabe setzt voraus, dass Variablen in dem Format eingelesen werden, das das Kommando SHOW-VARIABLE mit seinen Operanden bei der Ausgabe von Variablen erzeugt. Das heißt, es werden Variablenname und Variablenwert eingelesen.

Werden einfache Variablen eingelesen, die bereits existieren, wird ihnen ein Wert zugewiesen.

Handelt es sich bei den einfachen Variablen um Variablenelemente, sind folgende Fälle zu unterscheiden:

– Es wird ein Listenelement eingelesen: Das Kommando wird abgewiesen.
– Es wird ein Strukturelement eingelesen, dessen übergeordnete Struktur noch nicht deklariert ist: Die Struktur wird implizit mit SCOPE = *CURRENT angelegt.
  Wenn eine statische Struktur eingelesen wird, die in der Prozedur noch nicht existiert, wird sie nicht als statische, sondern als dynamische Struktur neu angelegt.

Sowohl für das Einlesen einfacher als auch zusammengesetzter Variablen gilt: Wird eine Variable eingelesen, die noch nicht initialisiert ist (Inhalt *NO-INIT), das heißt, der noch kein Wert zugewiesen wurde, wird die entsprechende Variable in der Prozedur implizit gelöscht (implizites FREE-VARIABLE).

In eine Listenvariable einlesen

Im Operanden VARIABLE-NAME des Kommandos READ-VARIABLE kann der Name einer Listenvariablen angegeben werden. Wenn in der Prozedur noch keine Liste dieses Namens existiert, wird, wenn zulässig, implizit eine Liste angelegt mit dem Geltungsbereich SCOPE = *CURRENT und dem Datentyp TYPE = *STRING.

Die Liste kann überschrieben (WRITE-MODE = *REPLACE) oder erweitert werden (WRITE-MODE = *EXTEND).

„Überschreiben“ bedeutet: Der alte Inhalt der Liste wird zunächst gelöscht. Dann werden den Listenelementen der Reihe nach die eingelesenen Werte zugewiesen, angefangen beim Listenkopf. „Erweitern“ bedeutet: Der alte Inhalt der Liste bleibt erhalten. Bei jeder Zuweisung wird die Liste um ein Element erweitert. Das heißt, an das Ende der Liste wird ein Element angehängt, dem dann der eingelesene Wert zugewiesen wird.

6.3.3.2 Eingabemedium

Die Eingabequelle wird im Operanden INPUT definiert. Standardmäßig ist die Datensichtstation *TERMINAL eingestellt. Eingabequelle kann aber auch sein:

– das Terminal
– eine katalogisierte Datei
– eine Variable
– ein Bibliothekselement
– die Systemdatei SYSDTA

Wie die eingelesenen Werte einer Variablen zugewiesen werden, hängt von den Angaben im Operanden VARIABLE-NAME ab (siehe Seite 715).

Terminal

Die Eingaben werden von der Datenstation (= *TERMINAL) gelesen. Jede Eingabe, die mit der DUE-Taste abgeschlossen wird, gilt dann als ein Variablenwert.

Durch den Unteroperanden SECRET-INPUT hat man zudem die Möglichkeit einer geheimen Datenstation-Eingabe über ein geschütztes Feld.

Benutzerdatei / Bibliothekselement

Benutzerdatei und Bibliothekselement unterscheiden sich nur in der Art der Speicherung und daher in der Angabe des Namens. Wenn als Eingabequelle eine Benutzerdatei oder ein Bibliothekselement angegeben ist, wird jeder Datensatz als ein Wert interpretiert, der einer Variablen zugewiesen wird.

Für ISAM-Dateien kann eingestellt werden, ob der ISAM-Schlüssel erhalten bleiben oder bei der Eingabe entfernt werden soll. Standardmäßig wird der ISAM-Schlüssel entfernt und nur der Rest des Datensatzes einer Variablen zugewiesen.

Listenvariable

Wird als Eingabequelle eine Listenvariable angegeben, muss diese Liste mit dem Datentyp *STRING deklariert sein und einen gültigen Inhalt haben. (Die Liste kann auch mit dem Datentyp *ANY deklariert sein, wenn sie nur String-Werte enthält.)
Jedes Element dieser Liste gilt dann als ein Wert, der einer Variablen zugewiesen wird.

Systemdatei SYSDTA

Daten werden von SYSDTA genauso eingelesen wie von einer Datei. Allerdings mit folgender Ausnahme: Wenn das Lesen vor SYSDTA- EOF beendet wird, wird beim nächsten Lesen von SYSDTA auf den nächsten SYSDTA-Satz zugegriffen. Das erlaubt gezieltes Lesen von SYSDTA, z.B. für den Fall: wenn nur eine einfache Variable gelesen werden soll.
Die Eingabe endet mit dem Ende der Systemdatei SYSDTA, d.h. entweder bei Dateiende, wenn SYSDTA einer katalogisierten Datei zugewiesen ist, oder mit dem nächsten Kommando, wenn SYSDTA SYSCMD zugewiesen ist (Default-Voreinstellung von S-Prozeduren).
Einlesen von SYSDTA kann mit HOLD-PROGRAM, K2 oder BEGIN-BLOCK PROGRAM-INPUT = *MIXED-WITH-CMD nicht unterbrochen werden (um in den Kommando-Modus umzuschalten). Diese Kommandos bzw. Maßnahmen verursachen nur, dass ein Einlesestopp, d.h. SYSDTA-EOF (Dateiende), zum jeweiligen Kommando gemeldet wird. Nur das Kommando SEND-DATA beendet das READ-VARIABLE-Kommando nicht.

6.3.4 Variablen ausgeben

Variableninhalte können mit dem Kommando SHOW-VARIABLE ausgegeben werden. SHOW-VARIABLE kann auf mehrere Variablen gleichzeitig angewendet werden, die Ausgabe erfolgt standardmäßig auf SYSOUT.

6.3.4.1 Ausgabequelle

Auf welche Variable(n) sich der Kommandoaufruf beziehen soll, wird in zwei Operanden festgelegt: die Variablennamen im Operand VARIABLE-NAME, als zusätzliches Suchkriterium der Geltungsbereich im Operanden SELECT. Auf diese Weise kann der Programmierer nicht nur abfragen, welche Variablen in seiner Prozedur deklariert wurden, sondern auch, welche Variablen zum Zeitpunkt des Auftrags sichtbar sind.
Die Art der Ausgabe wird durch einen Operanden gesteuert: INFORMATION. Er legt einerseits den Datentyp der Ausgabe fest und andererseits ob nur der Variablenwert oder auch der Variablenname ausgegeben werden soll. Wenn eine Variable zwar deklariert, aber noch nicht initialisiert ist, wird als Variablenwert der String „NO-INIT“ ausgegeben (allerdings nur bei INFORMATION = *PAR(VALUE = *C-LITERAL bzw. VALUE = *X-LITERAL bzw. VALUE = *WITHOUT-QUOTES)).

6.3.4.2 Ausgabeziel

Das Ausgabeziel wird mit dem Operanden OUTPUT definiert. Ausgabeziel kann sein:
- die Systemdateien SYSOUT (Voreinstellung) oder SYSLST
- eine Benutzerdatei oder ein Bibliothekselement
- eine Listenvariable

Bei der Ausgabe von Variablen entspricht jeder Variablenwert einer Ausgabezeile.

Benutzerdatei oder Bibliothekselement

Wird die Ausgabe in eine Benutzerdatei oder in ein Bibliothekselement umgelenkt, wird jeder Variableninhalt zu einem Datensatz.

Für Dateien kann der Benutzer festlegen, ob sie überschrieben oder erweitert werden sollen. „Überschreiben“ bedeutet: Der Inhalt der Datei wird gelöscht. Der erste ausgegebene Variableninhalt wird zum ersten Datensatz der Datei. „Erweitern“ bedeutet, dass die neuen Datensätze an das Ende der Datei angehängt werden.

Listenvariable

Die Ausgabe des Variableninhalts kann in ein Variablenaggregat vom Typ Liste umgelenkt werden. Dabei darf diese Listenvariable nicht gleichzeitig Quelle und Ziel der Ausgabe sein.

Wenn die Listenvariable noch nicht existiert, wird sie implizit deklariert, und zwar mit dem Datentyp *STRING und SCOPE = *CURRENT.

Wenn die Listenvariable existiert, hängt es von der Vereinbarung im Zusatzoperanden WRITE-MODE ab, ob sie überschrieben oder erweitert wird.

„Überschreiben“ bedeutet: Der alte Inhalt der Listenelemente wird gelöscht und den Listenelementen der Reihe nach der neue Inhalt zugewiesen. Dabei wird jeweils ein ausgegebener Variablenwert einem Listenelement zugewiesen.

„Erweitern“ bedeutet, dass die Listenvariable dynamisch erweitert wird. Für jeden ausgegebenen Variablenwert wird ein neues Listenelement an die Liste angehängt.

6.3.4.3 Ausgabe von Strukturlayouts

Das Kommando SHOW-STRUCTURE-LAYOUT ist ähnlich aufgebaut wie das Kommando SHOW-VARIABLE. Es gibt allerdings nicht den Inhalt von Variablen aus, sondern zeigt den Aufbau von Strukturlayouts. Es bezieht sich also auf den Aufbau explizit deklarierter statischer Strukturlayouts.

Strukturlayouts können nur mit SHOW-STRUCTURE-LAYOUT angezeigt werden. Da Elementen von Strukturlayouts keine Werte zugewiesen werden können, zeigt SHOW-STRUCTURE-LAYOUT nur die im angegebenen Layout deklarierten Elementnamen.

Beim Operanden SCOPE ist zu berücksichtigen, dass taskglobale Strukturlayouts immer sichtbar sind.

Ansonsten gelten für SHOW-STRUCTURE-LAYOUT die gleichen Aussagen wie für SHOW-VARIABLE beschrieben.

6.3.5 SDF-Kommando-Strings zu S-Variablen konvertieren und umgekehrt

SDF-Kommando-Strings können in S-Variablen vom Typ Struktur oder Liste gemäß ganz bestimmter Regeln konvertiert werden. Genauso ist es umgekehrt möglich, S-Variablen vom Typ Struktur oder Liste in SDF-Strings zu konvertieren.

Die Konvertierung eines SDF-Kommando-Strings in eine S-Variable wird mit dem Operanden *STRING-TO-VARIABLE(...) im Kommando SET-VARIABLE eingestellt (siehe Seite 753), die umgekehrte Konvertierung mit der vordefinierten Funktion VARIABLE-TO-STRING( ) (siehe Seite 537).

Hinweis

Die Angabe von Stellungsoperanden bei SET-VARIABLE ist für die Konvertierung nicht möglich. Sie führt zu Fehlermeldungen.

Konvertierungsregeln

SDF-Kommando-Strings zu S-Variablen konvertieren

1. Operanden werden in Strukturelemente konvertiert.
2. Einfache Operandenwerte werden in einfache Strukturelement-Werte konvertiert.
3. SDF-Listen werden in Listenvariablen konvertiert.
4. Die Konvertierung erfolgt unabhängig von SDF-Datentypen.
   Standardmäßig (VALUE-TYPE = *STD) gilt Folgendes: Wenn im Eingabe-String ein Integer-Wert gefunden wird, wird für die Variable als Datentyp Integer abgespeichert. Wenn im String TRUE oder FALSE gefunden wird (in Groß- oder Kleinbuchstaben), wird für die Variable als Datentyp Boolean abgespeichert. In allen anderen Fällen wird für die Variable als Datentyp String abgespeichert.
   Bei Angabe VALUE-TYPE = *STRING wird wertunabhängig für die Variable als Datentyp String abgespeichert.
5. Kommando-/Anweisungsnamen werden in Werte von Elementen mit dem reservierten Namen SYSOPER konvertiert.
6. Struktureinleitende Werte werden in Werte von Elementen mit dem reservierten Namen SYSSTRUC konvertiert.
7. Operanden in SDF-Strukturen werden in Strukturelement-Namen der 2. Ordnung konvertiert. Diese Elemente werden an den Element-Namen angehängt, der aus der Konvertierung des Operanden resultiert, der wiederum die Struktur bezeichnet.

S-Variablen zu SDF-Kommando-Strings konvertieren

S-Variablen vom Typ String werden in Zeichenketten ähnlich der &-Ersetzung konvertiert. Integer-Werte werden automatisch in Strings konvertiert. Boolean-Werte werden in die String-Werte TRUE bzw. FALSE konvertiert.

So muss eine S-Variable folgendermaßen initialisiert werden, wenn der Wert bei der Konvertierung einer S-Variablen zu einem SDF-String ein Wert vom Typ C-String ist:

```
/SET-VARIABLE DATA.OPER = 'C''mychain'''
```

Damit wird der String-Wert `'C''mychain'''` in der Variable DATA.OPER gespeichert und bei der Konvertierung in folgende SDF-Syntax umgewandelt:

```
OPER = C'mychain'
```

Das führt dazu, dass der Wert `C'mychain'` mit dem SDF-Datentyp <C-String> gemäß dem Operanden OPER spezifiziert wird.

Zusammenfassung

SDF-Syntax-String	S-Variablen-Aggregat (Name DATA)
OPER = value	DATA.OPER = 'value'
operation oper1 = val1	DATA.SYSOPER = 'operation' DATA.OPER1 = 'val1'
oper = struc (oper1 = val1)	DATA.OPER.SYSSTRUC = 'struc' DATA.OPER.OPER1 = 'val1'
oper = (val1,val2,val3)	DATA.OPER#1 = 'val1' DATA.OPER#2 = 'val2' DATA.OPER#3 = 'val3'

Ausnahmen

Nur die externe Form von SDF-Strings wird nach diesen Regeln zerlegt. Die Basisdaten, die verarbeitet werden, sind die Zeichenketten. Weder semantische Informationen noch interne SDF-Syntax-Deskriptoren werden konvertiert in S-Variable gespeichert: z.B. wenn OPER = A (OP1=VAL1) in eine S-Variable gespeichert wird, wird darin nicht angezeigt, ob A ein Schlüsselwort (keyword), ein Name (name) oder ein Dateiname (filename) ist.

Einschränkungen bei der String-Eingabe

Der SDF-String FROM=(file,*LIB(LIB=lib,EL=elem)) kann nicht in eine S-Variable übersetzt werden, weil SDF-P keine heterogenen Listen unterstützt.

Beispiele (für Einschränkungen)

Beispiel 1

FROM=(file,*LIB(LIB=lib,EL=elem)) wird konvertiert in:

```
A.FROM#1.SYSSTRUC = 'file'
A.FROM#2.SYSSTRUC = '*LIB'
A.FROM#2.LIB = 'lib'
A.FROM#2.EL = 'elem'
```

Solche Variablenaggregate sind möglich für Listen dynamischer Strukturen. Jedoch muss das erste Listenelement als SYSSTRUC-Element konvertiert werden, das nicht übereinstimmt mit der aktuellen Struktur des SDF-Strings. A.FROM#1 = 'file' ist nicht möglich.

Beispiel 2

OP = (a,b,c) wird konvertiert in:

```
A.OP#1 = 'a'
A.OP#2 = 'b'
A.OP#3 = 'c'
```

Dagegen wird OP = (a,b,c(OPR=d)) konvertiert in:

```
A.OP#1.SYSSTRUC = 'a'
A.OP#2.SYSSTRUC = 'b'
A.OP#3.SYSSTRUC = 'c'
A.OP#3.OPR = 'd'
```

Hinweis

Die Konvertierung eines einzelnen Wertes ist von der Struktur des Eingabe-Strings abhängig und kann deshalb zurückgewiesen werden (außer bei VALUE-TYPE= *STRING).

Beispiel 3

Die SDF-Strings FCB-TYPE=ISAM und FCB-TYPE=ISAM(KEY-POS=5,KEY-LEN=8) führt zu zwei verschiedenen Strukturen in SDF-P:

```
DATA.FCB-TYPE = 'ISAM'
```

und

```
DATA.FCB-TYPE.SYSSTRUC = 'ISAM'
DATA.FCB-TYPE.KEY-LEN = 8
DATA.FCB-TYPE.KEY-POS = 5
```

Dabei können in S-Prozeduren die zwei Variablenstrukturen durch die vordefinierte Funktion VARIABLE-ATTRIBUTE(..., ATTRIBUTE = *TYPE) sortiert werden. So sind z.B. bei VARIABLE-ATTRIBUTE('DATA.FCB-TYPE', ATTRIBUTE = *TYPE) die Ergebnisse:

– *STRUCTURE, wenn der Operandenwert eine Struktur ist.
– in den anderen Fällen *ANY oder *STRING

Beispiel 4

Die Strings OPER=A(OP1=X,OP2=Y) und OPER=B(OP1=X,OP2=Y) erzeugen dieselben Strukturelemente, allerdings hat OP1 in beiden Fällen unterschiedliche Bedeutung:

```
DATA.OPER.SYSSTRUC = 'A'
DATA.OPER.OP1 = 'X'
DATA.OPER.OP2 = 'Y'

DATA.OPER.SYSSTRUC = 'B'
DATA.OPER.OP1 = 'X'
DATA.OPER.OP2 = 'Y'
```

6.3.6 S-Variablen und Prozedurparameter

Prozedurparameter sind im BS2000 Parameter, die beim Aufruf einer Prozedur vom Aufrufer an die Prozedur übergeben werden. Sie dienen dazu, Informationen aus einer Prozedur in eine andere zu übertragen.

Prozedurparameter gibt es sowohl in S- als auch in Nicht-S-Prozeduren.

In S-Prozeduren werden Prozedurparameter als S-Variablen angelegt und bearbeitet. Allerdings haben Prozedurparameter nicht die Eigenschaften, die man von „normalen“ S-Variablen kennt, sondern sie unterscheiden sich von ihnen durch ihre Funktion, den Ort und die Kommandos, mit denen sie deklariert werden, sowie dadurch, dass für sie nicht alle Variablenmerkmale gelten.

Die folgende Tabelle zeigt diese Unterschiede und Gemeinsamkeiten.

	S-Variable	**Prozedurparameter**
Ort der Deklaration	Prozedurrumpf	Prozedurkopf
Kommando für die Deklaration	DECLARE-VARIABLE DECLARE-ELEMENT	DECLARE-PARAMETER
Variablentyp	einfache Variable zusammengesetzte Variable	einfache Variable
zulässiger Datentyp	ANY STRING INTEGER BOOLEAN	ANY STRING INTEGER BOOLEAN
Geltungsbereich	explizit definierbar: Current Prozedur Task	implizit definierbar: Current
Behälter	als Variablenbehälter: S-Variable Jobvariable	kein Behälter

Während in Nicht-S-Prozeduren die Werte beim Prozeduraufruf über Prozedurparameter nur an die aufgerufene Prozedur übergeben werden können, kann in S-Prozeduren dazu über Prozedurparameter auf Variableninhalte übergeordneter Prozeduren zugegriffen werden. Voraussetzung dafür ist, dass die Prozedur entweder mit INCLUDE-PROCEDURE aufgerufen wird oder die betreffenden Variablen taskglobale Variablen sind. (Näheres siehe im Abschnitt „Geltungsbereich von Variablen“ auf Seite 162).

Die Verknüpfung von Variablen und Prozedurparameter geht so vonstatten, dass die Prozedurparameter im Kommando DECLARE-PARAMETER mit TRANSFER-TYPE = *BY-REFERENCE deklariert werden. Das bedeutet: Die Zeichenfolge, die im Prozedurparameter übergeben wird, wird als Variablenname interpretiert. In der aufgerufenen Prozedur wird dann auf die Variable der aufrufenden Prozedur zugegriffen. (Näheres dazu siehe Abschnitt „Prozedurparameter übergeben“ auf Seite 108.)

6.3.7 Jobvariablen und S-Variablen

Jobvariablen sind Bestandteil des kostenpflichtigen Softwareprodukts JV (Jobvariablen). Nur wenn JV geladen ist, ist Zugriff auf Jobvariablen möglich. (Das Jobvariablen-System und die Jobvariablen sind ausführlich beschrieben im Handbuch „Jobvariablen“ [5]).

Jobvariablen werden sowohl in Nicht-S- als auch in S-Prozeduren eingesetzt.

Sowohl in S- als auch in Nicht-S-Prozeduren werden Jobvariablen dazu verwendet, Stapelaufträge, das heißt im Hintergrund gestartete Prozeduren, zu synchronisieren.

Die Verknüpfung von Variablen und Jobvariablen wird bei der Variablendeklaration festgelegt (Operand CONTAINER). Die S-Variablen müssen vom Datentyp STRING sein und dürfen nicht länger als 256 Bytes sein.

Jobvariablen, die noch nicht explizit initialisiert sind, enthalten den Wert 256 X'EE', um eine Unterscheidung zum Leerstring (String der Länge 0) zu ermöglichen.

Jobvariablen werden wie Dateien über einen Katalogeintrag verwaltet. Sie können auch wie Dateien mit einem Kennwort geschützt werden. Damit auf solche kennwortgeschützten Jobvariablen zugegriffen werden kann, zum Beispiel bei der Definition einer Jobvariablen als Variablenbehälter, muss das Kennwort mit dem Kommando ADD-PASSWORD in die Kennwort-Tabelle des Auftrags eingetragen werden. Erst dann kann das SDF-P-Kommando aufgerufen werden, in dem auf die Jobvariable zugegriffen wird.

Eine wichtige Rolle spielen Jobvariablen auch bei der Ausdrucksersetzung. Diese ist detailliert im Abschnitt „Ausdrucksersetzung“ auf Seite 55 beschrieben.

Die Tabelle auf der folgenden Seite zeigt eine Gegenüberstellung von S-Variablen und Jobvariablen.

Hinweis

Ob S-Variablen initalisiert sind, kann mit einer vordefinierten Funktion abgefragt werden: IS-INITIALIZED( )

Merkmal	**S-Variable**	**Jobvariable (JV)**
Name SDF-Datentyp Länge Zeichenvorrat	 <composed-name> 1..255 A...Z, 0...9, #, @, -, .	 <filename> 1..54 A...Z, 0...9, $, #, @, -, .
zulässiger Datentyp (Variableninhalt)	STRING BOOLEAN INTEGER	STRING
max. Größe (=Feldlänge)	4096 Bytes (STRING)	256 Bytes
Variablenformen	einfache Variable zusammengesetzte Variable	Benutzer-JV Sonder-JV Monitor-JV
Geltungsbereich	Task Prozedur Include (Permanenter Behälter in Bibliothek)	Task (temporäre JV); JV-System (permanente JV)
Deklaration	implizit bei der Zuweisung explizit mit SDF-P-Kommando	explizit mit Kommando (Benutzer-JV)
Initialisierung	bei der Deklaration	-
Wertzuweisung	bei der Deklaration explizite Zuweisung	explizite Zuweisung (Benutzer-JV)
Löschen	automatisch bei Prozedur-bzw. Task-Ende explizit mit Kommando	automatisch bei Task-Ende (nur temporäre JV) explizit mit Kommando

7 S-Variablenströme

Mit S-Variablenströmen ist es möglich - neben den bildschirmorientierten SYSOUT-Ausgaben - in umfangreichen Maße strukturierte Informationen zu übertragen und in S-Variablen zu lenken oder einem Ausgabe-Server zur weiteren Verarbeitung zu übergeben (z.B. FHS zur Ausgabe in FHS-Masken).

7.1 Konzept der S-Variablenströme

Das Konzept der S-Variablenströme basiert auf der Interaktion von Client und Server. In Abwandlung des klassischen Client-Server-Modells bestehen dabei im SDF-P-Kontext folgende drei Verknüpfungen:

– Auf der Client-Seite werden vom Benutzer S-Variablen (die vom System oder mit DECLARE-VARIABLE oder mit SET-VARIABLE erzeugt wurden) zum Server geschickt und/oder die vom Server zurückgeschickten Variablen aufgenommen.

– Der Server (FHS oder SDF-P) verarbeitet je nach Anforderungen des Clients diese S-Variablen. So kann z.B. FHS aufgefordert werden, die vom Client erhaltenen S-Variablen auf Bildschirmmasken auszugeben und zu aktualisieren. Wartet der Client zudem auf Antwort, schickt FHS diese oder andere Variablen später wieder an den Client zurück.

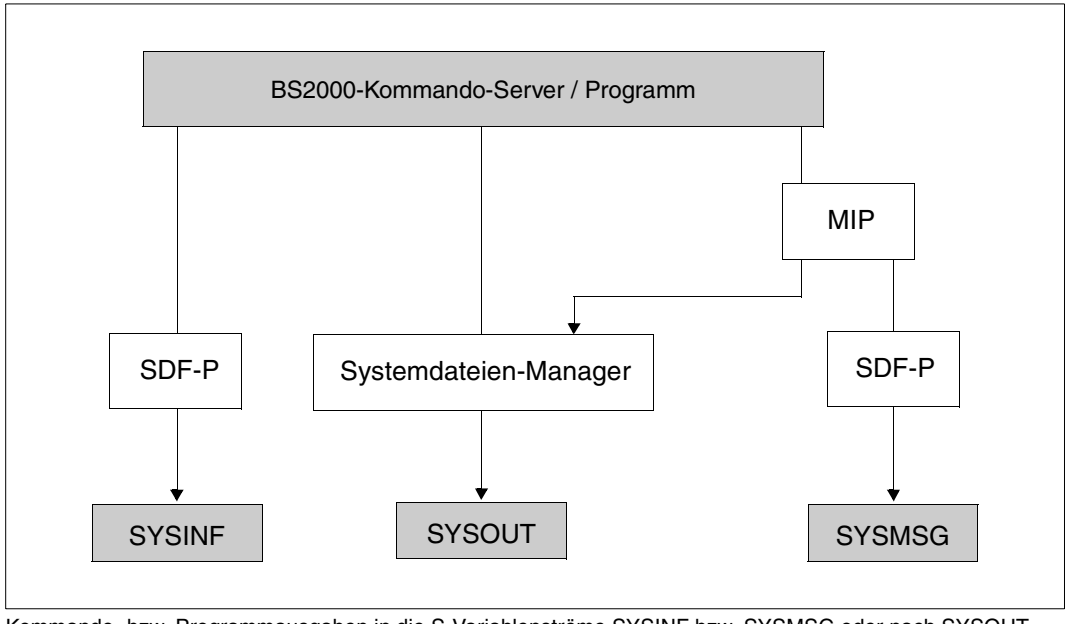
– SDF-P hat - neben der optionalen Aufgabe des Servers - die Rolle der Steuereinheit bzw. des Routers. So ist z.B. SDF-P dafür verantwortlich, dass die Server-Verbindung zum Client bei Laufzeit geknüpft wird. Diese Verbindung ist nicht starr, sondern kann dynamisch ausgewählt werden, um eine möglichst große Unabhängigkeit des Client-Codes vom Server zu besitzen.

7.1.1 Funktionsumfang

- Ausgabe von BS2000-SHOW-Kommandos in S-Variable
  Ausgaben von SHOW-Kommandos werden in S-Variablen gelenkt, die wiederum als Filter zwischen BS2000-Kommando-Server und Ausgabe-Server dienen: Die Variablenwerte können so z.B. vom Ausgabe-Server FHS in interaktive Bildschirmmasken zur komfortablen Benutzung eingefügt werden. (Siehe dazu auch im Handbuch „FHS“[19].)
- Wiederverwendung von Ausgabe-Informationen als Eingabe-Informationen für spätere Kommandos
  Die in S-Variablen abgelegten Ausgabedaten von Kommandos können als Eingabedaten für spätere Kommandos benutzt werden. Nicht nur einfache Variablen können so Werte einzelner Operanden ersetzen, sondern auch Variablen vom Typ Struktur können so Operandenstrukturen oder sogar ein ganzes Kommando bzw. eine ganze Anweisung ersetzen.
- Auswahl des Ausgabe-Servers zur Laufzeit
  Durch einen dynamischen Auswahlmechanismus für Anwendungen und S-Prozeduren braucht der Ausgabe-Server (z.B. SDF-P oder FHS) erst zur Laufzeit bestimmt zu werden.
- Variabler Einsatz von FHS
  FHS ist sowohl auf Programm- wie auch auf Kommandoebene als Server im SDF-P-Kontext verfügbar. Dazu können mit S-Prozeduren Anwendungen für FHS erstellt werden.
- Neudefinition der Aufgaben des Kommandos EXECUTE-CMD
  Während mit dem Kommando ASSIGN-STREAM die Grundzuweisungen erstellt werden, kann mithilfe des Kommandos EXECUTE-CMD der Standard-Variablenstrom SYSVAR dazu alternativ zugewiesen werden.
- Komplementäre oder alternative Ausgaben auf SYSOUT und in S-Variablen
  S-Variablen bzw. S-Variablenströme können alternativ oder komplementär zur Ausgabe nach SYSOUT gesteuert werden. Die Standardeinstellung bei ASSIGN-STREAM ist, dass beide komplementär ausgegeben werden.

7.1.2 Die S-Variablenströme SYSINF, SYSMSG und SYSVAR

Folgendes Schema zeigt sowohl die Ausgabe nach SYSOUT durch den Systemdateien-Manager als auch die Umlenkung der Ausgabedaten in die S-Variablenströme SYSINF und SYSMSG. Die Zusammenfassung der Ströme SYSINF und SYSMSG wird als SYSVAR bezeichnet.



Kommando- bzw. Programmausgaben in die S-Variablenströme SYSINF bzw. SYSMSG oder nach SYSOUT

SYSINF

Der S-Variablenstrom SYSINF enthält die strukturierte Ausgabe von (SHOW-)Kommandos und Programmen. Die Ausgabeformate sind im Handbuch „Kommandos, Band 6“ [4] zusammengestellt.

SYSMSG

Der S-Variablenstrom SYSMSG enthält die strukturierte Ausgabe von garantierten Meldungen. Die Ausgabe von Meldungen in S-Variable ist im Handbuch „MSGMAKER“ [21] ausführlich beschrieben.

SYSOUT

Der Ausgabestrom SYSOUT enthält im Normalfall alle Ausgaben der Kommando-Server und Dienstprogramme. Die Ausgabe ist bildschirmorientiert aufbereitet.

7.1.3 S-Variablenströme zuweisen

Im Normalfall weist man einen S-Variablenstrom mit dem Kommando ASSIGN-STREAM einem Ausgabeziel zu. Das Ausgabeziel kann ein Server oder eine S-Variable sein.

Mit dem Kommando EXECUTE-CMD wird implizit der S-Variablenstrom SYSVAR einer S-Variablen zugewiesen. Die Zuweisung ist aber nur temporär für das spezifizierte Kommando gültig.

Zuweisung mit ASSIGN-STREAM

Das Kommando ASSIGN-STREAM besitzt zwei Hauptoperanden: STREAM-NAME und TO.

Operand STREAM-NAME

Im Operanden STREAM-NAME wird der Name des S-Variablenstroms angegeben, der zugewiesen werden soll. Es stehen dazu drei vom System vorgegebene S-Variablenströme zur Verfügung, die alle das Präfix SYS besitzen: SYSINF, SYSMSG, SYSVAR. Man kann allerdings auch einen eigenen S-Variablenstrom bzw. einen eigenen Namen hier angeben. Eigene S-Variablenströme sind aber nur sinnvoll, wenn man z.B. FHS-Anwendungen benutzen will.

Operand TO

Im Operanden TO wird das Ausgabeziel bzw. der Server angegeben, der mit dem S-Variablenstrom verknüpft werden soll. Es gibt folgende Möglichkeiten:

– TO = *STD
*STD ist der Defaultwert. In Abhängigkeit von der Angabe bei STREAM-NAME gelten folgende Zuweisungen:

STREAM-NAME=	**TO=*STD**	**Übertragene Informationen**
SYSINF	SYSVAR	Strukt. Ausgaben von Kommandos und Programmen
SYSMSG	SYSVAR	Strukt. garantierte Meldungen
SYSVAR	*DUMMY	Strukt. Kommando- und Programm-Ausgaben bzw. strukt. Meldungen
<structured-name 1..20>	*DUMMY	Benutzer-Variablenstrom

Der Standard-Variablenstrom für Systemausgaben und garantierte Meldungen ist SYSVAR, dem wiederum standardmäßig *DUMMY zugewiesen ist. So koexistieren SYSOUT- und SYSVAR-Ausgaben. (Wenn kein SYSOUT gewünscht wird, muss das Kommando ASSIGN-SYSOUT TO-FILE=*DUMMY spezifiziert werden.)

– TO = <structured-name 1..20>
  Name eines (eigenen) Servers.

– TO = *DUMMY
  Mit *DUMMY wählt man keine Zuweisung.

– TO = *SAME-AS-CALLING-PROC
  Die Angabe *SAME-AS-CALLING-PROC bedeutet, dass die in der aufrufenden Prozedur spezifizierte Zuweisung weiterhin gültig ist.

– TO = *VARIABLE(...)
  Ausgabeziel ist die angegebene S-Variable. Indirekt ist SDF-P als Server zugewiesen. Es können Kontrollvariablen für den Datenaustausch mit dem Server spezifiziert werden. Alle angegebenen Variablen müssen als Listenvariablen vom Typ Struktur deklariert werden.
  Von den Kommandos TRANSMIT-BY-STREAM und ASSIGN-STREAM aus gesehen werden die Daten in folgender Reihenfolge abgearbeitet:

TRANSMIT-BY-STREAM	**Richtung**	**ASSIGN-STREAM**
(1) VARIABLE-NAME	⟶	VARIABLE-NAME
(2) CONTROL-VAR-NAME	⟶	CONTROL-VAR-NAME
(3) RETURN-VARIABLE-NAME	⟵	RETURN-VARIABLE-NAME
(4) RET-CONTROL-VAR-NAME	⟵	RET-CONTROL-VAR-NAME

  Tritt während einer dieser Operationen ein Fehler auf, so werden diese - wie alle folgenden - Operationen beendet. Die Übertragung wird dann mit dem Status zum Fehlerzeitpunkt abgebrochen.

– TO = *SERVER(...)
  Name eines Servers, z.B. FHS. Dem Server können zusätzliche Informationen (Name der Formatbibliothek für FHS) übergeben werden.

Hinweise

– Die verschiedenen Informationen von SYSINF und SYSMSG können auch bei der Zuweisung SYSVAR getrennt weiterverarbeitet werden.
– Angaben von Systemdateien (SYSDTA, SYSCMD, SYSOUT, SYSLST, SYSOPT, SYSIPT) bei STREAM-NAME werden abgewiesen.
– Sind die in *VARIABLE(...) angegebenen S-Variablen im Moment der ASSIGN-STREAM-Zuweisung unvollständig deklariert, wird diese Zuweisung abgewiesen.
– Sind die Variablen nach der ASSIGN-STREAM-Zuweisung unvollständig deklariert, weil sie z.B. zwischenzeitlich mit DELETE-VARIABLE bearbeitet wurden, wird die nächste Übertragung mit einer Warnung abgewiesen und der Variablenstrom wird durch SDF-P auf *DUMMY gesetzt.

– Wenn dieselbe S-Variable zwei unterschiedlichen Variablenströmen zugewiesen ist, werden die Daten beider Variablenströme in der Reihenfolge ihrer Übertragung abgearbeitet.
– S-Variable können zwischen zwei Übertragungen verändert werden. Die nächste Übertragung berücksichtigt diese Veränderung.

Beispiel

```
/DECLARE-VARIABLE OPS-VAR(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
/ASSIGN-STREAM SYSINF,TO=*VARIABLE(OPS-VAR)
/ASSIGN-SYSOUT TO=#ERROR-SYSOUT
```

7.1.4 S-Variablen mittels S-Variablenströmen übertragen

Die Übertragung der S-Variablen wird in S-Prozeduren mit dem Kommando TRANSMIT-BY-STREAM geregelt. Dieses Kommando steuert von Client-Seite die Variablenübermittlung über einen ausgewählten S-Variablenstrom zum und/oder vom angegebenen Server.

Man gibt dabei im Operanden STREAM-NAME den ausgewählten Variablenstrom an.

In den restlichen Operanden VARIABLE-NAME, RETURN-VARIABLE-NAME, CONTROL-VAR-NAME und RET-CONTROL-VAR-NAME wird in Entsprechung zur Einstellung bei ASSIGN-STREAM ...,TO=*VARIABLE( ) bestimmt, welche Variablen zum bzw. vom Server übertragen werden. D.h. analog zu ASSIGN-STREAM müssen hier die angegebenen Variablen als Strukturen deklariert sein. Genauso haben bei der Reihenfolge der abzuarbeitenden Daten auch hier die bei VARIABLE-NAME und CONTROL-VAR-NAME angegebenen Variablen Vorrang vor den restlichen.

Weitere Hinweise (vor allem zum Standard Header) siehe Kommando TRANSMIT-BY-STREAM, Seite 800.

Beispiel

```
/DECLARE-VARIABLE A1(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
/DECLARE-VARIABLE A2(TYPE=*STRUCTURE)
/ASSIGN-STREAM SYSINF,TO=*VARIABLE(A1)
/TRANSMIT-BY-STREAM SYSINF, VARIABLE=A2, RETURN-VARIABLE=*NONE
```

7.1.5 Zuweisungen von S-Variablenströmen ausgeben

Mit dem Kommando SHOW-STREAM-ASSIGNMENT kann man sich die aktuelle Zuweisung des oder der angegebenen S-Variablenströme anzeigen lassen (Näheres siehe Seite 762).

Im Operanden STREAM-NAME gibt man den oder die Variablenströme an, die angezeigt werden sollen. Bei Angabe des Werts *STD-STREAMS werden alle Standard-Variablenströme ausgegeben, d.h. alle S-Variablenströme mit dem Prefix „SYS“.

Im Operanden INFORMATION können Informationen über den zugewiesenen Server ausgegeben werden. Mit der Voreinstelllung *CURRENT-ASSIGNMENT wird ausgegeben, was im Operand TO bei ASSIGN-STREAM angegeben wurde. Bei Angabe von *FINAL-DESTINATION wird der aktuelle Server-Name ausgegeben.

Im Operanden OUTPUT gibt man an, wohin die Ausgabe geschickt werden soll. Standardmäßig wird sie nach SYSOUT und parallel dazu zu der Variablen geschickt, die im Kommando ASSIGN-STREAM angegeben wurde. Bei Angabe von *SYSLST erfolgt nur eine Ausgabe auf SYSLST und nicht in eine Variable.

Beispiel

Eingabe

```
/DECLARE-VARIABLE VAR1(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
/ASSIGN-STREAM SYSINF,TO=*VARIABLE(VAR1)
/SHOW-STREAM-ASSIGNMENT SYSINF
```

Ausgabe

```
STREAM-NAME = SYSINF
 ASSIGN-LEVEL = 0
 DESTINATION = *VARIABLE
    VARIABLE-NAME = VAR1
        VAR-MODE = *EXTEND
    RETURN-VARIABLE-NAME = *NONE
    CONTROL-VAR-NAME = *NONE
    RET-CONTROL-VAR-NAME = *NONE
```

7.1.6 S-Variablenströme löschen

Mit dem Kommando DELETE-STREAM können S-Variablenströme gelöscht werden. Dabei können im Operanden STREAM-NAME die zu löschenden Variablenströme entweder als Suchmuster oder explizit in einer Liste angegeben werden.

Löschen bedeutet, dass der Variablenstrom mit *DUMMY belegt wird und nicht mehr als Zuweisungsziel (ASSIGN-STREAM TO=...) oder Übertragungsziel (TRANSMIT-BY-STREAM) verwendet werden kann.

Es können nur S-Variablenströme gelöscht werden, die sich auf der obersten Ebene (außerhalb von Prozeduren) befinden oder in davon aufgerufenen Prozeduren, wenn SYSTEM-FILE-CONTEXT=*SAME-AS-CALLER eingestellt ist.

Variablenströme mit reservierten Namen können nicht gelöscht werden.

In Prozeduren benutzte Variablenströme mit nicht-reservierten Namen werden implizit gelöscht, wenn die Prozedur verlassen wird und im Kommando SET-PROCEDURE-OPTIONS nicht SYSTEM-FILE-CONTEXT=*SAME-AS-CALLER eingestellt ist.

7.2 Strukturierte Ausgaben in S-Variablen

SHOW-Kommandos können ihre Informationen in zusammengesetzte S-Variablen vom Typ Struktur ausgeben. Durch die strukturierte Ausgabe kann der Benutzer direkt auf Einzelinformationen zugreifen. Jedes SHOW-Kommando mit dieser Funktionalität gibt den Aufbau der Struktur vor:

– Für ein im SHOW-Kommando angegebenes Objekt (z.B Datei, Gerät) wird eine Struktur definiert. Werden mehrere Objekte angegeben (z.B. als Musterzeichenfolge), wird eine Liste von Strukturen angelegt.

– Für jede Einzelinformation zu diesem Objekt wird eine S-Variable als Element dieser Struktur definiert und die Einzelinformation als Inhalt zugewiesen.

– Lassen sich Informationen zu einem Objekt hierarchisch weiter untergliedern, wird für jede Hierarchie eine zusammengesetzte S-Variable als Element der übergeordneten Struktur definiert. Eine hierarchisch untergeordnete S-Variable kann dabei eine einfache S-Variable, eine Struktur oder Liste einfacher S-Variablen und/oder Strukturen sein.

– Die Namen der Elemente sind für das jeweilige SHOW-Kommando vorgegeben. Sie entsprechen, soweit möglich, korrespondieren Operandennamen bzw. einer eindeutigen Abkürzung. Innerhalb aller SHOW-Kommandos werden Elemente, die gleiche Informationen enthalten, konsistent benannt. Die Namen der S-Variablen sind für jedes SHOW-Kommando vorgegeben und für die Folgeversionen garantiert.

– Die Inhalte der S-Variablen (die Einzelinformationen) entsprechen, soweit möglich, korrespondieren Operandenwerten bzw. eindeutigen Abkürzungen.

– Die S-Variablen besitzen einen festgelegten Typ: String, Integer oder Boolean.

Die Ausgabestruktur kann der Kommandobeschreibung im jeweiligen Produkthandbuch entnommen werden. Für die SHOW-Kommandos der BS2000/OSD-BC sind die Ausgabestrukturen im Handbuch „Kommandos, Band 6“ [4] zusammengestellt.

7.2.1 Vorgehensweise

1. S-Variable deklarieren

Der Benutzer deklariert eine Listenvariable vom Typ Struktur. Die Struktur sollte dynamisch erweiterbar sein (Voreinstellung).

```
/DECLARE-VARIABLE NAME=USERVAR(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
```

Wird die Struktur dagegen nur statisch angelegt, kann der Benutzer nur Informationsteile erhalten, für die er auch Strukturelemente explizit mit dem jeweils festgelegten Namen deklariert hat.

2. Strukturierte Ausgabe erzeugen

Kommando EXECUTE-CMD

Für die strukturierte Ausgabe *eines* Kommandos spezifiziert der Benutzer im Kommando EXECUTE-CMD ein SHOW-Kommando und gibt an, dass die strukturierte Systemausgabe in die bereits deklarierte S-Variable gelenkt werden soll.

```
/EXECUTE-CMD CMD =( SHOW-SYNTAX-VERSIONS SOFTWARE-UNIT-NAME=(JV, LMS) ), -
             /STRUCTURED-OUTPUT = USERVAR ,TEXT-OUTPUT = *NONE
```

Die Ausgabe nach SYSOUT wird dabei mit TEXT-OUTPUT=*NONE unterdrückt.

Wenn die Ausgabe in eine Variable mit statischer Struktur erfolgt, dann wird nur den bereits vorhandenen Elementen der Struktur ein Wert zugewiesen.
Ist für das Element kein Wert vorhanden, werden folgende typabhängige Defaultwerte eingesetzt:

```
INTEGER: 0
STRING:  '' (Leerstring)
BOOLEAN: FALSE
```

Wenn keiner der Elementnamen mit den definierten Namen übereinstimmt, findet weder eine Zuweisung statt noch wird eine Fehlermeldung oder Warnung ausgegeben. Die Variable ist in diesem Fall nach dem EXECUTE-CMD leer, wenn der Operand WRITE-MODE=*REPLACE im Kommando DECLARE-VARIABLE angegeben wurde (vor der Kommandoausführung wird ein implizites FREE-VARIABLE abgesetzt).

Im Operanden MSG-STRUCTURE-OUTPUT kann angegeben werden, dass garantierte Meldungen für die angegebenen Kommandos in eine S-Variable ausgegeben werden.

Da das Kommando EXECUTE-CMD die mit dem Kommando ASSIGN-STREAM getroffenen Zuweisungen für SYSINF und SYSMSG temporär überschreibt, erfolgt keine gleichzeitige Ausgabe für den dort spezifizierten Server.

Kommando ASSIGN-STREAM

Mit ASSIGN-STREAM kann der Benutzer die S-Variablenströme SYSINF, SYSMSG oder SYSVAR der deklarierten Variablen zuordnen. Die strukturierten Ausgaben der SHOW-Kommandos werden über SYSINF, garantierte Meldungen über SYSMSG übertragen. Während der Zuordnung wird für jedes abgesetzte Kommando, das strukturierte Ausgaben unterstützt, die S-Variable entsprechend erweitert. Die Zuordnung gilt solange, bis sie explizit aufgehoben oder die Prozedur beendet wird.

3. Inhalt der S-Variable ausgeben

```
/SHOW-VARIABLE  USERVAR
```

```
/show-var uservar,list-index-number=*yes
USERVAR#1.F-NAME = :1OSH:$TSOS.SYSSDF.JV.140                    ┐
USERVAR#1.TYPE = *SYS                                           │
USERVAR#1.SW-UNIT#1.NAME = JV                                   │
USERVAR#1.SW-UNIT#1.VERSION = 14.0C100                          │
USERVAR#1.SW-UNIT#1.COMPONENT#1.NAME = JVS    } (4)  } (3)      │ (1)
USERVAR#1.SW-UNIT#1.COMPONENT#1.VERSION = 42C }              │
USERVAR#1.SW-UNIT#1.COMPONENT#2.NAME = CJC    } (5)          │
USERVAR#1.SW-UNIT#1.COMPONENT#2.VERSION = 20C }              ┘
USERVAR#2.F-NAME = :1OSH:$TSOS.SYSSDF.LMS.033                   ┐
USERVAR#2.TYPE = *SYS                                           │
USERVAR#2.SW-UNIT#1.NAME = LMS                                  │ (2)
USERVAR#2.SW-UNIT#1.VERSION = 03.3B300                          │
USERVAR#2.SW-UNIT#1.COMPONENT#1.NAME = LMS                      │
USERVAR#2.SW-UNIT#1.COMPONENT#1.VERSION = 033                   ┘
/
```

Erläuterung der Ausgabe:

Die vom Benutzer definierte S-Variable (hier USERVAR) enthält die Gesamtausgabe. Die Zeichenfolge „#i“ zeigt die Nummer des jeweiligen Listenelements an (im SHOW-VARIABLE-Kommando wurde LIST-INDEX-NUMBER=*YES angegeben). Die S-Variable USERVAR enthält 2 Elemente:
Einmal die Struktur für Informationen über die Softwareeinheit JV, in der Ausgabe mit (1) gekennzeichnet, und einmal die Struktur für Informationen über die Softwareeinheit LMS, in der Ausgabe mit (2) gekennzeichnet.
Die Information für eine Softwareeinheit besteht aus den Elementen F-NAME, TYPE und SW-UNIT, wobei SW-UNIT selbst wieder eine Liste enthalten kann. Im Falle der Softwareeinheit JV enthält SW-UNIT ein Listenelement, nämlich die Struktur, die aus aus den Elementen NAME, VERSION und COMPONENT gebildet wird (in der Ausgabe mit (3) gekennzeichnet).
COMPONENT kann wiederum eine Liste sein: Die Softwareeinheit JV besteht aus 2 Komponenten, d.h. COMPONENT enthält 2 Listenelemente. Jedes Listenelement ist eine Struktur mit den Elementen NAME und VERSION. In der Ausgabe sind die Komponenten CJC und JVS mit (4) bzw. (5) gekennzeichnet.

4. Auf Einzelinformationen zugreifen

Auf einzelne Informationen kann der Benutzer über den Namen der S-Variablen zugreifen. Der anzugebende Name setzt sich wie folgt zusammen:

uservar#i.element — Dabei bedeuten:

uservar	Name der Struktur, die vom Benutzer deklariert wurde
#i	i-tes Element der Liste Für i=1 kann „i“ entfallen, d.h. es wird nur „#“ angegeben.
Punkt	Begrenzer im Namen von zusammengesetzen S-Variablen.
element	vordefinierter Name des Strukturelements element kann wiederum eine zusammengesetzte Variablen sein: z.B. uservar#.SW-UNIT#.NAME

Der Benutzer kann sich den Inhalt z.B. mit „SHOW-VARIABLE uservar#.element“ anzeigen lassen oder mittels Variablenersetzung weiterverwenden:

```
/show-var (uservar#.sw-unit#.component#1.name,
           uservar#.sw-unit#.component#2.name),list-index-number=*yes
USERVAR#1.SW-UNIT#1.COMPONENT#1.NAME = JVS
USERVAR#1.SW-UNIT#1.COMPONENT#2.NAME = CJC

/write-text '*** SW-UNIT &(uservar#2.sw-unit#.name) -
/mit der Version &(uservar#2.sw-unit#.version) ***'
*** SW-UNIT LMS mit der Version 03.3B300 ***

/show-var uservar#2,list-index-number=*yes
USERVAR#2.F-NAME = :1OSH:$TSOS.SYSSDF.LMS.033
USERVAR#2.TYPE = *SYS
USERVAR#2.SW-UNIT#1.NAME = LMS
USERVAR#2.SW-UNIT#1.VERSION = 03.3B300
USERVAR#2.SW-UNIT#1.COMPONENT#1.NAME = LMS
USERVAR#2.SW-UNIT#1.COMPONENT#1.VERSION = 033
```

7.2.2 Beispiel

```
/declare-var var-name=out-1(type=*structure),multiple-elements=*list
/execute-cmd cmd=(show-file-attr file-name=job*,inf=*par(alloc=*yes)),-
/                 text-output=*none,structure-output=out-1

/show-var out-1,inf=*par(value=*c-lit,list-index-number=*yes)
OUT-1#1.F-NAME = ':2OSG:$USER1.JOBA'
OUT-1#1.CAT-ID = '2OSG'
OUT-1#1.USER-ID = 'USER1'
OUT-1#1.SHORT-F-NAME = 'JOBA'
OUT-1#1.F-SIZE = 3
OUT-1#1.SUP = '*PUB'
OUT-1#1.HIGHEST-USED-PAGES = 1
OUT-1#1.SEC-ALLOC = 24
OUT-1#1.BLOCK-COUNT = 0
OUT-1#1.EXT#1.VOL = 'GVS2.2'
OUT-1#1.EXT#1.DEV = 'D3435'
OUT-1#1.EXT#1.NUM-OF-EXT = 1
*END-OF-VAR
OUT-1#1.NUM-OF-EXT = 1
*END-OF-VAR
OUT-1#2.F-NAME = ':2OSG:$USER1.JOBB'
OUT-1#2.CAT-ID = '2OSG'
OUT-1#2.USER-ID = 'USER1'
OUT-1#2.SHORT-F-NAME = 'JOBB'
OUT-1#2.F-SIZE = 3
OUT-1#2.SUP = '*PUB'
OUT-1#2.HIGHEST-USED-PAGES = 1
OUT-1#2.SEC-ALLOC = 24
OUT-1#2.BLOCK-COUNT = 0
OUT-1#2.EXT#1.VOL = 'GVS2.3'
OUT-1#2.EXT#1.DEV = 'D3435'
OUT-1#2.EXT#1.NUM-OF-EXT = 1
*END-OF-VAR
OUT-1#2.NUM-OF-EXT = 1
*END-OF-VAR
OUT-1#3.F-NAME = ':2OSG:$USER1.JOBC'
OUT-1#3.CAT-ID = '2OSG'
OUT-1#3.USER-ID = 'USER1'
OUT-1#3.SHORT-F-NAME = 'JOBC'
OUT-1#3.F-SIZE = 3
OUT-1#3.SUP = '*PUB'
OUT-1#3.HIGHEST-USED-PAGES = 1
OUT-1#3.SEC-ALLOC = 24
OUT-1#3.BLOCK-COUNT = 0
OUT-1#3.EXT#1.VOL = 'GVS2.0'
OUT-1#3.EXT#1.DEV = 'D3435'
```

```
OUT-1#3.EXT#1.NUM-OF-EXT = 1
*END-OF-VAR
OUT-1#3.NUM-OF-EXT = 1
*END-OF-VAR
```

Erläuterung:

Die vom Benutzer definierte Listenvariable VAR enthält drei Elemente.

Mit `inf=*par(alloc=*yes)` werden für die ausgewählten Dateien alle Dateimerkmale ausgegeben, die die Speicherplatzbelegung betreffen. Die Dateimerkmale F-NAME, CAT-ID, USER-ID,....,EXT bilden die Elemente der Struktur. Das Element EXT ist wiederum eine Liste, die aus den Elementen VOL, DEV und NUM-OF-EXT besteht.

Lassen sich die Informationen zu einem Objekt hierarchisch weiter untergliedern (siehe Element EXT), wird für jede Hierarchie eine zusammengesetzte S-Variable als Element der übergeordneten Struktur definiert. Eine hierarchisch untergeordnete S-Variable kann dabei eine einfache S-Variable (siehe VOL, DEV, NUM-OF-EXT), eine Struktur oder eine Liste einfacher S-Variablen und/oder Strukturen sein.

Die Namen der Listenelemente (z. B. F-NAME, F-SIZE) sind für das jeweilige SHOW-Kommando vorgegeben. Sie schließen sich an den vom Anwender deklarierten S-Variablennamen an. Ist die Information auf dieser Ebene weiter hierarchisch untergliedert, schließen sich weitere Namen, durch einen Punkt getrennt, an.

7.3 FHS als Ausgabe-Server

Ab FHS V8.1 kann FHS die Rolle des Ausgabe-Servers für S-Variablenströme übernehmen. Dazu können mithilfe von S-Prozeduren Anwendungen für FHS benutzt und gesteuert werden (siehe dazu auch Handbuch „FHS“[19]).

Beide Funktionen werden in den folgenden zwei Abschnitten beschrieben.

7.3.1 FHS als Ausgabe-Server benützen

Als Ausgabe-Server ist FHS sowohl auf Programm- wie auch auf Kommandoebene im SDF-P-Kontext verfügbar. D.h.:

– FHS-PRIV kann einerseits durch ein TU-Programm unter Benutzung eines TRANSVV SVC aufgerufen werden.
– FHS-PRIV kann andererseits auf Kommandoebene durch TRANSMIT-BY-STREAM aufgerufen werden.

Voraussetzung für beide Möglichkeiten ist, dass der Variablenstrom, der vom Programm benutzt wird, im Operanden SERVER des Kommandos ASSIGN-STREAM FHS zugewiesen wird.

Wenn durch ein TRANSMIT-BY-STREAM-Kommando Informationen bei FHS eingehen, stellt FHS Anzeigedienste zur Verfügung, die den Diensten ähnlich sind, die FHS im TU-Modus durch die Schnittstellen DISPLAY, ADDPOP and REMPOP anbietet (siehe dazu Handbuch „FHS“ [19]).

7.3.1.1 Strukturlayout und Initialisierung

Wird FHS als Ausgabe-Server benützt, muss für FHS das Strukturlayout definiert und bestimmte Werte initialisiert werden. Dazu wird in der (ausgelieferten) Bibliothek $TSOS.SYSPRC.FHS.<version> eine S-Prozedur (Element SYSFHS-CONTROL) mitgeliefert, die für FHS V8.3 folgendermaßen aussieht:

```
/set-procedure-options caller=include
/begin-parameter-declaration
/   declare-parameter -
/   "------------ std param ---------------------------------*"-
/               (PREFIX       (type=string,init='SYSFHS-') -
/               ,INCLUDE-FORM (type=string,init='LAYOUT') "/initialize" -
/               ,VARIABLE-NAME(type=string,init='') -
/   "------------ include specific param -------------------*"-
/  " action variables " -
/               ,SERVICE      (type=string,init='*DISPLAY') -
/               ,DIAGINFO     (type=string,init='*NO') -
/               ,POP-LOCATION (type=string,init='*NONE') -
```

```
/               ,POP-LOC-IND  (type=integer,init=0) -
/               ,ROW          (type=integer,init=2) -
/               ,COLUMN       (type=integer,init=2) -
/  " resource variables " -
/               ,RESOURCE      (type=string,init='*SAME') -
/               ,MESSAGE-ID    (type=string,init='*NONE') -
/               ,MESSAGE-FIELD (type=string,init='*NONE') -
/               ,MSG-FIELD-IND (type=integer,init=0) -
/  " panel variables " -
/               ,CURSOR-OUTPUT-INDEX (type=integer,init=0) -
/               ,CURSOR-OUTPUT       (type=string,init='*NONE')  -
/               ,CURSOR-OUTPUT-POS   (type=integer,init=0) -
/               ,LOCK                (type=string,init='*NO') -
/               ,ALARM               (type=string,init='*NO') -
/               ,HARDCOPY            (type=string,init='*NO') -
/               ,AUTOTAB             (type=string,init='*YES')  -
/               ,MANDATORY           (type=string,init='*NO')  -
/               ,REFRESH             (type=string,init='*NO')  -
/               ,ACK                 (type=string,init='*NO')  -
/               ,KEYLOCK             (type=string,init='*NONE')  -
/  " field attributes " -
/               ,ATTR-LIST    (type=integer,init=0) -
/                " number of list elements to reset       "-
/               ,FIELD     (type=string,init='*CURSOR') -
/               ,FIELD-IND (type=string,init='0') -
/               ,TYPE      (type=string,init='*UNCHANGED') -
/               ,HILITE    (type=string,init='*UNCHANGED') -
/               ,INTENSITY (type=string,init='*UNCHANGED') -
/               ,COLOR     (type=string,init='*UNCHANGED') -
/               ,OUTPUT    (type=string,init='*UNCHANGED') -
/   " input information " -
/               ,COMMAND               (type=string,init='') -
/               ,FHS-VERSION           (type=string,init='') -
/               ,CURSOR-INPUT          (type=string,init='') -
/               ,CURSOR-INPUT-INDEX    (type=integer,init=0) -
/               ,CURSOR-INPUT-POS      (type=integer,init=0) -
/               )
/end-parameter-declaration
/
/if (upper-case(INCLUDE-FORM) == 'LAYOUT')
/
/begin-structure ATTR,scope=proc
/   declare-element -
/                   (FIELD     (type=string) -
/                   ,FIELD-IND (type=string) -
/                   ,TYPE      (type=string) -
/                   ,HILITE    (type=string) -
/                   ,INTENSITY (type=string) -
```

```
/                    ,COLOR     (type=string) -
/                    ,OUTPUT    (type=string) -
/                    )
/end-structure
/
/begin-structure name=&PREFIX.LAYOUT,scope=proc
/  declare-element STD-HEADER(type=structure(&PREFIX.FHDR))
/  declare-element -
/  " action variables " -
/                   (SERVICE      (type=string) -
/                   ,DIAGINFO     (type=string) -
/                   ,POP-LOCATION (type=string) -
/                   ,POP-LOC-IND  (type=integer) -
/                   ,ROW          (type=integer) -
/                   ,COLUMN       (type=integer) -
/  " resource variables " -
/                   ,RESOURCE         (type=string) -
/                   ,MESSAGE-ID       (type=string) -
/                   ,MESSAGE-FIELD    (type=string) -
/                   ,MSG-FIELD-IND    (type=integer) -
/  " panel variables " -
/                   ,CURSOR-OUTPUT-INDEX (type=integer) -
/                   ,CURSOR-OUTPUT       (type=string)  -
/                   ,CURSOR-OUTPUT-POS   (type=integer) -
/                   ,LOCK                (type=string)  -
/                   ,ALARM               (type=string)  -
/                   ,HARDCOPY            (type=string)  -
/                   ,AUTOTAB             (type=string)  -
/                   ,MANDATORY           (type=string)  -
/                   ,REFRESH             (type=string)  -
/                   ,ACK                 (type=string)  -
/                   ,KEYLOCK             (type=string)  -
/                   )
/  " field attributes "
/  declare-element ATTR            (type=struc(attr))-
/                 ,mult-elem=list
/
/   " input information " -
/   declare-element -
/                   (COMMAND               (type=string) -
/                   ,FHS-VERSION           (type=string) -
/                   ,CURSOR-INPUT          (type=string) -
/                   ,CURSOR-INPUT-INDEX    (type=integer) -
/                   ,CURSOR-INPUT-POS      (type=integer) -
/                   )
/end-structure
/
/else-if (upper-case(INCLUDE-FORM) == 'INITIALIZE')
```

```
/
/   if (VARIABLE-NAME == '')
/      write-text '%  mandatory parameter variable-name missing.'
/      raise-error
/   end-if
/  declare-variable PARAM(type=string)
/  SYSPRC-NAME = '$.SYSPRC.FHS.083'
/  IF ( SDF-P-VERSION >= 'V02.0A00' )
/     SYSPRC-NAME  = INSTALLATION-PATH -
/                       ( LOGICAL-ID        = 'SYSPRC' -
/                        ,INSTALLATION-UNIT = 'FHS' -
/                        ,VERSION           = 'V08.3' -
/                        ,DEFAULT-PATH-NAME = '&SYSPRC-NAME')
/  END-IF
/  include-procedure *lib-elem(lib=&SYSPRC-NAME,el=FHDR) -
/   "------------ std param ---------------------------------*"-
/               ,(INCLUDE-FORM='INITIALIZE' -
/                ,VARIABLE-NAME='&VARIABLE-NAME..STD-HEADER' -
/   "------------ include specific param -------------------*"-
/                ,UNIT ='FHS',  "fhs unit name" -
/                ,FUNCTION ='DISPLAY', "fhs fc for display?" -
/                ,VERSION = 2 "control variable layout version"-
/                )
/
/   for PARAM = -
/                   ('SERVICE'             -
/                   ,'DIAGINFO'            -
/                   ,'POP-LOCATION'        -
/                   ,'POP-LOC-IND'         -
/                   ,'ROW'                 -
/                   ,'COLUMN'              -
/                   ,'RESOURCE'            -
/                   ,'MESSAGE-ID'          -
/                   ,'MESSAGE-FIELD'       -
/                   ,'MSG-FIELD-IND'       -
/                   ,'CURSOR-OUTPUT-INDEX' -
/                   ,'CURSOR-OUTPUT'       -
/                   ,'CURSOR-OUTPUT-POS'   -
/                   ,'LOCK'                -
/                   ,'ALARM'               -
/                   ,'HARDCOPY'            -
/                   ,'AUTOTAB'             -
/                   ,'MANDATORY'           -
/                   ,'REFRESH'             -
/                   ,'ACK'                 -
/                   ,'KEYLOCK'             -
/                   ,'COMMAND'             -
/                   ,'FHS-VERSION'         -
```

```
/                  ,'CURSOR-INPUT'        -
/                  ,'CURSOR-INPUT-INDEX'  -
/                  ,'CURSOR-INPUT-POS'    -
/                  )
/         &VARIABLE-NAME..&PARAM = &PARAM
/   end-for
/
/   if (ATTR-LIST > 0)
/      I = 1
/      while ( I <= ATTR-LIST )
/         for PARAM = -
/                     ('FIELD'     -
/                     ,'FIELD-IND' -
/                     ,'TYPE'      -
/                     ,'HILITE'    -
/                     ,'INTENSITY' -
/                     ,'COLOR'     -
/                     ,'OUTPUT'    -
/                     )
/            &VARIABLE-NAME..ATTR#I.&PARAM = &PARAM
/         end-for
/         I=I+1
/      end-while
/   end-if
/
/else
/  write-text '%  form=&INCLUDE-FORM not supported; include aborts'
/  raise-error
/end-if
/EXIT-PROCEDURE
```

Hinweis

In einem Include wird in dieser Prozedur der Standard Header eingebunden. Das ist eine S-Prozedur, die mit SDF-P mitgeliefert wird und für Funktionsidentifikationen und Returncode-Informationen zuständig ist (siehe dazu Kommando TRANSMIT-BY-STREAM auf Seite 796).

Beispiel

Die folgende Beispielprozedur illustriert, wie die Variable MYVAR mithilfe von SYSFHS-CONTROL und dem Standard Header deklariert und initialisiert wird.

```
/INCLUDE-PROC *LIB-ELEM(LIB=$TSOS.SYSPRC.SDF-P-BASYS.024,EL=FHDR),-
/(PREFIX='SYSFHS-')
/INCLUDE-PROC *LIB-ELEM(LIB=$TSOS.SYSPRC.FHS.083,EL=SYSFHS-CONTROL)
/DECLARE-VAR MYVAR(TYPE=*STRUCT(SYSFHS-LAYOUT))
/INCLUDE-PROC *LIB-ELEM(LIB=$TSOS.SYSPRC.FHS.083,EL=SYSFHS-CONTROL),-
/(INCLUDE-FORM=INITIALIZE,VARIABLE-NAME='MYVAR', ATTR-LIST = 2)

/SHOW-VARIABLE MYVAR
```

Die Variable wird dann in der folgenden Form generiert und initialisiert:

```
MYVAR.STD-HEADER.INTERFACE-ID.UNIT = FHS
MYVAR.STD-HEADER.INTERFACE-ID.FUNCTION = DISPLAY
MYVAR.STD-HEADER.INTERFACE-ID.VERSION = 2
MYVAR.STD-HEADER.RETURNCODE.SUBCODE2 = 0
MYVAR.STD-HEADER.RETURNCODE.SUBCODE1 = 0
MYVAR.STD-HEADER.RETURNCODE.MAINCODE = CMD0001
MYVAR.SERVICE = *DISPLAY
MYVAR.DIAGINFO = *NO
MYVAR.POP-LOCATION = *NONE
MYVAR.POP-LOC-IND = 0
MYVAR.ROW = 2
MYVAR.COLUMN = 2
MYVAR.RESOURCE = *SAME
MYVAR.MESSAGE-ID = *NONE
MYVAR.MESSAGE-FIELD = *NONE
MYVAR.MSG-FIELD-IND = 0
MYVAR.CURSOR-OUTPUT-INDEX = 0
MYVAR.CURSOR-OUTPUT = *NONE
MYVAR.CURSOR-OUTPUT-POS = 0
MYVAR.LOCK = *NO
MYVAR.ALARM = *NO
MYVAR.HARDCOPY = *NO
MYVAR.AUTOTAB = *YES
MYVAR.MANDATORY = *NO
MYVAR.REFRESH = *NO
MYVAR.ACK = *NO
MYVAR.KEYLOCK = *NONE
MYVAR.ATTR(*LIST).FIELD = *CURSOR
MYVAR.ATTR(*LIST).FIELD-IND = 0
MYVAR.ATTR(*LIST).TYPE = *UNCHANGED
MYVAR.ATTR(*LIST).HILITE = *UNCHANGED
MYVAR.ATTR(*LIST).INTENSITY = *UNCHANGED
MYVAR.ATTR(*LIST).COLOR = *UNCHANGED
```

```
MYVAR.ATTR(*LIST).OUTPUT = *UNCHANGED
MYVAR.ATTR(*LIST).FIELD = *CURSOR
MYVAR.ATTR(*LIST).FIELD-IND = 0
MYVAR.ATTR(*LIST).TYPE = *UNCHANGED
MYVAR.ATTR(*LIST).HILITE = *UNCHANGED
MYVAR.ATTR(*LIST).INTENSITY = *UNCHANGED
MYVAR.ATTR(*LIST).COLOR = *UNCHANGED
MYVAR.ATTR(*LIST).OUTPUT = *UNCHANGED
MYVAR.COMMAND =
MYVAR.FHS-VERSION =
MYVAR.CURSOR-INPUT =
MYVAR.CURSOR-INPUT-INDEX = 0
MYVAR.CURSOR-INPUT-POS = 0
```

FHS-Returncodes

SDF-P belegt die Kommando-Returncodes von TRANSMIT-BY-STREAM. Die Returncode-Variable der zurückgegebenen Struktur wird von FHS belegt.

Name	Datentyp	Returnwerte
SUBCODE2	integer	<integer>
SUBCODE1	integer	<integer>
MAINCODE	string	<name 1..7>

7.3.2 FHS-Anwendungen durch S-Prozeduren nutzen und steuern

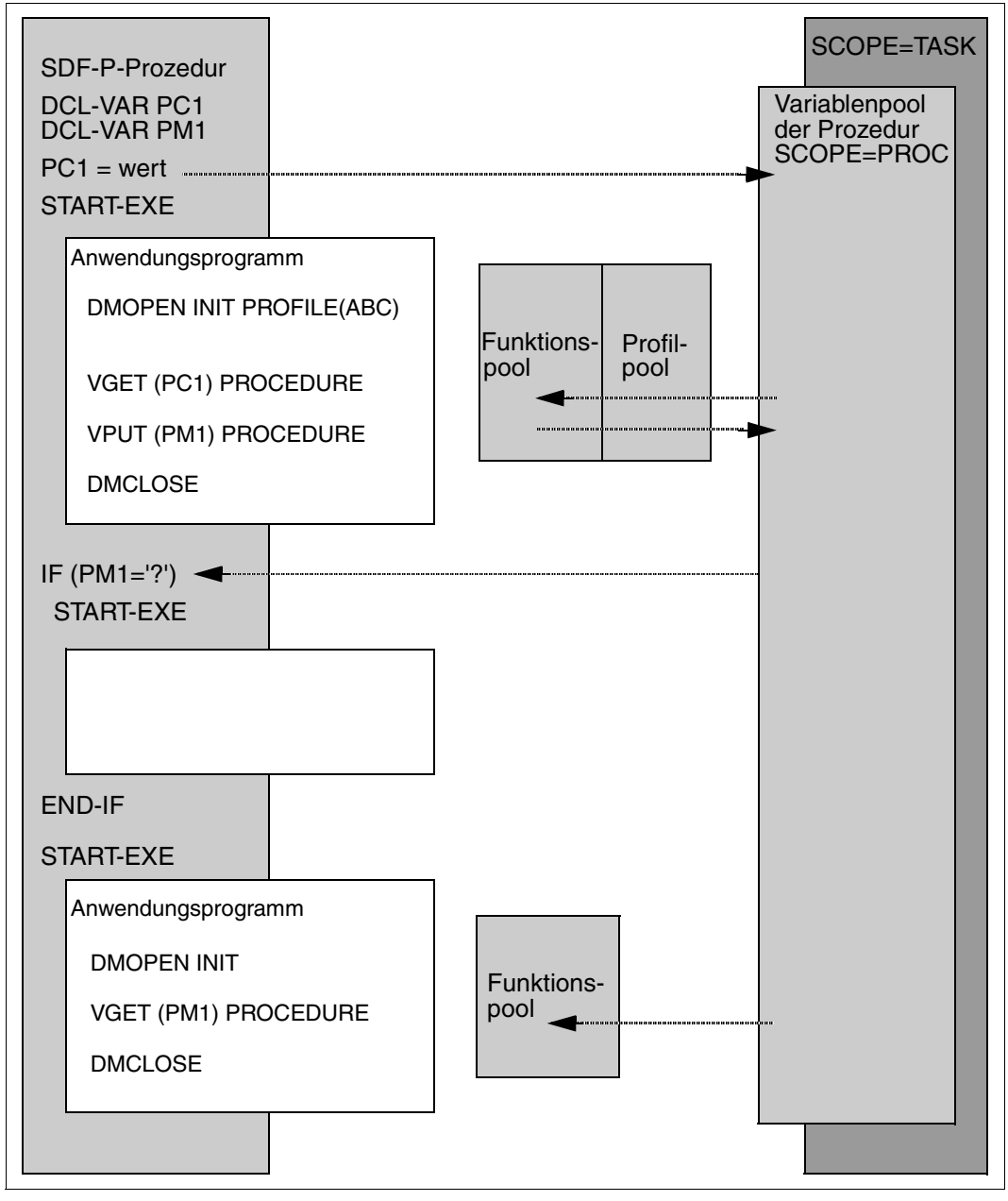
7.3.2.1 S-Variablen mit FHS ausgeben

S-Prozeduren können die Kommandos ASSIGN-STREAM und TRANSMIT-BY-STREAM benutzen, um einen Dialog mit dem Endbenützer mittels FHS-Bedienfeldern zu starten, die mit FHS-Dialog-Variablen gefüllt sind.

7.3.2.2 S-Variablen in FHS-TIAM-Programmen ausgeben und erzeugen

Die Makros VGET und VPUT in FHS erlauben einem Anwendungsprogramm, einfache S-Variable zu lesen und zu schreiben. Damit ist eine Kommunikation zwischen einer S-Prozedur und einem von dieser Prozedur aufgerufenen Anwendungsprogramm möglich.

Das folgende Bild zeigt den Variablenaustausch zwischen Anwendungsprogramm und einer S-Prozedur.



7.3.2.3 FHS-Anwendungen in geschachtelten S-Prozeduren steuern

In geschachtelten S-Prozeduren werden Zuweisungen von S-Variablenströmen genauso wie bei Systemdateien (SYSDTA,SYSOUT,..) gestapelt. Der Anwender sollte darum im Kommando SET-PROCEDURE-OPTIONS die Operandeneinstellung SYSTEM-FILE-CONTEXT=*STD oder *OWN benutzen, wenn er seine Stromzuweisungen auch gestapelt verarbeiten will.

Im TPR-Modus sichert FHS seine Anzeige-Umgebung in Übereinstimmung mit der S-Variablenströmen-Zuweisung im Kommando TRANSMIT-BY-STREAM. Zur Zuweisungszeit initialisiert FHS einen dem Variablenstrom spezifischen Kontext, der automatisch bei jeder FHS-Operation mit demselben Variablenstrom benutzt wird. Dies passiert bis zum Ende der Zuweisung, z.B. wenn demselben Variablenstrom *DUMMY oder ein anderer Server zugewiesen wird - entweder explizit oder implizit bei Prozedurende (gemäß der SYSTEM-FILE-CONTEXT-Zuweisung).

Dieser Mechanismus schließt ein, dass geschachtelte Prozeduren unabhängig codiert sein können ohne Überlagerungen von Anzeigebefehlen in verschiedenen FHS-Kontexten.

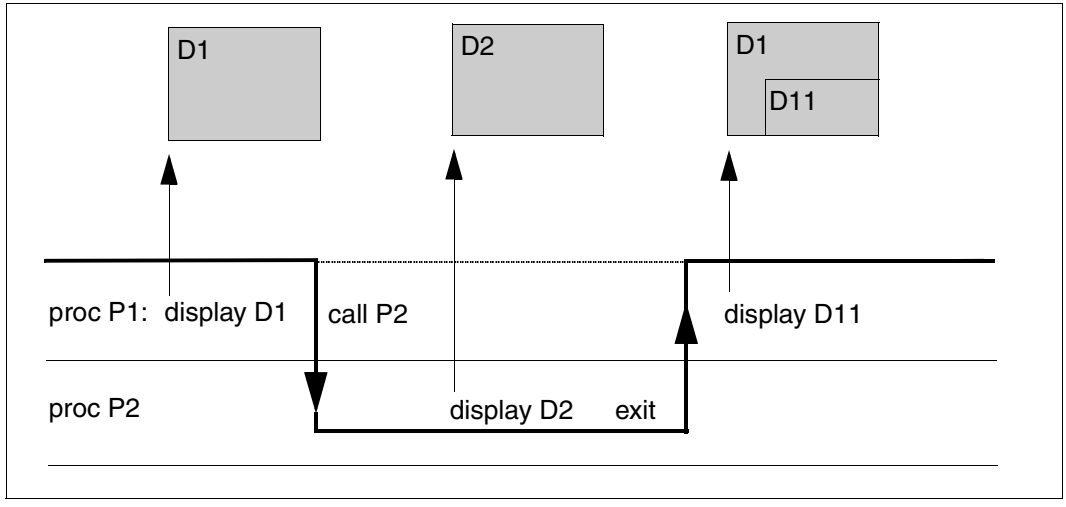
Um die Unabhängigkeit von der aufrufenden Prozedur zu sichern, ist es notwendig der Variablen SYSFHS-CONTROL.REFRESH den Wert *YES zuzuweisen, um ein Bedienfeld nach dem Aufruf dieser Prozedur auszugeben.

Beispiel

In der Prozedur P1 wird der Variablenstrom S1 FHS zugewiesen; das Bedienfeld D1 wird in der Prozedur P1 angezeigt. P1 ruft die Prozedur P2 auf ; der Variablenstrom S2 in P2 ist ebenfalls FHS zugewiesen; P2 zeigt das Bedienfeld D2 an. D2 überschreibt D1 vollkommen.

```
Proc P1 :  assign   S1 -----> FHS
           displays D1
           calls    P2
           pop-up   D11

Proc P2 :  assign   S2 -----> FHS
           displays D2
           exit
```

Nach Beendigung von P2 geht FHS in die Anzeige-Umgebung von S1 zurück. Dafür wird das „Pop up-Menü“ auf dem gegenwärtigen Bildschirm (mit oder ohne Abzweigung, z.B. Bedienfeld D11) wieder implizit auf dem gegenwärtigen Bedienfeld von P1 (D1) wiederhergestellt.



Das Bedienfeld D2 ist gelöscht worden und das Bedienfeld D1 ist bei der nächsten Anzeige der Prozedur P1 wiederhergestellt worden.

Hinweis

Variablen, die in einem Bedienfeld angezeigt werden sollen, müssen in der gegenwärtigen S-Prozedur sichtbar sein (siehe Abschnitt „Geltungsbereich von Variablen“ auf Seite 162).

7.3.2.4 Anwendungsbeispiel

In diesem Beispiel wird ausführlich eine Anwendungsmöglichkeit gezeigt, wie das Zusammenspiel von S-Prozeduren, S-Variablenströmen und FHS für die Erstellung eines graphischen Bibliotheksmanagers genutzt werden kann.

Die einzelnen dafür verantwortlichen S-Prozeduren sind als Elemente in der (Benutzer-) Bibliothek LIBRARY-MANAGER.PL unter den Namen RUN, SCREEN01 und SCREEN02 mit dem Typ J gespeichert. Dabei ist RUN die steuernde S-Prozedur, die mit dem Kommando CALL-PROCEDURE gestartet wird. SCREEN01 (siehe Seite 219) und SCREEN02 (siehe Seite 222) sind dagegen davon abhängig für die zwei möglichen Standard-Bildschirmanzeigen verantwortlich.

Im Folgenden werden diese drei S-Prozeduren aufgelistet. Anschließend werden einige Anwendungen gezeigt, um zu demonstrieren, wie der FHS-unterstützte Bibliotheksmanager eingesetzt werden kann.

Prozedur: RUN

```
/SET-PROCEDURE-OPTIONS CALLER=CALL
/
/"-----------------------------------------------------------"
/"First get library name from which this procedure is called"
/"-----------------------------------------------------------"
/
/DECLARE-VARIABLE SYSOUT(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
/EXECUTE-CMD (SHOW-SYSTEM-FILE-ASSIGNMENT SYSTEM-FILE=*SYSCMD),-
            /STRUCTURE-OUTPUT=SYSOUT,TEXT-OUTPUT=*NONE
/LIBRARY-NAME = SYSOUT#1.SYSCMD.LIB
/
/"-----------------------------------------------------------"
/"Get FHS library names via IMON-GPN                         "
/"-----------------------------------------------------------"
/
/FHSLNK = '$TSOS.SYSFHS.FHS.082.FHS-DM.D'
/FHSLNK = INSTALLATION-PATH(LOGICAL-ID        = 'SYSFHS.FHS-DM.D' , -
                           /INSTALLATION-UNIT = 'FHS' , -
                           /VERSION           = *STD , -
                           /DEFAULT-PATH-NAME = FHSLNK)
/
/FHSPRC = '$TSOS.SYSPRC.FHS.082'
/FHSPRC = INSTALLATION-PATH(LOGICAL-ID        = 'SYSPRC' , -
                           /INSTALLATION-UNIT = 'FHS' , -
                           /VERSION           = *STD , -
                           /DEFAULT-PATH-NAME = FHSPRC)
/
/"-----------------------------------------------------------"
/"Get SDF-P-BASYS library name via IMON-GPN                  "
```

```
/"-------------------------------------------------------------"
/
/SDPPRC = '$TSOS.SYSPRC.SDF-P-BASYS.022'
/SDPPRC = INSTALLATION-PATH(LOGICAL-ID        = 'SYSPRC' , -
                           /INSTALLATION-UNIT = 'SDF-P-BASYS' , -
                           /VERSION           = *STD , -
                           /DEFAULT-PATH-NAME = SDPPRC)
/
/"-------------------------------------------------------------"
/"Initialize FHS control variables                            "
/"-------------------------------------------------------------"
/WRITE-TEXT 'LIBRARY MANAGER V1.0 - LOADING'
/SHOW-VAR *ALL
/INCLUDE-PROCEDURE *LIBRARY-ELEMENT(LIBRARY = &(SDPPRC) -
/                                  ,ELEMENT = FHDR) -
/                 ,PROCEDURE-PARAMETERS = (PREFIX = 'SYSFHS-')
/SHOW-VAR *ALL
/INCLUDE-PROCEDURE *LIBRARY-ELEMENT(LIBRARY = &(FHSPRC) -
/                                  ,ELEMENT = SYSFHS-CONTROL)
/DECLARE-VARIABLE  SYSPINFO (TYPE = *STRUCTURE(SYSFHS-LAYOUT))
/DECLARE-VARIABLE  SYSPINFO-SAVE (TYPE = *STRUCTURE(SYSFHS-LAYOUT))
/INCLUDE-PROCEDURE *LIBRARY-ELEMENT(LIBRARY = &(FHSPRC) -
/                                  ,ELEMENT = SYSFHS-CONTROL) -
/                 ,PROCEDURE-PARAMETERS = (INCLUDE-FORM='INITIALIZE' -
/                                         ,VARIABLE-NAME='SYSPINFO')
/
/"-------------------------------------------------------------"
/"Assign the stream to FHS                                     "
/"-------------------------------------------------------------"
/ADD-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=&(FHSLNK)
/ASSIGN-STREAM STREAM-NAME = PRESENTATION -
/             ,TO          = *SERVER(FHS -
/             ,SERVER-INFO = 'FHS-LIB = &(LIBRARY-NAME)')
/
/"-------------------------------------------------------------"
/"Start LMS                                                    "
/"-------------------------------------------------------------"
/ASSIGN-SYSOUT TO=*DUMMY
/START-LMS
/HOLD-PROGRAM
/ASSIGN-SYSOUT TO=*PRIMARY
/
/"-------------------------------------------------------------------"
/"Set timeout to 0 when switching from line mode to full screen    "
/"-------------------------------------------------------------------"
/MODIFY-TERMINAL-OPTIONS OVERFLOW-CONTROL = *TIME(TIMEOUT = 0)
/
/"-------------------------------------------------------------"
```

```
/"Call main procedure (screen01)                                    "
/"-------------------------------------------------------------------"
/INCLUDE-PROCEDURE *LIBRARY-ELEMENT(LIBRARY = &(LIBRARY-NAME) -
/                                  ,ELEMENT = SCREEN01)
/
/"-------------------------------------------------------------------"
/"Stop  LMS                                                         "
/"-------------------------------------------------------------------"
/RESUME-PROGRAM
//END
```

Prozedur: SCREEN01

```
/declare-variable screen01(type=*structure(*by-syscmd))
/begin-structure
/  declare-element name = filelist (type = *structure(*by-syscmd)) -
/                 ,multiple-elements = *list
/  begin-structure
/    declare-element choice
/    declare-element f-size
/    declare-element cat-id
/    declare-element user-id
/    declare-element short-f-name
/  end-structure
/  declare-element name = sdfplist-modindex (type = integer) -
/                 ,multiple-elements = *list
/  declare-element sdfplist-topindex(initial-value = 1)
/  declare-element sdfplist-botindex
/  declare-element sdfplist-numrow
/  declare-element file-menu
/  declare-element file-choice
/end-structure
/
/declare-variable i(type = *integer)
/
/while (true)
/
/     "initialize modindex list for 50 elements "
/     " (fhs requirement)                       "
/     for i = *counter(1,50)
/        screen01.sdfplist-modindex#&(i) = 0
/     end-for
/
/     "get library names"
/     exec-cmd cmd=(show-file-attributes -
/                      select=*by-attributes(type-of-files = *plam-library) -
/                     ,info=*name-and-space -
```

```
/                  ) -
/             ,structure-output=screen01.filelist -
/             ,text-output=*none -
/             ,returncode=*variable(subcode2=sub2 -
/                                  ,subcode1=sub1 -
/                                  ,maincode=main)
/
/     if (sub1 ne 0)
/        write-text 'Error &sub2 &sub1 &main returned by EXEC-CMD'
/        write-text 'LIBRARY MANAGER V1.0 abnormally terminated'
/        exit-procedure
/     end-if
/
/     syspinfo.resource = 'screen01'
/     syspinfo.service = '*display'
/     syspinfo.refresh = '*yes'
/     syspinfo.command = ''
/     screen01.sdfplist-numrow = size('screen01.filelist')
/     screen01.file-menu=0
/     screen01.file-choice=0
/     transmit-by-stream variable-name = screen01 -
/                        ,stream-name = presentation -
/                        ,control-var-name = syspinfo
/
/     if (   (syspinfo.std-header.returncode.maincode == 'IDH0004') -
/         or (syspinfo.std-header.returncode.maincode == 'IDH0008'))
/        write-text 'LIBRARY MANAGER V1.0 normally terminated'
/        exit-procedure
/     end-if
/
/     if (syspinfo.std-header.returncode.maincode ne 'IDH0000')
/        sub2 = syspinfo.std-header.returncode.subcode2
/        sub1 = syspinfo.std-header.returncode.subcode1
/        main = syspinfo.std-header.returncode.maincode
/        write-text 'Error &sub2 &sub1 &main returned by FHS server'
/        write-text 'LIBRARY MANAGER V1.0 abnormally terminated'
/        exit-procedure
/     end-if
/
/     if screen01.file-menu ne 0
/        if screen01.file-choice == 9
/           write-text 'LIBRARY MANAGER V1.0 normally terminated'
/           exit-procedure
/        else-if screen01.file-choice == 1
/         for i = *counter(1,size('screen01.sdfplist-modindex')), -
             /cond=(screen01.sdfplist-modindex#i ne 0)
/              screen01-curr-index = screen01.sdfplist-modindex#i
/              if screen01.filelist#screen01-curr-index.choice == '/'
```

```
/                 syspinfo-save = syspinfo
/                 include-procedure -
/                    name=*library-element(&library-name. -
/                                          ,screen02) -
/                   ,procedure-parameters=(&(screen01.filelist#screen01-curr-
index.short-f-name))
/                 if-cmd-error
/                    write-text 'LIBRARY MANAGER V1.0 abnormally terminated'
/                    exit-procedure
/                 else
/                    save-returncode
/                    if (maincode() = 'STOPOOK')
/                       write-text 'LIBRARY MANAGER V1.0 normally terminated'
/                       exit-procedure
/                    end-if
/                 end-if
/                 syspinfo = syspinfo-save
/              end-if
/           end-for
/        end-if
/     end-if
/
/     if (syspinfo.command ne '')
/        exec-cmd cmd=(&(syspinfo.command)) -
/                ,text-output=*none -
/                ,returncode=*variable(subcode2=sub2 -
/                                      ,subcode1=sub1 -
/                                      ,maincode=main)
/
/        if (sub1 ne 0)
/           write-text 'Error &sub2 &sub1 &main returned by command server'
/           write-text 'LIBRARY MANAGER V1.0 abnormally terminated'
/           exit-procedure
/        end-if
/     end-if
/end-while
```

Prozedur: SCREEN02

```
/begin-parameter-declaration
/  declare-parameter library
/end-parameter-declaration
/
/declare-variable screen02(type=*structure(*by-syscmd))
/begin-structure
/  declare-element name = elemlist(type = *structure(*dynamic)) -
/                 ,multiple-element = *list
/  declare-element name = sdfplist-modindex(type = *integer) -
/                 ,multiple-element = *list
/  declare-element sdfplist-topindex(initial-value = 1)
/  declare-element sdfplist-botindex
/  declare-element sdfplist-numrow
/  declare-element file-menu
/  declare-element file-choice
/end-structure
/
/declare-variable sysout(type = *string), multiple-elements = *list
/declare-variable error-on-print( type = *boolean, initial-value = false )
/declare-variable i(type = *integer)
/
/resume-program
//open-library library = &library.,mode = *update
/hold-program
/
/while (true)
/
/   "initialize modindex list for 50 elements "
/   " (fhs requirement)                       "
/   for i = *counter(1,50)
/      screen02.sdfplist-modindex#&(i) = 0
/   end-for
/
/   assign-sysout to = *variable(sysout)
/   resume-program
//  show-element-attributes -
//       element = *library-element(library = *std -
//                                 ,element = *all ( version = *all ) -
//                                 ,type = *all) -
//      ,information = *maximum -
//      ,sort = *by-name -
//      ,structure-output = screen02.elemlist
/   hold-program
/   assign-sysout to = *primary
/
/   if ( stmt-spinoff() == 'YES' )
```

```
/      show-variable sysout, information = *parameters( name = *none )
/      maincode = 'LMSOERR'
/      goto end
/   end-if
/
/   "Following loop is only necessary to rep a problem between"
/   "FHS and VAS. Correction in VAS V02.0A85, FHS V08.1A75"
/   for i = *counter(1,size('screen02.elemlist'))
/       screen02.elemlist#i.choice = ' '
/   end-for
/
/   syspinfo.resource = 'screen02'
/   syspinfo.service = '*display'
/   syspinfo.refresh = '*yes'
/   syspinfo.command = ''
/   screen02.sdfplist-numrow = size('screen02.elemlist')
/   screen02.file-menu=0
/   screen02.file-choice=0
/   transmit-by-stream variable-name = screen02 -
/                     ,stream-name = presentation -
/                     ,control-var-name = syspinfo
/
/   if (   (syspinfo.std-header.returncode.maincode == 'IDH0004') -
/       or (syspinfo.std-header.returncode.maincode == 'IDH0008'))
/      maincode = 'FHSEXIT'
/      goto end
/   end-if
/
/   if (syspinfo.std-header.returncode.maincode ne 'IDH0000')
/      sub2 = syspinfo.std-header.returncode.subcode2
/      sub1 = syspinfo.std-header.returncode.subcode1
/      main = syspinfo.std-header.returncode.maincode
/      write-text 'Error &sub2 &sub1 &main returned by FHS server'
/      maincode = 'FHSOERR'
/      goto end
/   end-if
/
/   if screen02.file-menu ne 0
/      if screen02.file-choice == 9
/         maincode = 'FHSORET'
/         goto end
/      else
/         for i = *counter(1,size('screen02.sdfplist-modindex')), -
             /cond=(screen02.sdfplist-modindex#i ne 0)
/            screen02-curr-index = screen02.sdfplist-modindex#i
/            if screen02.elemlist#screen02-curr-index.choice == '/'
/               element = screen02.elemlist#screen02-curr-index.elem
/               version = screen02.elemlist#screen02-curr-index.version
```

```
/               type    = screen02.elemlist#screen02-curr-index.type
/               if screen02.file-choice == 1 "delete element"
/                  assign-sysout to = *variable(sysout)
/                  resume-program
//                 delete-element element = *library-element -
//                                       ( library = *std -
//                                       , element = &element.-
//                                           ( version = &version. ) -
//                                       , type = &type. )
/                  hold-program
/                  assign-sysout *primary
/               else-if screen02.file-choice == 2 "edit element"
/                  assign-sysout to = *variable(sysout)
/                  resume-program
//                 edit-element element = *library-element -
//                                       ( library = *std -
//                                       , element = &element. -
//                                           ( version = &version. ) -
//                                       , type = &type. )
/                  hold-program
/                  assign-sysout *primary
/               else-if screen02.file-choice == 3 "copy element"
/                  write-text 'Function not implemented'
/               else-if screen02.file-choice == 4 "print element"
/                  assign-sysout to = *variable(sysout)
/                  print-file *library-element -
/                                ( library = &library. -
/                                , element = &element. -
/                                    ( version = &version. ) -
/                                , type = &type. )
/                  if-cmd-error
/                     error-on-print = true
/                  end-if
/                  assign-sysout *primary
/               else-if screen02.file-choice == 5 "select element"
/                  assign-sysout to = *variable(sysout)
/                  resume-program
//                 extract-element element = *library-element -
//                                       ( library = *std -
//                                       , element = &element. -
//                                           ( version = &version. ) -
//                                       , type = &type. ) -
//                                 ,to-file = *std
/                  hold-program
/                  assign-sysout *primary
/               else-if screen02.file-choice == 6 "add element"
/                  write-text 'Function not implemented'
/               end-if
```

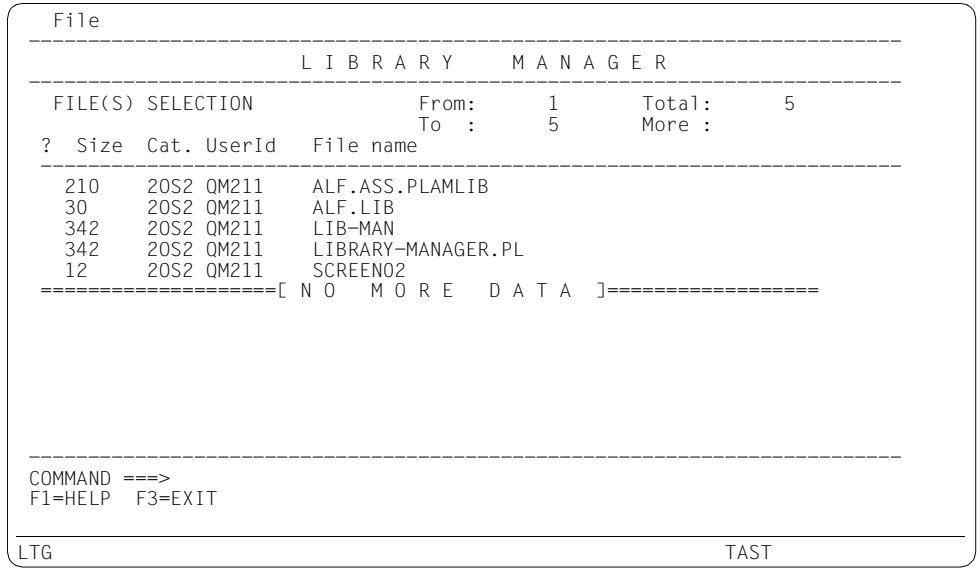
```
/
/               if ( error-on-print )
/                  show-variable sysout, information = *parameters( name =
*none )
/                  maincode = 'PRTOERR'
/                  goto end
/               end-if
/
/               if ( stmt-spinoff() == 'YES' )
/                  show-variable sysout, information = *parameters( name =
*none )
/                  maincode = 'LMSOERR'
/                  goto end
/               end-if
/            end-if
/         end-for
/      end-if
/   end-if
/
/   if (syspinfo.command ne '')
/      exec-cmd cmd=(&(syspinfo.command)) -
/              ,text-output=*none -
/              ,returncode=*variable(subcode2=sub2 -
/                                    ,subcode1=sub1 -
/                                    ,maincode=main)
/
/      if (sub1 ne 0)
/         write-text 'Error &sub2 &sub1 &main returned by command server'
/         maincode = 'CMDOERR'
/         goto end
/      end-if
/   end-if
/end-while
/
/
/end:
/if (   (maincode = 'CMDOERR') -
/    or (maincode = 'PRTOERR') -
/    or (maincode = 'LMSOERR') -
/    or (maincode = 'FHSOERR') -
/   )
/   exit-procedure error = *yes(subcode2 = 0 -
/                               ,subcode1 = 64 -
/                               ,maincode = STOPERR)
/else-if (maincode = 'FHSEXIT')
/   exit-procedure error = *yes(subcode2 = 0 -
/                               ,subcode1 = 0  -
/                               ,maincode = STOPOOK)
```

```
/else-if (maincode = 'FHSORET')
/   exit-procedure
/else
/   write-text 'Error &(sc2()) &(sc1()) &(mc()) reported'
/   exit-procedure error = *yes(subcode2 = 0 -
/                              ,subcode1 = 64 -
/                              ,maincode = STOPERR)
/end-if
```

Unter der Voraussetzung, dass FHS-PRIV geladen ist, kann der Bibliotheksmanager folgendermaßen aufgerufen werden:

```
/CALL-PROCEDURE FROM-FILE=*LIBRARY-ELEMENT(LIBRARY-MANAGER.PL, RUN)
```

Es muss dann eine Bildschirmanzeige (unter Benutzung von SCREEN01) erscheinen, welche die Namen der in der Kennung enthaltenen Bibliotheken auflistet, wie z.B:



Man hat nun z.B. die Möglichkeit, eine der aufgelisteten Bibliotheken zu öffnen und sich die jeweiligen Elemente anzeigen zu lassen. Dazu kennzeichnet man mit einem „/“ am Zeilenanfang die zu öffnende Bibliothek, springt dann mit der Tabulatortaste zum Feld „FILE“ (links oben), drückt die DUE-Taste und gibt in dem erscheinenden Pull-Down-Menü „1“ an.

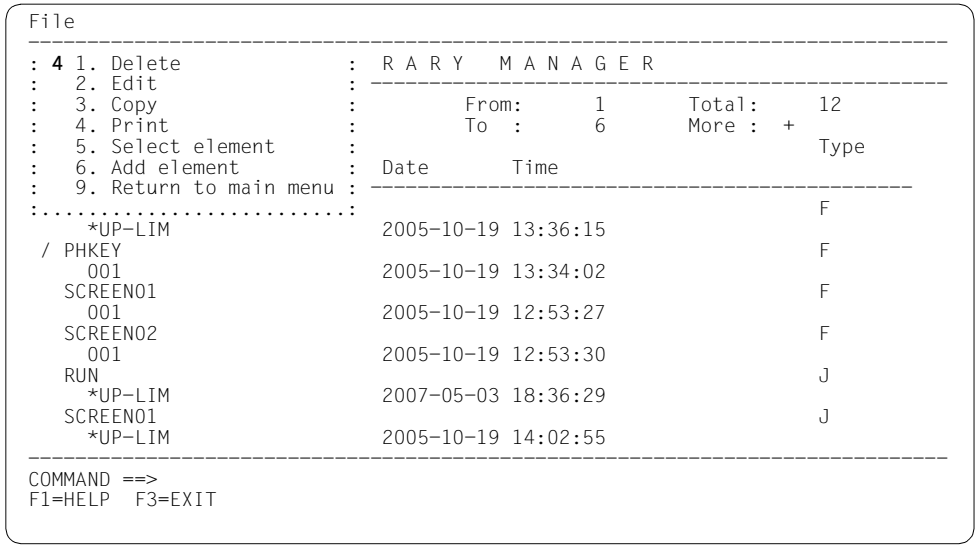
Die folgende Bidschirmausgabe soll dies verdeutlichen:

```
  File
--------------------------------------------------------------------------
: 1 1. Open library         :  A R Y    M A N A G E R
:   9. Exit Library-manager : -----------------------------------------
:...........................:    From:      1      Total:      5
                                 To  :      5      More :
 ?  Size  Cat. UserId   File name
 -------------------------------------------------------------------------
   210    20S2 QM211    ALF.ASS.PLAMLIB
   30     20S2 QM211    ALF.LIB
 / 342    20S2 QM211    LIB-MAN
   342    20S2 QM211    LIBRARY-MANAGER.PL
   12     20S2 QM211    SCREEN02
 ====================[ N O   M O R E   D A T A  ]==================







--------------------------------------------------------------------------
COMMAND ===>
F1=HELP  F3=EXIT
```

Nach nochmaligem Drücken der [DUE]-Taste, erscheinen in einer weiteren Bildschimausgabe (unter Benutzung von SCREEN02) die jeweiligen Elementnamen dieser Bibliothek (zusammen mit Datum und Zeit ihrer Erstellung sowie dem Typ usw.).

```
File
------------------------------------------------------------------------------
                        L I B R A R Y   M A N A G E R
------------------------------------------------------------------------------
  ELEMENT(S) SELECTION             From:      1      Total:     12
                                   To  :      6      More :  +
 ? Element                                                      Type
     Version                  Date       Time
 ---------------------------------------------------------------------------
   LISTHLP                                                      F
     *UP-LIM                  2005-10-19 13:36:15
   PHKEY                                                        F
     001                      2005-10-19 13:34:02
   SCREEN01                                                     F
     001                      2005-10-19 12:53:27
   SCREEN02                                                     F
     001                      2005-10-19 12:53:30
   RUN                                                          J
     *UP-LIM                  2007-05-03 18:36:29
   SCREEN01                                                     J
     *UP-LIM                  2005-10-19 14:02:55
------------------------------------------------------------------------------
COMMAND ==>
F1=HELP  F3=EXIT
```

Weiter hat man nun die Möglichkeit, eines der aufgelisteten Elemente zu bearbeiten. Dazu kennzeichnet man mit einem „/“ am Zeilenanfang das zu bearbeitende Element (z.B. PHKEY), springt dann mit der Tabulatortaste zum Feld „FILE“ (links oben), drückt die DUE-Taste und gibt in dem erscheinenden Pull-Down-Menü die Zahl an, die markiert, wie man das Element bearbeiten will: z.B. „4“, um es auszudrucken.



Mit Betätigung der DUE-Taste wird dies dann ausgeführt.

Soviel zu einigen Anwendungsmöglichkeiten: Mit der F3-Taste kann man den Bibliotheksmanager verlassen bzw. zum Ausgangsmenü zurückkehren.

Im Übrigen kann man mit Eingabe von „+“ oder „-“ in der Zeile „COMMAND“ nach dem Pfeil. scrollen. Ein Hilfemenü ist durch Auslösen der F1-Taste zu erreichen.

8 Funktionen

Funktionen in SDF-P sind folgendermaßen charakterisiert:

– Funktionen werden über einen Funktionsnamen aufgerufen.
– Aus den Eingabeparametern wird das Funktionsergebnis ermittelt.
– Es wird genau ein Ergebniswert zurückgegeben.
– Der Ergebniswert wird an der Stelle des Funktionsaufrufs eingesetzt.

Zum Lieferumfang von SDF-P gehören „vordefinierte Funktionen“ (auch als „built-in“-Funktionen bezeichnet), mit denen der Anwender Variablen und Zeichenfolgen bearbeiten oder Informationen über die aktuelle Systemumgebung abfragen kann. Dazu unterstützt SDF-P auch Funktionen, die der Systemverwalter über eine spezielle Assembler-Schnittstelle erstellt.

In diesem Kapitel wird nun einerseits beschrieben, wie Funktionen allgemein eingesetzt werden können, d.h. wie sie aufgerufen werden usw., und andererseits was zu berücksichtigen ist, wenn der Systemverwalter eigene Funktionen schreibt.

8.1 Funktionsaufruf

Funktionen werden über ihren Funktionsnamen aufgerufen.

SDF-P erkennt eine Funktion an den Klammern, die dem Funktionsnamen folgen, z.B. USER-IDENTIFICATION( ).

Diese Klammern können entfallen, wenn der Funktionsaufruf keine Eingabeparameter enthält. Dann interpretiert SDF-P den (Funktions-)Namen jedoch zunächst als Variablennamen. Nur wenn es keine Variable dieses Namens gibt, interpretiert SDF-P den Namen als Funktionsnamen.

Existiert eine Variable des gleichen Namens, greift SDF-P auf die Variable zu und setzt gegebenenfalls den Variablenwert ein.

Hinweis

Beim Funktionsaufruf sollten immer die Klammern mit angegeben werden, damit keine Verwechslungsmöglichkeit mit Variablen besteht.

Funktionen sind Bestandteile von Ausdrücken. Sie können also überall dort aufgerufen werden, wo Ausdrücke zugelassen sind.

In Zuweisungen können alle Funktionen auf der rechten Seite der Zuweisung (d.h. rechts vom Gleichheitszeichen) aufgerufen werden. So wird zum Beispiel einer Variablen der Ergebniswert der Funktion zugewiesen.

Beispiel

```
/BENUTZER = USER-IDENTIFICATION( )
```

Der Variablen BENUTZER wird die Benutzerkennung als Wert zugewiesen.

Werden Funktionen in Ausdrücken aufgerufen, setzt SDF-P bei der Auswertung des Ausdrucks das Ergebnis der Funktion an der Stelle in den Ausdruck ein, an der die Funktion aufgerufen wird.

Beispiel

```
/DECLARE-VARIABLE VAR-A
/NAME = 'ELEM1' // FIRST-VARIABLE-NAME('VAR-A')
/SHOW-VARIABLE NAME
NAME = ELEM1*END
```

Der Variablen NAME wird ein String zugewiesen, der zusammengesetzt wird aus dem String ELEM1 und dem String, den die Funktion als Ergebnis liefert (das ist in diesem Fall *END).

In Kommandoaufrufen können Funktionen dazu eingesetzt werden, ein Kommando aufzubauen. Zum Beispiel können Funktionen bei der Zuweisung von Operandenwerten aufgerufen werden. Sie können auch wie Variablen bei der &-Ersetzung verwendet werden.

Beispiel

```
/DECLARE-VARIABLE ST(TYPE=*STRUCTURE(*DYNAMIC))
/SET-VARIABLE ST.A1 = 'ANNA'
/SHOW-VARIABLE ST.A1
ST.A1 = ANNA
/SET-VARIABLE &(FIRST-VARIABLE-NAME('ST')) = 'MARIA'
/SHOW-VARIABLE ST.A1
ST.A1 = MARIA
```

Dem ersten Element der Struktur ST, das durch &-Ersetzung der Funktion FIRST-VARIABLE-NAME( ) ermittelt werden kann, wird als Inhalt der String 'MARIA' zugewiesen (und damit der alte Inhalt überschrieben).

8.1.1 Eingabeparameter beim Funktionsaufruf

Beim Aufruf von Funktionen ist zu unterscheiden zwischen Funktionen ohne Eingabeparameter und Funktionen mit Eingabeparametern.

Hinweis

In den folgenden Abschnitten sind die Syntaxregeln für Funktionsaufrufe erklärt, einschließlich der Abkürzungsmöglichkeiten. Diese Abkürzungsmöglichkeiten sollten jedoch nur dann genutzt werden, wenn die Auswertung leicht durchschaubar bleibt. Entsprechend den Regeln der strukturierten Programmierung und vor allem der besseren Wartbarkeit wegen sollten möglichst vollständige Funktionsaufrufe verwendet werden.

In den beiden folgenden Abschnitten werden die Syntax für Funktionsaufrufe ohne und für Funktionsaufrufe mit Eingabeparametern getrennt behandelt, da die Syntaxdarstellung so übersichtlicher bleibt.

8.1.2 Funktionen ohne Eingabeparameter

Syntax

```
function [ ( ) ]
```

function
Name der Funktion
( )
Kennzeichen für Funktionsaufruf

Die Klammern ( ) zeigen an, dass es sich um eine Funktion handelt und nicht um eine Variable. Wenn Funktions- und Variablennamen eindeutig vergeben sind, können die Klammern entfallen.

Wenn Funktions- und Variablennamen nicht eindeutig vergeben sind, das heißt, wenn es Funktionen und Variablen mit gleichem Namen gibt, müssen die Klammern als Kennzeichen der Funktion gesetzt werden. Werden die Klammern weggelassen, interpretiert SDF-P den Namen als Variable und setzt den Wert der Variablen ein, entsprechend den Regeln für die Behandlung von Variablen.

Beispiele

Wenn es keine Variable mit dem Namen USER-IDENTIFICATION gibt, sind die Angaben USER-IDENTIFICATION und USER-IDENTIFICATION( ) gleichwertig; sie werden beide als Funktionsaufrufe interpretiert.

Wenn eine Variable USER-IDENTIFICATION definiert ist, wird nur die Angabe USER-IDENTIFICATION( ) als Funktionsaufruf interpretiert; die Angabe USER-IDENTIFICATION wird in diesem Fall als Variablenname behandelt.

8.1.3 Funktionen mit Eingabeparametern

Syntax

```
function ( [ [ [ parameter = ] wert ], ...] )
```

function
Name der Funktion.

parameter
Name eines Eingabeparameter kann weggelassen werden.

wert
Wert des Eingabeparameters
Schlüsselwort oder Ausdruck, der einem in SDF-P gültigen Datentyp entspricht (STRING, INTEGER, BOOLEAN; zu Ausdruck: siehe Kapitel „Ausdrücke“ auf Seite 255).
Ein Schlüsselwort muss immer mit „*“ beginnen (zur Unterscheidung von Variablennamen).

Beispiele

Schlüsselwort als Parameterwert:

```
DATE(FORMAT=*ISO)
```

Ausdruck als Parameterwert:

```
/ADRESSE = 'hallo'
/U = UPPER-CASE(STRING=ADRESSE)
```

oder:

```
/A = 3
/STR = STRING(190+A)
```

Funktionen können mehrere Eingabeparameter haben, die dann alle dieser Syntax genügen. Entsprechend den Regeln für strukturierte Programmierung und für eine leichte Wartbarkeit von Programmen sollten Parameternamen möglichst angegeben werden, zumindest in Funktionsaufrufen mit mehreren Eingabeparametern.

Schlüsselwort-/Stellungsparameter

Bei Funktionen mit mehreren Eingabeparametern ist zu unterscheiden, ob die Parameter als Schlüsselwort- oder als Stellungsparameter eingesetzt werden.

Schlüsselwortparameter bedeutet, dass der Parametername als Schlüsselwort die Zuweisung identifiziert. Die Reihenfolge der Zuweisungen parameter = wert ist dann beliebig.

Stellungsparameter bedeutet, dass Parametername und Gleichheitszeichen bei der Zuweisung weggelassen werden; es wird nur der Parameterwert angegeben. Die Zuweisung ist dann nur durch ihre Stellung innerhalb der Zuweisungsfolge identifizierbar.

Wenn Parameternamen im Funktionsaufruf nicht angegeben werden, muss die Reihenfolge der Eingabeparameter in den Funktionsbeschreibungen im Kapitel „Vordefinierte Funktionen“ auf Seite 353, beachtet werden. Wenn keine Parameternamen angegeben sind, wertet SDF-P die Parameterwerte im Funktionsaufruf in genau dieser Reihenfolge aus.

Die Regeln für Stellungs- und Schlüsselwortparameter in SDF-P-Funktionen entsprechen den allgemeinen Regeln für Stellungs- und Schlüsselwortparameter bei SDF-Kommandos. Wichtig ist: Stellungsparameter sind nur vor Schlüsselwortparametern zulässig.

Beispiel

In der Syntaxdarstellung der Funktion FILL sind die Eingabeparameter in folgender Reihenfolge aufgeführt: STRING, LENGTH, SIDE, FILL-BYTE. Der Wert des Eingabeparameters SIDE wird über ein Schlüsselwort angegeben (*RIGHT oder *LEFT), die Werte der übrigen Eingabeparameter sind Ausdrücke, die über Variablen oder direkt als einzelnes Zeichen (bei FILL-BYTE), als Zeichenfolge (bei STRING) oder als Zahl (bei LENGTH) angegeben werden können. Die folgenden Funktionsaufrufe sind dabei gleichwertig:

```
/ADRESSE = 'ABCDE'
/B = FILL(STRING=ADRESSE, LENGTH=18, SIDE=*LEFT, FILL-BYTE=C' ')
/B = FILL('ABCDE', 18, *LEFT, C' ')
/B = FILL(LENGTH=18, FILL-BYTE=C' ',STRING=ADRESSE, SIDE=*LEFT)
```

Eingabeparameter können in einem Funktionsaufruf auch mehrfach angegeben werden. Es gilt dann jeweils die letzte Angabe.

Beispiel

```
/D = DATE(FORMAT=*ISO, FORMAT=*GERMAN)
```

Das Datum wird im Format *GERMAN geliefert.

8.1.4 Übernahme von Voreinstellungen

Wenn für einen Eingabeparameter eine Voreinstellung (= Default-Wert) definiert ist, braucht der Parameter im Funktionsaufruf nicht angegeben zu werden, wenn die Voreinstellung übernommen werden soll (der Wert der Voreinstellungen ist in den Syntaxdarstellungen der Funktionen immer unterstrichen).

Für Funktionen mit nur einem Eingabeparameter, z. B. DATE( ), bedeutet dies, dass die Klammern als Kennzeichen des Funktionsaufrufs ausreichen. Sie können sogar entfallen, wenn keine gleichnamigen Variablen definiert sind.

Beispiel

Die folgenden Funktionsaufrufe der Funktion DATE( ) sind gleichwertig, vorausgesetzt, es gibt keine Variable mit dem Namen DATE:

```
/D = DATE(FORMAT=*ISO)
/D = DATE(*ISO)
/D = DATE( )
/D = DATE
```

```
/SHOW-VARIABLE D
D = 1996-05-20141
```

Regeln für Funktionen mit mehreren Eingabeparametern:

– Wenn alle Eingabeparameter als Schlüsselwortparameter angegeben werden, können die Eingabeparameter, deren Voreinstellungen übernommen werden sollen, ohne Ersatz entfallen.
– Wenn alle Eingabeparameter als Stellungsparameter angegeben werden, muss die Reihenfolge der Eingabeparameter beachtet werden. Die Eingabeparameter, deren Voreinstellung übernommen werden soll, müssen durch ein Komma ersetzt werden (Ausnahme: sie stehen am Ende der Parameterliste).
– Werden Stellungs- und Schlüsselwortparameter gemischt, müssen zuerst die Stellungsparameter angegeben werden, dann die Schlüsselwortparameter. In der Liste der Stellungsparameter müssen Eingabeparameter, deren Voreinstellungen übernommen werden sollen, durch Kommas ersetzt werden.

Beispiele

Am Beispiel der Funktion FILL( ) sollen die Regeln für die Übernahme von Voreinstellungen gezeigt werden. Für die Eingabeparameter von FILL( ) gilt folgende Syntax:

FILL
STRING = string_ausdruck ,LENGTH = zahl ,SIDE = *RIGHT / *LEFT ,FILL-BYTE = C' ' / zeichen

Für die beiden Eingabeparameter SIDE und FILL-BYTE sind demnach Voreinstellungen definiert:

– *RIGHT bewirkt Auffüllen nach rechts
– C'␣' bewirkt Auffüllen mit Leerzeichen (nicht mit „leerer Zeichenfolge“ verwechseln (C'')!)

Wenn Sie jetzt beide Voreinstellungen übernehmen und für STRING den Inhalt der Variablen ADRESSE sowie für LENGTH die Zahl 18 einsetzen wollen, sind folgende Funktionsaufrufe gleichwertig:

```
FILL(STRING=ADRESSE, LENGTH=18, SIDE=*RIGHT, FILL-BYTE=C' ')
FILL(STRING=ADRESSE, LENGTH=18,,)
FILL(STRING=ADRESSE, LENGTH=18)
FILL(ADRESSE,18)
```

Wenn Sie die Parameternamen angeben, können Sie auch die Reihenfolge der Eingabeparameter ändern:

```
FILL(LENGTH=18, STRING=ADRESSE)
```

Wenn nur die Voreinstellung für SIDE übernommen und eigene Füllzeichen definiert werden sollen (z. B. Punkte), sind folgende Funktionsaufrufe gleichwertig:

```
FILL(STRING=ADRESSE, LENGTH=18, SIDE=*RIGHT, FILL-BYTE=C'.')
FILL(STRING=ADRESSE, LENGTH=18, FILL-BYTE=C'.')
FILL(ADRESSE, 18, SIDE=*RIGHT, FILL-BYTE=C'.')
FILL(ADRESSE, 18, ,C'.')
```

Wie bei allen Abkürzungsmöglichkeiten ist immer darauf zu achten, dass einem „Nachfolger“ das Einarbeiten in die Prozeduren nicht unnötig erschwert wird. Funktionsaufrufe sollten daher nur so weit gekürzt werden, dass sie noch übersichtlich und leicht verständlich bleiben.

8.1.5 Richtlinien für die Angabe von Werten bei Eingabeparametern

Je nach Angaben von Anführungszeichen und Escape-Zeichen werden bei vordefinierten Funktionen, die Dateinamen bzw. Variablen akzeptieren, verschiedene Aktionen ausgeführt. Im Normalfall passiert dabei dasselbe wie in anderen vordefinierten Funktionen. Es muss jedoch darauf hingewiesen werden, dass das Verhalten nicht mit BS2000-Kommandos (z.B. DVS-Kommandos) vergleichbar ist und darum im ersten Moment zu scheinbar überraschenden Ergebnissen führt.

Beispiel für Funktionen, die Dateinamen akzeptieren

```
/FILE = 'ABC'

/A=IS-CATALOGED-FILE(FILE)      "TESTET, OB 'ABC' KATALOGISIERT IST"
/A=IS-CATALOGED-FILE('FILE')    "TESTET, OB 'FILE' KATALOGISIERT IST"
/A=IS-CATALOGED-FILE('&FILE')   "TESTET, OB 'ABC' KATALOGISIERT IST"
```

Beispiel für Funktionen, die Variablennamen akzeptieren

```
/A = 'B'
/B = 0

/CURRENT-TYPE('A')   "GIBT *STRING ZURUECK"
/CURRENT-TYPE('B')   "GIBT *INTEGER ZURUECK"
```

8.1.6 Abkürzungen von Namen und Schlüsselwörtern

Funktionsnamen dürfen nicht beliebig abgekürzt werden. Allerdings ist für einige Funktionen als Aliasname eine feste Abkürzung definiert; diese ist in der Syntaxdarstellung als Kurzname angegeben.

Beispiel

```
SUBSTRING( )
SUBSTR( )
```

Die Funktion kann sowohl mit dem Namen SUBSTRING als auch mit SUBSTR aufgerufen werden.

Parameternamen können bis zur Eindeutigkeit abgekürzt werden.

Nicht möglich ist eine Abkürzung in der Form „parameter =“. Diese Angabe wird zurückgewiesen.

Schlüsselwörter können entsprechend den SDF-Abkürzungsregeln bis zur Eindeutigkeit abgekürzt werden. Beachten Sie dabei Folgendes:

– Die Eindeutigkeit der Abkürzung ist bei späteren SDF-P-Versionen eventuell nicht gegeben, wenn es neue Schlüsselwörter mit gleicher Abkürzungsmöglichkeit gibt. Daher sollte auf die maximale Abkürzbarkeit verzichtet werden.
– Prozeduren sollten übersichtlich und leicht lesbar sein, dies wird durch die Abkürzung bis zur Eindeutigkeit erschwert.

8.2 Funktionsergebnis

Die zum Lieferumfang von SDF-P gehörenden vordefinierten Funktionen ermitteln aus den Eingabeparametern des Funktionsaufrufs und den aktuellen Umgebungsdaten immer genau einen Rückgabewert. Dieser Wert ist das Funktionsergebnis.

Beispiel

Die Funktion USER-IDENTIFICATION( ) ist eine Funktion ohne Eingabeparameter, sie liefert als Rückgabewert die Benutzerkennung der laufenden Task.

Aufruf:	Rückgabewert:
`USER-IDENTIFICATION( )`	`'US123456'`

Beispiel

Die Funktion DATE( ) hat genau einen Eingabeparameter; sie liefert als Rückgabewert das aktuelle Datum. Der Eingabeparameter FORMAT bestimmt, in welchem Format das Datum übergeben wird, das heißt, wie die Werte für Tag, Monat und Jahr aufeinander folgen.

Aufruf:	Rückgabewert
`DATE(FORMAT=*ISO)`	`'1996-06-24176'`
`DATE(FORMAT=*GERMAN)`	`'24.06.1996'`
`DATE(FORMAT=*AMERICAN)`	`'06/24/96176'`

Beispiel

Die Funktion FILL hat vier Eingabeparameter, auch sie liefert genau einen Rückgabewert, und zwar eine Zeichenfolge, die mit Füllzeichen auf die angegebene Länge aufgefüllt und in der angegebenen Richtung ausgerichtet ist.

Aufruf:	Rückgabewert:
`FILL('ABCDE', 8, SIDE=*RIGHT, FILL-BYTE=C'.')`	`'ABCDE...'`
`FILL('ABCDE', 8, SIDE=*LEFT)`	`'   ABCDE'`

8.3 Funktionsgruppen bei vordefinierten Funktionen

Die zum SDF-P Lieferumfang gehörenden vordefinierten Funktionen lassen sich in Gruppen einteilen.

8.3.1 String-Funktionen

Hier werden alle Funktionen vorgestellt, die Strings bearbeiten oder analysieren.

String bearbeiten

Folgende Funktionen bearbeiten einen oder mehrere Strings und liefern als Ergebnis einen neuen String:

Groß-/ Kleinschreibung	UPPER-CASE( )	setzt Kleinbuchstaben in Großbuchstaben um
	LOWER-CASE( )	setzt Großbuchstaben in Kleinbuchstaben um
Stringlänge	FILL( )	verlängert einen String mit Füllzeichen
	TRIM( )	entfernt gleiche Zeichen am Stringanfang oder -ende
Teilstring	SUBSTRING( )	liefert einen gesuchten Teilstring
	REPLACE( )	ersetzt einen gesuchten Teilstring
SDF-Liste	EXTEND-SDF-LIST( )	fügt einer SDF-Liste ein neues Element hinzu
Neuer Name	RENAME( )	gibt dem angegebenen String einen neuen Namen unter Benutzung von Wildcards
Feldabtrennung	EXTRACT-FIELD( )	trennt von einem String ein Feld ab

String analysieren

Die folgenden Funktionen analysieren einen gegebenen String:

Anfangsposition	INDEX( )	sucht die Position eines Teilstrings im Gesamtstring
Stringlänge	LENGTH( )	bestimmt die Länge eines Strings
Zeichensuche	VERIFY( )	prüft, ob gegebene Zeichen im angegebenen String enthalten sind
Muster	WILDCARD( )	prüft, ob ein String ein bestimmtes Muster enthält
C-Literal	IS-C-LITERAL( )	prüft, ob ein String ein C-Literal ist
X-Literal	IS-X-LITERAL( )	prüft, ob ein String ein X-Literal ist

Zahl	IS-INTEGER( )	prüft, ob ein String eine ganze Zahl darstellt, das heißt, ob der String in einen Integer-Wert konvertiert werden kann
SDF-Struktur	SDF-STRUCTURE-VALUE( )	liefert den Wert einer SDF-Struktur als String
	IS-SDF-STRUCTURE()	prüft, ob der String eine SDF-Struktur ist
SDF-Liste	IS-SDF-LIST( )	prüft, ob der String eine SDF-Liste ist (eine SDF-Liste ist ein String, der nach den syntaktischen Regeln für Operandenlisten in Kommandos interpretiert wird)
	SUBLIST( )	liefert ein Element einer SDF-Liste
	SUBLIST-NUMBER( )	liefert die Anzahl der Elemente einer SDF-Liste
SDF-Datentyp	CHECK-DATA-TYPE( )	prüft, ob der String die SDF-Datentyp-Bestimmungen erfüllt
Listenvariable	SEARCH-LIST-INDEX( )	sucht String (auch regulärer POSIX-Ausdruck) in einer Listenvariablen

Namen prüfen

Die folgende Funktion prüft, ob der angegebene String den geforderten Namenskonventionen genügt:

Variablenname	IS-VARIABLE-NAME()	

Funktionen für Variablenzugriffe

Die in diesem Abschnitt vorgestellten Funktionen liefern als Rückgabewert einen Variablennamen, den Inhalt einer Variablen oder Angaben zum Aufbau der Variablen. Sie ermöglichen die Bearbeitung von zusammengesetzten Variablen.

Zusammengesetzte Variable	FIRST-VARIABLE-NAME( )	liefert den Variablennamen des ersten Variablenelements
	NEXT-VARIABLE-NAME( )	liefert den Variablennamen des folgenden Variablenelements
Zahl der Variablenelemente	SIZE( )	liefert die Anzahl der Elemente einer zusammengesetzten Variablen
Obere Grenze für Listengröße	LIMIT( )	gibt aus, wie viele Elemente die angegebene Listenvariable enthalten darf
Wert eines Arrayindex	ARRAY-INDEX( )	liefert den Wert eines Arrayindex, d. h. des Arrayindex, der den angegebenen Bedingungen genügt

Variablenmerkmale

Variablentyp	CURRENT-TYPE( )	liefert den aktuellen Datentyp einer einfachen Variablen
Attributausprägung	VARIABLE-ATTRIBUTE( )	liefert den Wert des angegebenen Attributs
Variablendeklaration	IS-DECLARED( )	prüft, ob eine Variable deklariert ist
Initialisierung	IS-INITIALIZED( )	prüft, ob eine Variable einen gültigen Wert enthält
Geltungsbereich	LAYOUT-SCOPE( )	liefert den Geltungsbereich eines Strukturlayouts

8.3.2 Umgebungsinformationen

Mit den Funktionen, die in diesem Abschnitt vorgestellt werden, lassen sich Informationen über die aufrufende Task, den Job, die Prozedur, über Benutzerschalter, Jobvariablen, Systemoptionen etc. abrufen. Die meisten dieser Funktionen werden ohne Eingabeparameter aufgerufen, da die Zuordnung zum abzufragenden Wert eindeutig ist.

Informationen über Job / Task / Dateien

Jobname	JOB-NAME( )	liefert den Jobnamen der laufenden Task; das ist der Name, der beim SET-LOGON-PARAMETERS angegeben wurde
Betriebsmodus	TASK-MODE( )	gibt aus, in welchem Betriebsmodus sich die aktuelle Task befindet
TSN	TSN( )	liefert die Auftragsnummer (TSN) der aktuellen Task
Priorität	RUN-PRIORITY( )	liefert die Prioritätsstufe der aktuellen Task
Abrechnungsnummer	ACCOUNT( )	liefert die Abrechnungsnummer der Task
Benutzerkennung	USER-IDENTIFICATION( )	liefert die Benutzerkennung der laufenden Task
Default-Catid	STD-CAT-ID( )	liefert die Katalogkennung des Default-Pubsets der Benutzerkennung der aktuellen Task
Home-Pubset	HOME-CAT-ID( )	liefert die Katalogkennung (Catid) des Home-Pubsets der Benutzerkennung der aktuellen Task
Katalogeintrag	IS-CATALOGED-FILE( )	prüft, ob die angegebene Datei katalogisiert ist
	IS-CATALOGED-JV( )	prüft, ob die angegebene Jobvariable katalogisiert ist
	IS-LIBRARY( )	prüft, ob die angegebene Datei eine Bibliothek ist
	IS-LIBRARY-ELEMENT( )	prüft, ob das angegebene Bibliothekselement existiert
Aufrufe zählen	COUNTER( )	zählt die Aufrufe von COUNTER( )
Programmname	PROG-NAME( )	liefert den Namen der aktuell geladenen Programmdatei
Pfadname	INSTALLATION-PATH( )	gibt für eine Datei den Pfadnamen gemäß der Produktversion an.
Dateiinhalt	IS-EMPTY-FILE( )	prüft, ob die Datei leer ist

SYSFILE-Management: Systemdateien

SYSCMD	SYSCMD( )	liefert *PRIMARY oder den Namen der Datei, der SYSCMD zugewiesen ist
SYSDTA	SYSDTA( )	liefert *PRIMARY oder *SYSCMD oder den Namen der Datei, der SYSDTA zugewiesen ist
SYSLST	SYSLST( )	liefert *PRIMARY oder den Namen der Datei, der SYSLST zugewiesen ist
SYSOUT	SYSOUT( )	liefert *PRIMARY oder den Namen der Datei, der SYSOUT zugewiesen ist

Datum/Zeit

Datum	DATE( )	liefert das aktuelle Tagesdatum im angegebenen Format
	DATE-VALUE( )	liefert ein bestimmtes Tagesdatum im angegebenen Format
	ELAPSED-DAYS( )	liefert die Anzahl der Differenztage zwischen zwei Datumsangaben
Wochentag	DAY( )	liefert den Namen des aktuellen Tages in der angegebenen Sprache als Abkürzung
Monat	MONTH( )	liefert den Namen des aktuellen Monats in der angegebenen Sprache als Abkürzung
Uhrzeit	TIME( )	liefert die aktuelle Uhrzeit auf die Sekunde genau, mit beliebigem Trennzeichen zwischen den verschiedenen Einheiten

Systemdaten

In diesem Abschnitt werden Funktionen zusammengefasst, die Systemdaten liefern, die sich auf die eingesetzte Hard- und Software beziehen sowie auf die Einstellungen, die der Systemverwalter für das laufende System getroffen hat. Nicht enthalten sind taskbezogene Daten oder Jobvariablen-Informationen.

Rechnername	`HOST( )`	liefert den internen Namen des BS2000-Rechners, auf dem die aktuelle Task läuft
Abfrage: SDF-P im System	`IS-SDF-P( )`	liefert TRUE, wenn SDF-P im System geladen ist
Systemparameter	`SYSTEM-INFORMATION( )`	liefert die Werte von Systemparametern; Eingabeparameter wie im Makro SINF
System-ID	`SYS-ID( )`	liefert das Systemkennzeichen
Systemlaufnummer	`SESSION-NUMBER( )`	liefert die interne Nummer der aktuellen Session
SDF-P-Version	`SDF-P-VERSION( )`	liefert die Versionsbezeichnungen der geladenen Subsysteme SDF-P und SDF-P-BASYS

TIAM-Information

Stationsname	`STATION( )`	liefert den Stationsnamen der TIAM-Station
Gerätetyp	`STATION-TYPE( )`	liefert den Gerätetyp der TIAM-Station
Prozessorname	`PROCESSOR( )`	liefert den Prozessornamen der TIAM-Station

Prozedurinformationen

Die folgenden Funktionen prüfen die Einstellung von Prozedur-Eigenschaften und liefern Informationen über die aktuelle Prozedur:

Schachtelungstiefe	PROC-LEVEL( )	liefert die Schachtelungstiefe der S-Prozedur zum Zeitpunkt des Funktionsaufrufs
Protokollierung	LOGGING-MODE( )	zeigt an, ob für die aktuelle S-Prozedur Protokollierung von Kommandos oder Daten eingeschaltet ist
Spinoff	STMT-SPINOFF( )	zeigt an, ob Statement-Spinoff eingeschaltet ist
Benutzerschalter	USER-SWITCH( )	fragt den Zustand des angegebenen Benutzerschalters ab
Aufruf	EXPLICIT-CALL( )	liefert die Art des Prozeduraufrufs
	SYSTEM-CALL( )	liefert die Syntaxdatei-Hierarchiestufe für das prozeduraufrufende Kommando

Jobvariablen

Die folgenden Funktionen liefern Informationen über Jobvariablen, die den Auftrag oder das Programm überwachen. Jobvariablen gehören zum (kostenpflichtigen) Produkt JV (Jobvariablensystem) und wurden in den bisherigen BS2000-Prozeduren zur Prozedurüberwachung und -Steuerung genutzt:

Auftragsüberwachung	JOB-MONJV( )	liefert den Namen der Monitor-JV, die den aktuellen Auftrag überwacht
Programmüberwachung	PROG-MONJV( )	liefert den Namen der Monitor-JV, die das laufende Programm überwacht
Inhalt	JV( )	liefert den Inhalt der angegebene Jobvariablen
Klasse	JOB-CLASS( )	liefert die Jobklasse der aktuellen Task
Katalogeintrag	IS-CATALOGED-JV( )	prüft, ob die angegebenen Jobvariable katalogisiert ist

8.3.3 Konvertierungsfunktionen

Die in diesem Abschnitt vorgestellten Funktionen stehen für explizites Konvertieren zur Verfügung.

String-Konvertierung für Literale, C-Literale, X-Literale

BOOLEAN( )	konvertiert einen Ausdruck nach BOOLEAN
FROM-C-LITERAL( )	konvertiert ein C-Literal in den String, der durch das Literal dargestellt wird (inverse Funktion zu TO-C-LITERAL)
FROM-X-LITERAL( )	konvertiert ein X-Literal in den String, der durch das Literal dargestellt wird (inverse Funktion zu TO-X-LITERAL)
HASH-STRING( )	verschlüsselt einen Ausdruck als String
HASH-VALUE( )	verschlüsselt einen Ausdruck als Integer-Wert
INTEGER( )	konvertiert einen Ausdruck nach INTEGER
INTEGER-TO-X-LITERAL( )	konvertiert eine Dezimalzahl in ein 4 Byte langes X-Literal, das die Codierung der Zahl enthält (inverse Funktion zu X-LITERAL-TO-INTEGER)
STRING( )	konvertiert einen Ausdruck nach STRING
TO-C-LITERAL( )	konvertiert einen String in ein C-Literal
TO-X-LITERAL( )	konvertiert einen String in die externe hexadezimale Darstellung des Strings
VARIABLE-TO-STRING( )	konvertiert eine S-Variable in einen SDF-String
X-LITERAL-TO-INTEGER( )	konvertiert einen maximal 4 Byte langen String in eine Dezimalzahl (inverse Funktion zu INTEGER-TO-X-LITERAL)

Zeichenweise (um)kodieren

CHARACTER-TO-INTEGER( )	liefert für das gegebene Zeichen den Wert im EBCDI-Code als Integer-Zahl
INTEGER-TO-CHARACTER( )	liefert für einen gegebenen Integer-Wert zwischen 1 und 255 das Zeichen, das im EBCDI-Code mit diesem Wert codiert ist
TRANSLATE( )	ersetzt einen String durch einen vom Benutzer definierten anderen String
TRANSLATE-BOOLEAN( )	ordnet dem Ergebnis eines booleschen Ausdrucks einen vom Benutzer definierten anderen Ausdruck zu

8.3.4 Kommando-Returncodes/Fehlermeldungen

Die SDF-P- und SDF-Kommandos liefern einen standardisierten Kommando-Returncode, der aus drei Komponenten besteht. Diese Komponenten zeigen an, wie das Kommando ausgeführt wurde oder wo ein Fehler auftrat.

Der Kommando-Returncode besteht aus folgenden drei Komponenten: Subcode1 (SC1) bezeichnet die Fehlerklasse, Subcode2 (SC2) liefert Zusatzinformationen zur Fehlerklasse, Maincode liefert einen sieben Byte langen Fehlerschlüssel, dem ein Meldungstext zugeordnet ist.

Subcode1 abfragen	`SUBCODE1( )`	Liefert als Ergebnis den Wert 0, wenn noch kein Fehler aufgetreten ist, oder die Fehlerklasse des zuletzt aufgetretenen Fehlers; das heißt den Subcode1 des letzten Kommandos, das nicht ordnungsgemäß ausgeführt wurde.
Subcode2 abfragen	`SUBCODE2( )`	Liefert die Zusatzinformation für die Fehlerklasse. SUBCODE2( ) braucht also nur dann aufgerufen zu werden, wenn SUBCODE1( ) einen Wert ungleich 0 geliefert hat.
Maincode abfragen	`MAINCODE( )`	Liefert den 7 Byte langen Fehlerschlüssel, der die Meldungsklasse und den Fehler genau beschreibt. Die Fehlermeldung zu diesem Schlüssel kann mit der Funktion MSG( ) oder mit dem Kommando HELP-MSG angefordert werden.
Fehlermeldungstext anfordern	`MSG( )`	Liefert den Meldungstext, der dem angegebenen Meldungsschlüssel zugeordnet ist (in der angegebenen Sprache).

8.4 Systemverwalter-Funktionen

Voraussetzung für die Erstellung von benutzereigenen bzw. Systemverwalter-Funktionen ist, dass das Subsystem SDF-P-BIF vom Systemverwalter geladen worden ist. Darin sind die Werkzeuge bzw. Makros enthalten, um selbst Funktionen schreiben und installieren zu können.

Es gibt eine Assembler-Schnittstelle, mit der der Systemverwalter Funktionen erstellen kann. Diese ist im Kapitel „Programmschnittstellen“ auf Seite 311 beschrieben.

Da nur der Systemverwalter benutzereigene Funktionen entwickeln kann, ist er auch für die Korrektheit und die Sicherheit dieser Funktionen verantwortlich.

8.4.1 Namenskonventionen

Der Name der Systemverwalter-Funktion darf nicht identisch mit einer bereits oder zukünftig bestehenden Funktion sein, da es sonst zu Inkompatibilitäten kommen kann. Darum sollten Namen selbsterstellter Funktionen mit einem „X-“ beginnen analog zu den mit SDF-A selbstdefinierten Kommandos, also z.B. X-MY-BUILT-IN.

Im Übrigen dürfen Namen von Funktionen allgemein nicht länger als 20 Zeichen sein.

8.4.2 Programme erstellen

Programme, mit denen selbsterstellte Funktionen implemetiert werden, enthalten zwei verschiedene Teile:

1. Die Syntaxbeschreibung
   Hier wird die Syntax der Funktionsparameter beschrieben. Der Code wird im Makro BIFDESC generiert.
2. Die Ausführungsbeschreibung
   Hier ist der Code der Funktion enthalten. Vom System wird eine Parameterliste an die Funktion übergeben, die ein Array von n Sätzen enthält (string_length, string_ptr, value_type); n ist dabei die Anzahl der Funktionsparameter plus zwei (function_value, returncode). Die Anzahl der Funktionsparameter kann allerdings nicht größer als 254 sein. Im Makro BIFMDL sind die Sätze mit Werten sowie die Datenstrukturen enthalten.

8.4.3 BIFTAB und Objekte aktualisieren

Wenn ein Programm für eine Funktion geschrieben wurde, muss der Systemverwalter den BIFTAB-Source, der in SYSSRC.SDF-P-BIF.010 enthalten ist, aktualisieren. Der Makro BIFDEF generiert den Tabelleneintrag, der den Namen der Funktion mit den Adressen des Ausführungsmodul-Eintrags und des Syntaxbeschreibungs-Eintrags verbindet.

Das Objekt, das für dieses BIFTAB-Modul generiert wird, muss der Datei SYSLNK.SDF-P-BIF.010 angefügt werden. Damit wird auch ein früheres BIFTAB-Modul gelöscht. Geladen wird dieses Modul beim nächsten Start des Subsystems SDF-P-BIF durch einen automatischen Link.

Die Objekte, die für die Syntaxbeschreibung und den Code der Funktion generiert wurden, müssen auch der Datei SYSLNK.SDF-P-BIF.010 angefügt werden. Sie werden auch beim nächsten Start des Subsystems SDF-P-BIF durch einen automatischen Link geladen, aber nur wenn die Einträge im Modul BIFTAB definiert sind.

Auf die hinzugefügten Funktionen kann nach dem nächsten Start des Subsystems SDF-P-BIF zugegriffen werden.

8.4.4 Parameterübergabe

Die Parameterübergabe wird durch Register realisiert.

Alle Eingabeparameter sowie Rückgabewert und Returncode werden in einer Struktur beschrieben (Adresse, Länge, Typ).

Neben expliziten Typen (String, Integer und Boolean) können auch Schlüsselwörter als Eingabeparameter angegeben werden.

Näheres siehe im Kapitel „Programmschnittstellen“ auf Seite 311.

Hinweis

Systemverwalter-Funktionen können nur im TPR-Modus aufgerufen werden. Ansonsten verhalten sie sich beim Aufruf wie vordefinierte Funktionen.

8.4.5 Beispiele

Es soll die folgende Funktion in Assembler implementiert werden:

```
XSUBSTRING[2/3](STRING = <string_ausdruck>
                ,START = <arithm_ausdruck>
                ,LENGTH = <arithm_ausdruck> / *REST-LENGTH
               )
```

Assembler-Schnittstelle

Titel der Funktion:	XSUBSTRING2
Entry:	XSUBEX2
Eingabedaten:	STRING, START POSITION, LENGTH
Ausgabedaten:	SUBSTRING

Erster Schritt : Syntaxbeschreibung

```
BIFDESC     NAME='XSUBSTRING2',                -
            ENTRYN=(*CSECT,XSUBDEF2),          -
            PARLIST=                           -
             ((STRING,*STRING),                -
               (START,*INTEGER,1),             -
               (LENGTH,*INTEGER,*REST-LENGTH,  -
                *REST-LENGTH)                  -
               ),                              -
            PARFORM=BY-VALUE,
            VALTYPE=*STRING
END
```

Zweiter Schritt: Ausführungsprogramm

```
XSUBEX2   CSECT
XSUBEX2   AMODE   ANY
XSUBEX2   RMODE   ANY
          STM     14,12,12(13)       Speichere die Register des Aufrufers
          BASR    10,0               R10: Basisregister
          USING   *,10
          USING   DSMDL1,4
          USING   DSMDL2,5
*
          L       1,0(0,1)           R1 > a(p1)   ?? R1=a(p1)
*
          LA      5,48(1)            R5 > fünftes Element der Operanden-
                                     liste (z.B. RC)
          L       5,4(5)             R5 > RC
          MVC     0(9,5),OKRC        Set RC = OK
*
          LA      4,0(1)             bei R4 beginnt Operandenliste
          L       6,BIF1VLG          R6: Länge des Operanden STRING
          L       7,BIF1VPT          R7: Adresse des Operanden STRING
          LTR     6,6                Ist die Länge von STRING=0?
          BE      MSGNULL            Sende eine Meldung NILL_STRING
*
          LA      4,12(1)            bei R4 folgt zweites Element der
                                     Operandenliste (z.B. START)
          LA      2,1(0,0)           R2 = 1
          L       8,BIF1VPT          R8: Adresse des Werts START
          L       8,0(8)             R8: Wert von START
          CR      8,2                Wenn START < 1
          BL      MSGBOUN            Sende Meldung OUT_OF_BOUNDS
          CR      8,6                Wenn START > Länge von STRING
          BH      MSGBOUN            Sende Meldung OUT_OF_BOUNDS
*
          LA      4,24(1)            bei R4 folgt drittes Element der
                                     Operandenliste (z.B. LENGTH)
*                                    R3 = Länge von STRING - START + 1
          LA      3,1(0,0)           R3 = 1
          AR      3,6                R3 = R3 + Länge von STRING
          SR      3,8                R3 = R3 - START
          CLI     BIF1VTY,BIF1INT
          BE      LENINT
LENKEYW   DS      0H
*                                    *REST-LENGHT ist gesucht
          LR      9,3
          B       @0001
```

```
LENINT     DS      OH
*                                      LENGTH = *INTEGER
           L       9,BIFTVPT           R9: Adresse des Werts LENGHT
           L       9,0(9)              R9: Wert von LENGTH
           CR      9,2                 Wenn LENGTH < 1
           BL      MSGBOUN             Sende Meldung OUT_OF_BOUNDS
           CR      9,3                 Wenn LENGTH < REST-LENGTH
           BNH     @0001               Dann ok
           LR      9,3                 LENGTH abgeschnitten
           MVC     0(9,5),TRUNCRC      Setze RC = TRUNCATED
@0001      DS      OH
                                       Berechne den Funktionswert
           LA      4,36(1)             bei R4 folgt viertes Element der
                                       Operandenliste (z.B. FUNCTION_VALUE)
           L       6,BIF1VPT           R6 : Funktionwertadresse
           AR      8,7                 R8 = Adresse von STRING + START
           SR      8,2                 R8 = R8-1
           LR      7,9                 R7 = Länge des Funktionswerts
           ST      7,BIF1VLG
           MVCL    6,8                 Kopiere in Funktionswert
           B       RETURN
*
MSGNULL    MSG7X   MF=E,PARAM=NULLPL
           MVC     0(9,5),NULLRC       Setze RC = NULL_STRING
           B       RETURN
*
MSGBOUN    MSG7X   MF=E,PARAM=BOUNDPL  Setze RC = OUT OF BOUNDS
           MVC     0(9,5),BOUNDRC
*
RETURN     LM      14,12,12(13)        Register (Aufrufer) zurückschreiben
           BR      14                  Zurück zum Aufrufer
OKRC       DC      XL1'00'
           DC      XL1'00'
           DC      CL7'CMD0001'
TRUNCRC    DC      XL1'02'
           DC      XL1'00'
           DC      CL7'SDP0414'
BOUNDRC    DC      XL1'00'
           DC      XL1'01'
           DC      CL7'SDP0412'
NULLRC     DC      XL1'00'
           DC      XL1'01'
           DC      CL7'SDP0411'
           DS      0F
```

```
NULLPL      MSG7X    MF=L,ID=SDP0411
BOUNDPL     MSG7X    MF=L,ID=SDP0412
            DS       0F
DSMDL1      BIFMDL1  MF=D
            DS       0F
DSMDL2      BIFMDL2  MF=D
            END
```

Dritter Schritt: BIFTAB aktualisieren

```
BIFTAB      CSECT
            .....
XSUBSTR2    BIFDEF   MF=L,NAME=XSUBSTRING2,SYNTAX=XSUBDEF2,CODE=XSUBEX2
            .....
            END
```

9 Ausdrücke

Ausdrücke legen fest, wie aus gegebenen Werten ein neuer Wert berechnet wird. Die Operanden, die über Operatoren verknüpft werden, können Basisterme oder wieder Ausdrücke sein. Basisterme sind innerhalb eines Ausdrucks Terme, die nicht weiter zerlegt werden können, also keine Operatoren enthalten.

Im einfachsten Fall besteht ein Ausdruck nur aus einem Operanden; der Wert des Ausdrucks ist dann der Wert des Operanden. Solche Ausdrücke und Ausdrücke, die nur aus einem Operator bestehen, dessen Operanden Basisterme sind, werden als „einfache Ausdrücke“ bezeichnet. Im Gegensatz dazu stehen zusammengesetzte Ausdrücke, bei denen mindestens einer der Operanden wieder ein Ausdruck ist. Dieser Ausdruck kann wieder einfach oder zusammengesetzt sein.

Unter dem Oberbegriff „Basisterme“ werden zusammengefasst:

– Zahlen
– Boolesche Konstanten
– String-Literale
– Variablennamen
– Funktionsaufrufe

Die Operatoren werden in vier Kategorien eingeteilt:

– Arithmetische Operatoren
– Logische Operatoren
– Vergleichsoperatoren
– Verkettungsoperator

Der Datentyp eines Ausdrucks ohne Operatoren wird durch den Datentyp des Basisterms bestimmt, ansonsten durch die Kombination der Operatoren. Es sind dadurch drei Typen von Ausdrücken zu unterscheiden:

– Arithmetische Ausdrücke
– Logische bzw. boolesche Ausdrücke
– String-Ausdrücke

In diesem Kapitel werden zunächst die Basisterme beschrieben, dann die Operatoren, die Ausdruckstypen und schließlich die Syntax-, Interpretations- und Auswertungsregeln für Ausdrücke.

9.1 Basisterme

Basisterme sind die Terme innerhalb eines Ausdrucks, die nicht weiter zerlegt werden können, d. h. Terme, die keinen Operator mehr enthalten.

Die folgende Tabelle stellt die verschiedenen Basisterme und ihre Darstellung in der Metasyntax einander gegenüber:

Basisterm	**Darstellung**
Zahl	<integer>
String-Literale	<c-string> / <x-string>
Boolesche Konstante	<boole-const>
Variablenname	<composed-name>
Funktionsaufruf	<functioncall>

Diese Basisterme werden in den folgenden Abschnitten detailliert beschrieben.

9.1.1 Zahl

In Ausdrücken sind nur ganze Zahlen erlaubt. Für den Wert, der durch die Zahl repräsentiert wird, wird der Begriff Integer-Wert verwendet (dementsprechend werden die ganzen Zahlen manchmal auch als „Integer“ bezeichnet).

Datentyp	<integer>
Zeichenvorrat	0 ... 9, +, -
Wertebereich	$-2^{31} \leq \text{zahl} \leq +2^{31}-1$

Zahlen dürfen nicht in Hochkommas eingeschlossen werden, da Zeichenfolgen in Hochkommas als Strings interpretiert werden.

Zahlen können über arithmetische Operatoren und Vergleichsoperatoren miteinander verknüpft werden.

Zahlen können auch als Inhalt von Variablen, als Ergebnis eines Funktionsaufrufs oder Ergebnis eines Ausdrucks in einen neuen Ausdruck eingehen.

Beispiel

Korrekte Integer-Zuweisungen:

```
/A = -12345
/B = 3456
/C = +1287
```

Die Variablen A, B und C werden mit den angegebenen numerischen Werten initialisiert.

Falsche Integer-Zuweisungen:

```
/D = +123,5
/E = '+1234'
```

Bei der Initialisierung der Variablen D meldet SDF-P einen Syntaxfehler im Variablennamen (+123,5 wird nicht als Wert interpretiert). Die Variable E schließlich wird mit dem String '+1234' initialisiert; sobald diese Variable in einer arithmetischen Operation eingesetzt wird, meldet SDF-P einen falschen Datentyp. (Vorausgesetzt E hat den Typ ANY oder STRING. Sonst tritt schon bei der Zuweisung ein Fehler auf.)

9.1.2 Boolesche Konstanten

Es gibt zwei boolesche Konstanten: TRUE oder FALSE. Als Synonyme sind außer TRUE und FALSE auch YES / NO und ON / OFF zulässig. Diese Namen sind reservierte Namen und können daher nicht als Variablennamen eingesetzt werden. Die folgende Tabelle definiert den Datentyp und zeigt, welche Namen für boolesche Konstanten gültig sind:

Datentyp	<boole-const>
Wertebereich	TRUE, FALSE
Zeichenvorrat	TRUE, YES, ON, FALSE, NO, OFF

Boolesche Konstanten dürfen nicht in Hochkommas eingeschlossen werden, da Zeichenfolgen in Hochkommas als Strings interpretiert werden.

Boolesche Konstanten können über Vergleichs- oder logische Operatoren mit logischen Ausdrücken verknüpft werden.

Boolesche Konstanten werden direkt angegeben. Boolesche Variablen, Ausdrücke oder Funktionen liefern als Ergebnis boolesche Werte, die entweder gleich TRUE oder gleich FALSE sind.

Beispiel

```
/DECLARE-VARIABLE SCHALTER-1(TYPE=*BOOLEAN)
/DECLARE-VARIABLE SCHALTER-2(TYPE=*ANY)
/SCHALTER-1 = ON     "korrekte Zuweisung"
/SCHALTER-2 = 'OFF'  "falsche Zuweisung"
```

Beide Zuweisungen sind syntaktisch korrekt, jedoch nur SCHALTER-1 kann später in logischen Operationen als boolesche Konstante eingesetzt werden. SCHALTER-2 wurde ein String zugewiesen; dementsprechend kann SCHALTER-2 nur in Vergleichs- oder String-Operationen eingesetzt werden; andernfalls meldet SDF-P einen falschen Datentyp.

9.1.3 String-Literale

String-Literale sind Folgen beliebiger Zeichen, eingeschlossen in Hochkommas. In der Literatur werden String-Literale auch als Zeichenketten bezeichnet; in diesem Handbuch sind beide Begriffe aber nicht identisch. So muss eine Zeichenkette nicht in Hochkommas eingeschlossen sein. Darum ist z.B. ABC eine Zeichenkette und 'ABC' ein String-Literal.

Zu unterscheiden sind bei String-Literalen zwei Darstellungsarten: als C-String und X-String.

Datentyp	**<c-string>**	**<x-string>**
Zeichenvorrat	alle EBCDIC-Zeichen	Hexadezimal-Ziffern (0 ... F)
Länge	beliebig	beliebig
Darstellung	[C]'.....'	X'.....'
Leerstring	[C]''	X''

Hinweis

Die internen Darstellungen von C- und X-Strings sind gleich, d.h. jeder C-String lässt sich auch als X-String darstellen.

Die Begriffe C-String und X-String beziehen sich darauf, wie die Bytes, aus denen die Zeichenkette besteht, dargestellt werden:

– Im C-String wird jedes Byte durch sein EBCDIC-Zeichen dargestellt (Zeichen = Character, Abkürzung: C). Daraus ergibt sich als Zeichenvorrat der gesamte EBCDI-Code; dies bedeutet auch, dass Groß- und Kleinschreibung erhalten bleiben.
– Im X-String wird jedes Byte durch die sich ergebenden EBCDIC-Zeichen der Halbbytes dargestellt; im X-String folgen also paarweise aufeinander die Darstellung von linkem und rechtem Halbbyte. Entsprechend ergeben sich als Zeichenvorrat die Ziffern des hexadezimalen Zahlensystems, das heißt die Ziffern von 0 bis F.

– Wird für einen X-String eine ungerade Anzahl Sedezimalziffern angegeben, wird er intern von links mit einer Null aufgefüllt. (Beispiel: X'123' wird zu X'0123')
– Ein Sonderfall ist der Leerstring: Er besteht nur aus den paarweisen Hochkommas; ein C oder X kann vorangestellt sein (C'' / X''). Der Leerstring darf nicht mit dem Leerzeichen-String verwechselt werden (C'␣' / X'40').

Strings sind immer in Hochkommas eingeschlossen; umgekehrt bedeutet dies: Zeichenketten und Zahlen, die in Hochkommas eingeschlossen sind, werden als Strings interpretiert.

Ist dem Hochkomma kein Zeichen vorangestellt, gilt der String standardmäßig als C-String. Dem X-String muss ein X vorangestellt sein. Andere Buchstaben sind nicht zulässig und führen zum Fehler.
Strings können über Vergleichsoperatoren oder Verkettungsoperatoren miteinander verknüpft werden.
Strings können auch als Inhalt von Variablen, als Ergebnis eines Funktionsaufrufs oder Ergebnis eines Ausdrucks in einen neuen Ausdruck eingehen.

Hinweis

Die Bezeichnung „string_ausdruck“, die als Parameterwert in vordefinierten Funktionen verwendet wird, steht für einen der folgenden Werte:
– eine in Hochkommas eingeschlossene Zeichenfolge (<c-string>)
– den Namen einer Variablen, die eine Zeichenkette enthält (<composed-name>)
– einen Ausdruck, der als Ergebnis eine Zeichenkette zurückliefert

Beispiel

```
/JV-NAME = 'MY-JV'
/MY-VAR = JV('JV-NAME')      Wert der Jobvariablen JV-NAME
/MY-VAR = JV(JV-NAME)        Wert der Jobvariablen MY-JV
/MY-VAR = JV('&JV-NAME')     Wert der Jobvariablen MY-JV
/MY-VAR = JV(JV(JV-NAME))    Wert der Jobvariablen, deren Name in der Job-
                             variablen MY-JV abgelegt ist
```

Beispiel

```
/A = 'ABCD'
/B = C'ABCD'
/C = X'C1C2C3C4'
```

Den Variablen A und B wird der gleiche C-String zugewiesen. Der Variablen C wird ein X-String zugewiesen, der ausgewertet die Zeichenfolge ABCD ergibt. Alle drei Variablen haben somit denselben Inhalt.

9.1.4 Variablenname

Variablennamen, die eine einfache Variable bezeichnen, können Bestandteil eines Ausdrucks sein. Voraussetzung ist, dass diese Variable bereits initialisiert ist, also einen gültigen Inhalt hat.

Für die Bildung von Variablennamen gelten die Regeln, die im Abschnitt „Variablennamen“ auf Seite 155 beschrieben sind.

In die Berechnung des Ausdrucks geht nicht der Variablenname ein, sondern der Inhalt der Variablen.

Der Inhalt der Variablen kann vom Datentyp INTEGER, BOOLEAN oder STRING sein; das heißt, er ist eine Zahl, ein boolescher Wert oder ein String. Dementsprechend gelten für die Auswertung des Variableninhalts die Regeln, wie sie in den vorhergehenden Abschnitten für Zahlen, boolesche Konstanten und Strings beschrieben wurden.

Damit der Variableninhalt ausgewertet werden kann, dürfen Variablennamen nicht in Hochkommas eingeschlossen werden. Werden sie dennoch in Hochkommas eingeschlossen, wird nicht der Variableninhalt ausgewertet, sondern der Variablenname als String interpretiert.

Beispiel

Es gelten folgende Variablendeklarationen und Variablenzuweisungen:

```
/SET-VARIABLE A = 36
/SET-VARIABLE B = 72
/DECLARE-VARIABLE C
```

Diese Variablen werden in (einfachen) Ausdrücken verwendet:

```
/D = A + B
```
„richtig“

```
/E = A - C
```
„falsch“

Die erste Zuweisung ist gültig, da die Variablen A und B gültig initialisiert sind. Die zweite Zuweisung, mit der die Variable E initialisiert werden soll, ist ungültig, da die Variable C zwar deklariert, aber nicht initialisiert ist.

9.1.5 Funktionsaufruf

Für den Einsatz von Funktionsaufrufen in Ausdrücken gelten ähnliche Regeln wie für den Einsatz von Variablennamen: In die Berechnung des Ausdrucks geht nicht der Funktionsname ein, sondern das Ergebnis, das die so aufgerufene Funktion liefert.

Wie Funktionen aufgerufen werden und welche Funktionen in strukturierten Prozeduren zur Verfügung stehen, ist im Kapitel „Funktionen“ auf Seite 229 beschrieben.

Das Ergebnis der Funktion kann vom Datentyp INTEGER, BOOLEAN oder STRING sein; das heißt es ist eine Zahl, ein boolescher Wert oder ein String. Dementsprechend gelten für die Funktionsergebnisse die Regeln, wie sie in den vorhergehenden Abschnitten für Zahlen, boolesche Konstanten und Strings beschrieben wurden.

Damit das Ergebnis des Funktionsaufrufs ausgewertet werden kann, dürfen Funktionsaufrufe nicht in Hochkommas eingeschlossen werden. Werden sie dennoch in Hochkommas eingeschlossen, wird der Name der Funktion als String interpretiert.

Bei Namensgleichheit von Variablen und Funktionen wird die Variable ausgewertet, entsprechend der Regel für S-Prozeduren. Verwechslungen können vermieden werden, indem im Funktionsaufruf die kennzeichnenden Klammern angegeben werden.

9.2 Operatoren

Operatoren verknüpfen Basisterme zu einfachen Ausdrücken; diese können wiederum durch Operatoren zu zusammengesetzten Ausdrücken verknüpft werden usw.

Da einige Operatoren auch als Sonderzeichen in Namen, zum Beispiel in Variablennamen zugelassen sind, sollten Operatoren generell in Leerzeichen eingeschlossen werden. Andernfalls können einige Operatoren nicht korrekt interpretiert werden; zum Beispiel kann das Minuszeichen (-) als Bindestrich interpretiert werden.

9.2.1 Arithmetische Operatoren

Über arithmetische Operatoren werden Rechenoperationen an Zahlen durchgeführt. Arithmetische Operatoren verknüpfen demnach Zahlen. Zahlen können angegeben werden als numerische Literale (dies entspricht einer numerischen Konstanten) oder Variablen, die einen gültigen numerischen Wert enthalten. In zusammengesetzten Ausdrücken verknüpfen arithmetische Operatoren Ausdrücke, deren Ergebnis ein numerischer Wert ist.

Das Ergebnis einer arithmetischen Operation ist immer ein numerischer (= arithmetischer) Wert, und zwar eine Zahl vom Datentyp Integer im Wertebereich zwischen -2^{31} und $+2^{31}-1$.

Operation	**Operator**
Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Modulo	MOD

9.2.1.1 Addition

Der Additions-Operator ist das Pluszeichen (+).

Regeln:

– Für das Pluszeichen als Operator in einer Addition:
Der Ergebniswert muss größer oder gleich -2^{31} und kleiner oder gleich $+2^{31}-1$ sein.

– Für das Pluszeichen als Vorzeichen:
Dem Pluszeichen muss direkt das numerische Literal oder der Variablenname folgen.

Beispiel

```
/A = +45
/B = 36
/C = 45 + 5
/D = A+ B
```

Alle diese Zuweisungen und auch die einfachen Additions-Ausdrücke sind gültig.

9.2.1.2 Subtraktion

Der Subtraktions-Operator ist das Minuszeichen (-).

Regeln:

Für das Minuszeichen als Operator in einer Subtraktion:
– Dem Minuszeichen muss ein Leerzeichen vorangehen oder folgen, damit es nicht mit dem Bindestrich verwechselt wird (siehe Beispiel).
– Der Ergebniswert muss größer oder gleich -2^{31} und kleiner oder gleich $2^{31}-1$ sein.

Für das Minuszeichen als Vorzeichen:
– Dem Minuszeichen muss ein Leerzeichen vorangehen.
– Dem Minuszeichen muss direkt das numerische Literal bzw. der Variablenname folgen.

Beispiel

```
/A = -45
/B = -45 - 5
/C = A - B
/D = A-B
```

Alle diese Zuweisungen sind syntaktisch korrekt: den Variablen A, B und C werden Integer-Werte zugewiesen (-45, -50, 5) und der Variablen D der Inhalt einer Variablen A-B, wenn diese deklariert und initialisiert ist. Gibt es keine Variable mit dem Namen A-B oder ist sie nicht initialisiert, führt diese Zuweisung zum Fehler.

9.2.1.3 Multiplikation

Der Multiplikations-Operator ist der Stern (*).

Regeln:
Der Ergebniswert muss im Wertebereich -2^{31} bis $+2^{31}-1$ liegen.

9.2.1.4 Division

Divisions-Operator ist der Schrägstrich (/). Bei der Division ist zu beachten, ob der Quotient eine ganze Zahl ist, das heißt, ob der Dividend ein Vielfaches des Divisors ist. Ist das nicht der Fall, kann der Divisionsrest über die Modulo-Operation bestimmt werden (siehe Abschnitt „Modulo-Operation“ auf Seite 265).

Regeln:

- Das Ergebnis ist eine ganze Zahl (ohne Rest) im Wertebereich -2^{31} bis $+2^{31}-1$.
- Das Ergebnis der Division wird folgendermaßen berechnet:
  - Wenn der Quotient eine ganze Zahl ist, die Division also keinen Rest ergibt, wird der Quotient als Ergebnis eingesetzt.
  - Wenn der Quotient keine ganze Zahl ist, die Division also einen Rest ergibt, wird als Ergebnis die dem Betrag nach nächstkleinere Zahl eingesetzt, versehen mit dem aus der Division resultierenden Vorzeichen.
- Division durch Null führt zum Fehler.

Beispiel

Zuweisung	Ergebnis
/A = 7	
/B = -4	
/C = A / B	-1
/D = A / 2	3

Die Division 7: -4 liefert den Quotienten -1,75. Da als Divisionsergebnis nur ganze Zahlen eingesetzt werden können, wird der Variablen C nicht die nächstkleinere ganze Zahl (-2) zugewiesen, sondern die dem Betrag nach nächstkleinere ganze Zahl, versehen mit dem Vorzeichen, das sich aus der Division ergibt: C = -1.
Die Division 7 : 2 liefert den Quotienten 3,5. Der Variablen D wird die dem Betrag nach nächstkleinere Zahl zugewiesen, versehen mit dem Vorzeichen, das sich aus der Division ergibt: D = 3.

9.2.1.5 Modulo-Operation

Die Modulo-Operation liefert den ganzzahligen Rest einer Division; Operator ist der reservierte Name MOD.

Regeln:

– Dem Operatornamen MOD muss ein Leerzeichen vorangehen und ein Leerzeichen folgen; andernfalls wird diese Zeichenfolge als Teil eines Variablennamens interpretiert.

– Das Ergebnis wird nach folgender Formel berechnet:

```
A MOD B = A – (A / B) * B
```

Beispiel

Zuweisung	Ergebnis
/Y = 9 MOD 4	1
/Y = -9 MOD 4	-1

Der Variablen Y wird der Wert 1 zugewiesen, da der Ausdruck folgendermaßen berechnet wird:

```
9 MOD 4 = 9 – (9 / 4) * 4
```

Zunächst wird die Klammer aufgelöst: die Division 9 : 4 liefert als Ergebnis die Zahl 2,25; in die Gleichung eingesetzt wird dementsprechend die Zahl 2:

```
9 MOD 4 = 9 – 2 * 4
```

Entsprechend den Rechenregeln ergibt sich 9 - 8, das heißt:

```
9 MOD 4 = 1
```

Für die Zuweisung an die Variable Z ergibt sich entsprechend der Wert -1:

```
–9 MOD 4 = –9 – (–9 / 4) * 4
         = –9 – (–2) * 4
         = –9 – (–8)
         = –1
```

9.2.2 Vergleichsoperatoren

Über die Vergleichsoperatoren werden in einfachen Ausdrücken zwei Basisterme gleichen Typs miteinander verglichen. In zusammengesetzten Ausdrücken werden durch die Vergleichsoperatoren Ausdrücke miteinander verglichen, deren Ergebnisse vom gleichen Typ sein müssen.

Vergleich	**Operatoren**		
kleiner als	LT	<	
kleiner oder gleich	LE	<=	
gleich	EQ	=	==
nicht gleich	NE	<>	
größer oder gleich	GE	>=	
größer als	GT	>	

Das Ergebnis einer Vergleichsoperation ist immer ein boolescher Wert, das heißt ein Wert aus dem Wertebereich FALSE oder TRUE.

Für alle Vergleichsoperatoren gelten die gleichen Regeln; diese Regeln werden daher nur einmal beschrieben.

Regeln:

– Die Operanden eines Vergleichsoperators müssen vom gleichen Typ sein; andernfalls wird eine Fehlermeldung ausgegeben und die Fehlerbehandlung angestoßen.
– Das Ergebnis einer Vergleichsoperation ist entweder TRUE (= wahr) oder FALSE (= falsch).
– Ist der Vergleichsoperator das Gleichheitszeichen (=), muss der Vergleichsausdruck in Klammern eingeschlossen werden, damit Prüfung auf Gleichheit und Zuweisung eines Wertes an einen Operanden eindeutig zu unterscheiden sind (operand$_1$ = operand$_2$). Wird das Gleichheitszeichen doppelt geschrieben (==), kann die Klammerung entfallen.

Beispiel

```
/B = A + ZAEHL
/IF (B = A + ZAEHL)
```

Die erste Zeile enthält eine Zuweisung: Der Variablen B wird als Ergebnis die Summe der Inhalte der Variablen A und ZAEHL zugewiesen.

Die zweite Zeile enthält einen Vergleichsausdruck: Wenn der Inhalt der Variablen B, der an anderer Stelle in der Prozedur gesetzt wurde, dem Ergebnis der Addition der Inhalte von A und ZAEHL entspricht, wird der „Then“-Zweig des IF-Blocks durchlaufen; der Inhalt der

Variablen B wird nicht verändert, ihr wird kein neuer Wert zugewiesen. Um die Unterscheidung Vergleich / Zuweisung deutlicher zu machen, kann der Vergleichsoperator wahlweise auch als == geschrieben werden:

```
/IF (B == A + ZAEHL)
```

Numerischer Vergleich

„Numerischer Vergleich“ bedeutet, dass beide Operanden des Vergleichsoperators integer Ausdrücke sind. Verglichen werden die Werte der Operanden.

Vergleich boolescher Werte

Beide Operanden des Vergleichsoperators müssen boolesche Ausdrücke sein.

Regeln:

Es sind nur folgende Operatoren zugelassen:

Operation	**Operatoren**		
gleich	EQ	=	==
nicht gleich	NE	<>	

String-Vergleich

„String-Vergleich“ bedeutet, dass beide Operanden des Vergleichsoperators String-Ausdrücke sind.

Regeln:

- Strings werden zeichenweise, das heißt byteweise, verglichen, und zwar von links nach rechts bis zum ersten unterschiedlichen Zeichen.
- Das erste unterschiedliche Zeichen bestimmt, welcher String größer bzw. kleiner ist; die weiteren Zeichen werden für den Vergleich nicht mehr berücksichtigt.
- Die Begriffe „größer“ und „kleiner“ beziehen sich auf die Reihenfolge der Zeichen im EBCDI-Code, von X'00' bis X'FF'.
- Sind die beiden Strings unterschiedlich lang, haben aber bis zum letzten Zeichen des kürzeren Strings die gleiche Zeichenfolge, so gilt der kürzere String als kleiner.
- Strings sind gleich, wenn sie gleich lang sind und sich in keinem Zeichen unterscheiden.

Zeichen- oder byteweiser Vergleich bedeutet: die EBCDIC-Äquivalente der Zeichen werden untersucht.

Beispiel

Zuweisung	Ergebnis
/A = 'ABC'	
/B = 'ABCDE'	
/C = X'C1C2C3'	
/D = 'B'	
/E = (B > A)	TRUE
/F = (D > A)	TRUE
/G = (C = A)	TRUE
/H = (B = A)	FALSE

Der Variablen E wird der boolesche Wert TRUE zugewiesen, da die Strings in den Variablen B und A in den ersten drei Zeichen identisch sind, der String in der Variablen A ('ABC') jedoch kürzer und daher kleiner ist als der String in der Variablen B ('ABCDE').

Auch der Variablen F wird der boolesche Wert TRUE zugewiesen: Der String 'B' (in der Variablen D) ist zwar kürzer als der String 'ABC' (in der Variablen A), das erste Zeichen des Strings 'B' hat jedoch im EBCDI-Code einen höheren Wert als das erste Zeichen des Strings 'ABC'.

Der Vergleich der Variableninhalte von C und A liefert Gleichheit, da der X-String, mit dem die Variable C initialisiert wurde, die Halbbyte-Schreibweise des Strings 'ABC' ist: Der Variablen G wird daher der boolesche Wert TRUE zugewiesen.

Der Variablen H wird der boolesche Wert FALSE zugewiesen, da die Strings in den Variablen B und A nicht gleich sind.

9.2.3 Logische Operatoren

Logische Operatoren verknüpfen zwei boolesche Ausdrücke miteinander (Ausnahme: NOT. Dieser Operator bezieht sich nur auf einen booleschen Ausdruck).

Das Ergebnis einer Verknüpfung mit logischen Operatoren ist immer ein boolescher Wert (TRUE oder FALSE), der über einen der Namen angesprochen werden kann, die für boolesche Konstanten reserviert sind.

Operation	**Operator**
Negation	NOT
Oder	OR
Und	AND
exklusives Oder (= entweder oder)	XOR

Regeln:

– Es gelten die Regeln für logische Verknüpfungen.
– NOT invertiert den Wert eines Ausdrucks.

Beispiel

```
/A = TRUE
/B = 4
/C = 20
/D = A OR (B > C)
/E = A AND (B > C)
```

Die Variable D erhält den Wert TRUE, da in der Oder-Verknüpfung einer der Operanden den Wert TRUE hat. Die Variable E erhält den Wert FALSE, da nur einer der beiden Operanden der Und-Verknüpfung den Wert TRUE hat.

9.2.4 Verkettungsoperator

Der Verkettungsoperator // verkettet zwei String-Ausdrücke miteinander.

Das Ergebnis einer Verkettung ist immer ein String.

Regel:
Die als Operanden angegebenen Strings werden ohne Lücke aneinander gekettet.

Beispiel

```
/A = 'Datum: '
/B = DATE(FORMAT = *GERMAN)
/C = A // B
```

Der Variablen A wird direkt der C-String 'Datum: ' zugewiesen, der Variablen B das Ergebnis der Funktion DATE, das ist das Tagesdatum als String. Diese Strings werden zu einem neuen String verkettet, der der Variablen C zugewiesen wird. C hat dann zum Beispiel folgenden Inhalt: Datum: 26.06.1996

9.3 Typen von Ausdrücken

Der Typ eines Ausdrucks ist immer identisch mit dem Datentyp des Ergebniswertes. Entsprechend den drei Datentypen werden daher drei Typen von Ausdrücken unterschieden:

– Arithmetische Ausdrücke
– Logische bzw. boolesche Ausdrücke
– String-Ausdrücke

Der Typ eines einfachen Ausdrucks wird durch den Operator bestimmt, der die Basisterme miteinander verknüpft. In der folgenden Tabelle sind einfache Ausdrücke und ihre Komponenten aufgelistet:

Typ	**Operatoren**	**Basisterme**	**Ergebnis-Datentyp**
arithmetischer Ausdruck	arithmetisch	Zahlen Variablennamen Funktionsaufrufe	Integer
Vergleichsausdruck	Vergleich	Zahlen boolesche Konstanten String-Literale Variablennamen Funktionsaufrufe	Boolean
logischer bzw. boolescher Ausdruck	logisch	boolesche Konstanten Variablennamen Funktionsaufrufe	Boolean
String-Ausdruck	Verkettung	String-Literale Variablennamen Funktionsaufrufe	String

Der Datentyp des Ergebnisses und damit des Ausdrucks wird bei zusammengesetzten Ausdrücken bestimmt durch den Operator, der als Letzter ausgewertet wird. Die Rangfolge der Operatoren und die Art wie Ausdrücke ausgewertet werden, ist im folgenden Abschnitt beschrieben.

9.4 Auswertung von Ausdrücken

Einfache Ausdrücke enthalten nur einen Operator; sie werden so ausgewertet, wie oben bei der Beschreibung der Operatoren gezeigt.

Zusammengesetzte Ausdrücke müssen zunächst in Teilausdrücke zerlegt werden bis hinunter zu einfachen Ausdrücken. Die Reihenfolge, in der die Teilausdrücke ausgewertet werden, wird durch die Priorität der Operatoren bestimmt. Durch Setzen von Klammern kann der Benutzer die Auswertung steuern.

9.4.1 Priorität von Operatoren

Die Priorität von Operatoren wird in zwei Stufen ausgewertet: zunächst entsprechend der Rangfolge der Operatortypen, dann innerhalb eines Operatortyps entsprechend der Rangfolge der Operatoren.

Rangfolge der „Operatortypen“:

1. Vorzeichen, Negation
2. arithmetische Operatoren
3. Verkettungsoperator
4. Vergleichsoperatoren
5. logische Operatoren

Rangfolge der arithmethischen Operatoren („Punkt- vor Strichrechnung“):

1. Multiplikation, Division, Modulo-Operation (*, /, MOD)
2. Addition, Subtraktion (+, -)

Rangfolge der logischen Operatoren:

1. Und-Verknüpfung (AND)
2. Oder-Verknüpfung (OR, XOR)

Vergleichsoperatoren:
Alle Vergleichsoperatoren sind gleichrangig.

Beispiel

Zusammengesetzte logische Ausdrücke werden häufig in IF-Blöcken verwendet, und zwar im Operanden CONDITION des IF-Kommandos. Wenn die Bedingung, die durch den Ausdruck festgelegt ist, wahr ist, wird das Kommando bearbeitet, das dem IF-Kommando folgt. Wenn die Bedingung falsch ist, wird das nächste ELSE-IF- oder ELSE-Kommando bearbeitet (Näheres zu IF-, ELSE-IF, und ELSE-Kommando: siehe Kapitel „S-Prozeduren erstellen“ oder Kapitel „SDF-P-Kommandos“).

Ein IF-Kommando könnte zum Beispiel folgende Bedingung enthalten:

```
A + B / C > D + C MOD E AND A + D * E < D * C OR F // G > H
```

Zum Zeitpunkt der IF-Abfrage haben die Variablen A bis H folgende Werte:

```
/A = 4
/B = 29
/C = 9
/D = 3
/E = 5
/F = 'ABC'
/G = 'DEF'
/H = 'ABCDE'
```

Der Ausdruck wird in folgenden Schritten ausgewertet:

1. Arithmetische Operatoren: Multiplikation / Division

Operation	entspricht	Ergebnis
B / C	29 / 9	3
C MOD E	9 MOD 5	4
D * E	3 * 5	15
D * C	3 * 9	27

Ergebnis von Schritt 1:

```
A + 3 > D + 4 AND A + 15 < 27 OR F // G > H
```

2. Arithmetische Operatoren: Addition

Operation	entspricht	Ergebnis
A + 3	4 + 3	7
D + 4	3 + 4	7
A + 15	4 + 15	19

Ergebnis von Schritt 2:

```
7 > 7 AND 19 < 27 OR F // G > H
```

3. Verkettungsoperator

Operation	entspricht	Ergebnis
F // G	'ABC' // 'DEF'	'ABCDEF'

Ergebnis von Schritt 3:

```
7 > 7 AND 19 < 27 OR 'ABCDEF' > H
```

4. Vergleichsoperatoren

Operation	Ergebnis
7 > 7	FALSE
19 < 27	TRUE
'ABCDEF' > 'ABCDE'	TRUE

Ergebnis von Schritt 4:

```
FALSE AND TRUE OR TRUE
```

5. Logische Operatoren: AND

Operation	Ergebnis
FALSE AND TRUE	FALSE

Ergebnis von Schritt 5:

```
FALSE OR TRUE
```

6. Logische Operatoren: OR

Operation	Ergebnis
FALSE OR TRUE	TRUE

Ergebnis von Schritt 6:

```
TRUE
```

Die Bedingung ist demnach erfüllt.

Reihenfolge der Auswertung

Die Reihenfolge der Auswertung ist bei allen Operanden undefiniert. Es kann vorkommen, dass der rechte Operand des AND zuerst ausgewertet wird.

Beispiel

```
/DECLARE-VARIABLE I(TYPE = INTEGER)
/IF IS-INITIALIZED ('I') AND (I < 10)
```

Da zu diesem Zeitpunkt I noch nicht initialisiert ist, kommt es bei der Auswertung von I < 10 zum Fehler.

9.4.2 Klammern

Durch das Setzen von Klammern ( ) kann ein Ausdruck zum einen so gegliedert werden, dass die Auswertung deutlich wird. Zum anderen kann durch das Setzen von Klammern auch die Auswertung gesteuert werden.

Enthält ein Ausdruck Klammern, werden zunächst die Teilausdrücke in den Klammern ausgewertet, entsprechend der Schachtelung der Klammern. Erst dann werden die nicht geklammerten Operatoren bearbeitet.

Die maximal mögliche Anzahl von Klammern hängt von der Komplexität des Ausdrucks ab; bis zu 50 Klammern werden aber in jedem Fall akzeptiert.

Beispiel

Im folgenden Beispiel wird die Auswertung des Ausdrucks durch das Setzen von Klammern gesteuert:

```
(A + B) / C > (D + C) MOD E AND ((A + D) * E < (D * C) OR (F // G) > H)
```

10 S-Prozeduren optimieren

Unter Gesichtspunkten der Performance wird die Verwendung großer, umfangreicher S-Prozeduren oft als weniger optimal beurteilt. Das folgende Kapitel soll den Programmierern von S-Prozeduren einige hilfreiche Tipps und Hinweise zum Schreiben von performanten S-Prozeduren geben. Da sich Problemstellungen in S-Prozeduren oft unterschiedlich umsetzen lassen, wird in den folgenden Tipps und Hinweisen jeweils die als performanter empfohlene Schreibweise einer weniger zu empfehlenden Schreibweise gegenüber gestellt und kurz miteinander verglichen. Zu folgenden Themenbereiche werden Optimierungsmöglichkeiten gezeigt:

- SDF-Syntaxanalyse
- Verwendung von Variablen
- Prozeduraufruf
- String in einer Liste suchen
- Verwendung von Kommentaren
- Programmaufrufe in der Prozedur

Am Ende des Kapitels werden anhand einer Beispielprozedur noch einmal alle Optimierungsmöglichkeiten gezeigt.

10.1 SDF-Syntaxanalyse

SDF bietet zur Vereinfachung der Kommandoeingabe eine Vielzahl von Möglichkeiten, wie z.B. Abkürzungsmöglichkeiten, implizite Variablendeklaration, Gebrauch von Stellungs- und Schlüsselwortoperanden. Der Gebrauch dieser Möglichkeiten, der in erster Linie die Eingabe im Dialog erleichtern soll, verlangt bei der Syntaxanalyse aber auch mehr Arbeitsschritte und kann sich deshalb im Prozedurmodus auch auf die Performance auswirken.

empfohlene Schreibweise	**weniger performante Schreibweise**

Beispiel 1:
`/FOR I=*COUNTER(FROM=1, TO=50)`
`/   DECLARE-VARIABLE VAR&I`
`/END-FOR`
	`/FOR I=*COUNT(1,50)`
	`/   DEC-VARI VAR&I`
	`/END-FOR`

Beispiel 2:
`/SHOW-VAR VARIABLE-NAME=*ALL,-`
`/   INFORMATION=*PARAMETERS(-`
`/   NAME=*FULL-NAME(LIST-INDEX-NUMBER=*YES))`
	`/SHOW-VAR *ALL,LIST-INDEX=Y`

Beispiel 3:
`/DECL-VAR (TST1, TST2, TST3),TYPE=*STRING`
	`/DECL-VAR TST1,TYPE=*STRING`
	`/DECL-VAR TST2,TYPE=*STRING`
	`/DECL-VAR TST3,TYPE=*STRING`

Beispiel 4: `/I = 1`	`/SET-VARIABLE I = 1`

Die Syntaxanalyse wird in folgenden Fällen wesentlich erleichtert:

- Namen von Kommandos/Anweisungen, Operanden, Schlüsselwortwerten werden vollständig angegeben (alternativ auch Aliasnamen). Der Gebrauch von Minimalabkürzungen, der manchmal zur Vermeidung von Kompatibilitätsproblemen empfohlen wird, ist aus Performance-Sicht nicht ausreichend (siehe Beispiel 1).
- Untergeordnete Operanden werden nicht außerhalb ihrer Struktur angegeben (siehe Beispiel 2).
- Können in einem Kommando bzw. einer Anweisung mehrere Objekte gleichzeitig (z.B. durch die Angabe einer Liste oder durch Musterzeichen) angegeben werden, muss die Syntaxanalyse nur einmal durchgeführt werden. Im Gegensatz dazu muss die Syntaxanalyse jedes Mal neu durchgeführt werden, wenn das Kommando bzw. die Anweisung für jedes Objekt erneut eingegeben wird (siehe Deklaration der String-Variablen TST1, TST2 und TST3 im Beispiel 3).
- Die SDF-Syntaxanalye kann auch im Fall des SET-VARIABLE-Kommandos vermieden werden, wenn das Kommando in der verkürzten Form (d.h. ohne den Kommandonamen) genutzt wird (siehe Beispiel 4).

Die Berücksichtigung dieser Punkte mag bei der Entwicklung bzw. Erstellung von S-Prozeduren vielleicht umständlich erscheinen, es ist aber zu bedenken, dass eine Prozedur nur einmal geschrieben, aber oft aufgerufen wird.

10.2 Verwendung von Variablen

10.2.1 Gruppierung von Kommandos, die eine Variable nutzen

SDF-P verwendet für die zuletzt genutzten Variablen einen Puffer, der eine begrenzte Anzahl von Elementen aufnehmen kann. Falls möglich, sollten Kommandos, die dieselbe Variable verwenden, innerhalb der Prozedur nahe zu einander gruppiert werden. Dadurch kann die Zugriffszeit für die Variable vermindert werden.

10.2.2 Listenvariable mit Werten versorgen

Die Initialisierung einer Listenvariable, die aus einfachen Variablen besteht (Typ ANY, STRING, INTEGER oder BOOLEAN), sollte in einem Kommando erfolgen. Die Initialisierung der Listenvariablen durch mehrere Kommandos sollte vermieden werden.

empfohlene Schreibweise	**weniger performante Schreibweise**

`/DECLARE-VARIABLE V(TYPE=*STRING),-`
`/  MULTIPLE-ELEMENT=*LIST`
`/V=*STRING-TO-VAR('(AA,BB,CC)')`
	`/DECLARE-VARIABLE V(TYPE=*STRING),-`
	`/  MULTIPLE-ELEMENT=*LIST`
	`/SET-VARIABLE V = 'AA',*EXTEND`
	`/SET-VARIABLE V = 'BB',*EXTEND`
	`/SET-VARIABLE V = 'CC',*EXTEND`

10.2.3 Erzeugung nicht benötigter Variablen

Die Ausgabe von SHOW-Kommandos wird oft mit dem Kommando EXECUTE-CMD (bzw. auch durch Zuweisung eines S-Variablen-Stroms) in eine zusammengesetzte Variable gelenkt, um die dabei erzeugten Variablen anschließend in einer FOR-Schleife auszuwerten. Dabei sollte beachtet werden, dass der Aufrufer bei vielen SHOW-Kommandos die ausgegebene Informationsmenge, und damit auch die Menge der erzeugten Variablen, über geeignete Operanden (z.B. INFORMATION=... oder SELECT=...) beeinflussen kann. Die gezielte Erzeugung von Variablen bei gleichzeitiger Vermeidung nicht benötigter Variablen ist einer sonst notwendigen IF-THEN-ELSE-Konstruktion in der FOR-Schleife vorzuziehen.

10.3 Verwendung von vordefinierten Funktionen

SDF-P erkennt eine Funktion an den Klammern, die dem Funktionsnamen folgen. Die Klammern sind optional, wenn der Funktionsaufruf keine Eingabeparameter enthält. Bei Weglassen der Klammern interpretiert SDF-P den Funktionsnamen zunächst als Variable (siehe Abschnitt „Funktionsaufruf“ auf Seite 229). Die Schreibweise mit Klammern vermeidet nicht nur Verwechslungsmöglichkeiten, sondern sie ist auch performanter, da SDF-P nicht erst nach einer Variablen suchen muss.

empfohlene Schreibweise	weniger performante Schreibweise
`/WRITE-TEXT 'Es ist jetzt: &(TIME())'`	`/WRITE-TEXT 'Es ist jetzt: &(TIME)'`

10.4 Prozeduraufruf

10.4.1 Aufruf mit CALL- oder INCLUDE-PROCEDURE

Da mit dem Kommando CALL-PROCEDURE sowohl S-Prozeduren (Analyse und Ausführung durch SDF-P) als auch Nicht-S-Prozeduren (Analyse und Ausführung durch SYSFILE) aufgerufen werden können, ist aus dem Aufruf allein noch nicht ersichtlich, welche Prozedurart vorliegt. Aus historischen Gründen wird beim CALL-PROCEDURE-Aufruf zunächst von einer Nicht-S-Prozedur ausgegangen, d.h. die Prozedur wird zunächst von der Systemkomponente SYSFILE analysiert. Erst danach wird eine S-Prozedur von SDF-P analysiert.
Im Gegensatz dazu ist das Kommando INCLUDE-PROCEDURE ausschließlich dem Aufruf von S-Prozeduren vorbehalten, d.h. die Analyse wird immer von SDF-P durchgeführt.

Falls das unterschiedliche Variablenmodell, das den beiden Kommandoaufrufen zu Grunde liegt, für den Aufrufer ohne Bedeutung ist, sollte eine S-Prozedur mit dem Kommando INCLUDE-PROCEDURE aufgerufen werden. Dadurch wird die unnötige Analyse durch die Komponente SYSFILE vermieden.
Auch für prozedurimplementierte Kommandos kann mit SDF-A ≥ V4.1A der Aufruf mit INCLUDE-PROCEDURE definiert werden (siehe Handbuch „SDF-A“ [16]).

10.4.2 Prozedur als Bibliothekselement

Seit der Einführung des COMPILE-PROCEDURE-Kommandos können S-Prozeduren, die als PLAM-Bibliothekselement abgelegt werden, vom Elementtyp J oder SYSJ sein. Der Elementtyp J wird empfohlen für Textprozeduren, der Elementtyp SYSJ für kompilierte Prozeduren. Bei Aufruf mit den Kommandos CALL- bzw. INCLUDE-PROCEDURE kann der Elementtyp angegeben werden. Voreingestellt ist TYPE=*STD, d.h. erst wenn das Element vom Typ SYSJ nicht existiert, wird das Element vom Typ J aufgerufen. Bei dieser Vorgehensweise wird ein zusätzlicher Bibliothekszugriff benötigt. Dieser zusätzliche Zugriff kann vermieden werden, wenn der Aufrufer den Elementtyp kennt und ihn auch explizit angibt.

10.4.3 Übergabe von Parametern bzw. Informationen

empfohlene Schreibweise	**weniger performante Schreibweise**

Prozedur 1:
 	
`/GET-IDOM-LIBRARY-NAME LOGICAL-ID='SYSSPR',-`
`/ INT-LIB=IDOM-GLB-SYSSPR`
	Prozedur 1:
	
	`/ASSIGN-STREAM SYSINF,-`
	`/ TO=*VAR(VAR-NAME=INT-LIB-NAME)`
	`/GET-IDOM-LIBRARY-NAME LOGICAL-ID='SYSSPR'`
	`/ASSIGN-STREAM SYSINF,TO=*SAME`
	`/IDOM-GLB-SYSSPR=INT-LIB-NAME#1.NAME`

Prozedur 2 für die Implementierung des Kommandos GET-IDOM-LIBRARY-NAME:
 	
`/ . . .`
`/DECLARE-PARAMETER INT-LIB(TYPE=*STRING,-`
`/ TRANSFER-TYPE=*BY-REFERENCE)`
`. . .`
`/INT-LIB = 'xxx'`
	Prozedur 2 für die Implementierung des Kommandos GET-IDOM-LIBRARY-NAME:
	
	`/INT-LIB.NAME='xxx'`
	`/TRANSMIT-BY-STREAM STREAM-NAME=SYSINF,-`
	`/ VAR-NAME=INT-LIB,-`
	`/ RETURN-VAR=*NONE`

Sollen durch eine Prozedur, die mit CALL-PROCEDURE (direkt oder wie im Beispiel als implementierte Prozedur) aufgerufen wird, Ausgabeinformationen in Variablen zurückgeliefert werden, sollte folgende Parameterdeklaration gewählt werden:

```
/DECLARE-PARAMETER <var-name>,(...,TRANSFER-TYPE-*BY-REFERENCE)
```

Diese Deklaration ist für einfache Variablen (Typ ANY, STRING, INTEGER oder BOOLEAN) die bessere Austauschmöglichkeit (im Gegensatz zur Verwendung von Variablenströmen, wie auf der rechten Seite im Beispiel).

10.5 String in einer Liste suchen

empfohlene Schreibweise	weniger performante Schreibweise
/MATCH = SEARCH-LIST-INDEX(- / LIST-VARIABLE-NAME = X,- / PATTERN = P)	/LOOP: FOR I = *COUNTER(1,SIZE('X'),1) / IF (INDEX(X#I,P) <> 0) / SET-VARIABLE MATCH = I / EXIT-BLOCK LOOP / END-IF /END-FOR

Die Funktion SEARCH-LIST-INDEX wurde speziell zur Verbesserung der Performance entwickelt. Sie durchsucht in einem Aufruf eine Listenvariable nach einer Zeichenfolge oder einem regulären Ausdruck und liefert den Index des ersten Treffers zurück. Die Anwendung dieser Funktion erspart die zeitaufwändige Suche in einer FOR-Schleife.

10.6 Kommentare

empfohlene Schreibweise	weniger performante Schreibweise
/ &* End-of-line comments	/REMARK Bad comments possibility / “Not so good comments”

Zeilenendekommentare (eingeleitet durch die Zeichenfolge &*) sollten bevorzugt verwendet werden. Kommentare, die in Anführungszeichen eingeschlossen sind, sollten vermieden werden. REMARK-Kommandos, die nur der Kommentierung dienen, sollten generell nicht verwendet werden.

10.7 Programmaufruf in Prozeduren

10.7.1 Verwendung der Dienstprogramme EDT und LMS in einer Prozedur

Werden die Dienstprogramme EDT und LMS in derselben Prozedur aufgerufen, sollte beachtet werden, dass beide Dienstprogramme einige Funktionen anbieten, die das andere Dienstprogramm ebenfalls zur Verfügung stellt: Im EDT kann z.B. ein Bibliothekelement geöffnet (@OPEN-Anweisung) und editiert werden, im LMS kann ein Bibliothekelement ebenfalls editiert werden (EDIT-ELEMENT-Anweisung).

Bei jedem Programmwechsel während der Bearbeitung eines Bibliothekselements wird zusätzliche Zeit zum Entladen des einen und zum Laden des anderen Programmes benötigt. Die direkte Bearbeitung in nur einem Dienstprogramm (EDT oder LMS) ist der Bearbeitung über mehrere Programmaufrufe (siehe nachfolgendes Beispiel) vorzuziehen:

1. LMS aufrufen - Element in Datei extrahieren - LMS beenden
2. EDT aufrufen - Datei einlesen und editieren - EDT beenden
3. LMS aufrufen - Datei wieder in Element ablegen - LMS beenden

empfohlene Schreibweise	**weniger performante Schreibweise**

`/START-LMS`
`//OPEN-LIBRARY LIB=&(LIB-1)`
`//EDIT-ELEMENT ELEM=&(ELEM-1),-`
`//  TYPE=&(TYP)`
`...`
`//END`
oder:
`/START-EDT`
`@OPEN L=&(LIB-1)(E=&(ELEM-1),&(TYP))`
`...`
`@CLOSE`
`@HALT`
	`/START-LMS`
	`//OPEN-LIBRARY LIB=&(LIB-1)`
	`//EXTRACT-ELEMENT ELEM=(&(ELEM-1),-`
	`//  TYPE=&(TYP)),TO=#WORK-1`
	`//END`
	`/START-EDT`
	`@READ '#WORK-1'`
	`...`
	`@WRITE OVERWRITE`
	`@HALT`
	`/START-LMS`
	`//OPEN-LIBRARY LIB=&(LIB-1)`
	`//ADD-ELEMENT #WORK-1,-`
	`//  TO-ELEM=(&(ELEM-1),TYPE=&(TYP))`
	`//END`

10.8 Beispiel einer optimierten Prozedur

Das folgende Beispiel zeigt eine Prozedur, in der alle Optimierungsmaßnahmen aus den vorangegangenen Abschnitten angewendet wurden. Die optimierte Prozedur (erstellt vom Team „BS2000 Performance Controlling and Modelling“) benötigt nur noch 62% der CPU-Zeit gegenüber der Ausgangsprozedur, die in weniger performanter Schreibweise erstellt war.

Optimierte Prozedur (empfohlene Schreibweise):

```
/SET-PROCEDURE-OPTIONS
/
/OPEN-VARIABLE-CONTAINER CONTAINER-NAME=CONTFS, -
/                         FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=BAD.LIB)
/DECLARE-VARIABLE VARIABLE-NAME=C-FS-L(TYPE=*STRUCTURE) -
/               , MULTIPLE-ELEMENTS=*LIST, CONTAINER=CONTFS
/
/&* The list variables and the output of show-file-attr
/ DECLARE-VARIABLE VARIABLE-NAME = ( -
/          LIST   ( TYPE = *STRING    ), -
/          TSIL   ( TYPE = *STRING    ), -
/          FS-OUT ( TYPE = *STRUCTURE ), -
/          ), MULTIPLE-ELEMENTS=*LIST
/
/&* List initialization
/ LIST = *STRING-TO-VAR ( '(ABC,DEF,GHI,JKL,MNO,ABC,DEF,GHI,JKL,MNO)' )
/
/&* List in reverse order
/ FOR I = *LIST(LIST)
/   TSIL = I, *PREFIX
/ END-FOR
/
/&* Show list variable in one command
/ SHOW-VARIABLE VARIABLE-NAME = ( LIST, TSIL )
/
/&* Search variable A inside LIST and save the result in B
/ A = 'I'
/ IND = SEARCH-LIST-INDEX('LIST', '^&A.$', PATTERN-TYPE=*REGULAR-EXPRESSION)
/ IF ( IND <> 0 )
/  B = LIST#IND
/ END-IF
/
/&* Get all files beginning with 'A'
/ EXECUTE-CMD CMD = (/SHOW-FILE-ATTRIBUTES A*) -
/           , STRUCTURE-OUTPUT = FS-OUT -
/           , TEXT-OUTPUT = *NONE
```

```
/
/&* Put contents in a container
/ C-FS-L = FS-OUT
/
/&* Calculate total size
/ TSIZE = 0
/ FOR I = *COUNTER( 1, SIZE('FS-OUT'), 1 )
/   TSIZE = TSIZE + FS-OUT#I.F-SIZE
/ END-FOR
/
/&* Save container ( close implicit at procedure end )
/ SAVE-VARIABLE-CONTAINER CONTAINER-NAME = CONTFS
/
/&* Display date, time and total file size
/ WRITE-TEXT -
/   'Today &(DATE()) at &(TIME()) we have a total file size of &TSIZE -
/    on user-id &(USER-ID())..'
/
/EXIT-PROCEDURE
```

Ausgangsprozedur (weniger empfohlene Schreibweise):

```
/SET-PROC-OPT
/
/OP-VAR-CONT CONTFS, *L(BAD.LIB)
/DECL-VAR C-FS-L,TYP=STRUCT,MULT=*L,CONT=CONTFS
/
/REMARK  This list variable
/ DECL-VAR LIST ( TYP = STRI ),MULT=*L
/
/REMARK  Its initialization
/ SET-VAR LIST ='ABC', *EXTEND; SET-VAR LIST ='DEF', *EXTEND
/ SET-VAR LIST ='GHI', *EXTEND; SET-VAR LIST ='JKL', *EXTEND
/ SET-VAR LIST ='MNO', *EXTEND; SET-VAR LIST ='ABC', *EXTEND
/ SET-VAR LIST ='DEF', *EXTEND; SET-VAR LIST ='GHI', *EXTEND
/ SET-VAR LIST ='JKL', *EXTEND; SET-VAR LIST ='MNO', *EXTEND
/
/REMARK  List in reverse order
/ DECL-VAR TSIL (TYP=STRI), MULT=*L
/
/ FOR I = *C( SIZE('LIST'), 1, -1)
/   SET-VAR TSIL#&(SIZE('LIST') - I + 1) = LIST#&I
/ END-F
/
/REMARK  Show list variable
/ SH-VAR LIST
/ SH-VAR TSIL
```

```
/
/REMARK  Search variable A inside LIST
/ SET-VAR A = 'I'
/ SET-VAR FOUND = FALSE
/ FOR I=*C(1,SIZE('LIST'),1), COND=(NOT FOUND)
/  IF ( LIST#&I == A )
/    SET-VAR FOUND = TRUE
/  EN-IF
/ END-F
/
/REMARK  Save it in B
/ IF ( FOUND )
/   SET-VAR B = LIST#&I
/ EN-IF
/
/REMARK  Get all file beginning with 'A'
/ DECL-V FS-OUT ,TYP=STRU,MULT=*L
/ EXEC-CMD (/SH-FIL-ATTR A*),STR-OUTPUT=FS-OUT,TEXT-OUT=*NONE
/
/REMARK  Calculate total size and put contents in a container
/ SET-VAR TSIZE = 0
/ FOR I=*C(1,SIZE('FS-OUT'),1)
/  SET-VAR TSIZE = TSIZE + FS-OUT#&I..F-SIZE
/  SET-VAR C-FS-L#&I = FS-OUT#&I
/ END-F
/
/REMARK  Save the container
/ SAVE-VAR-CONT CONTFS
/
/REMARK  Close the container
/ CLOSE-VAR-C CONTFS
/
/REMARK Display date, time and total file size
/ TDATE = DATE
/ TTIME = TIME
/
/W-T 'Today (&TDATE) at &TTIME we have a total file size of &TSIZE -
/     on user-id &USER-ID..'
/
/EXIT-PROC
```

11 S-Prozeduren testen

Dieses Kapitel beschreibt, welche Mittel dem Programmierer von S-Prozeduren bei der Fehlersuche zur Verfügung stehen.

Zum einen bietet SDF-P zwei Kommandos, die den Programmierer während der Testphase bei der Fehlersuche unterstützen, zum anderen kann der Programmierer Prozedur-Unterbrechungen nutzen, um den aktuellen Status der Prozedur abzufragen. Die Kommandos, die SDF-P als Testhilfe zur Verfügung stellt, sind: TRACE-PROCEDURE und MODIFY-PROCEDURE-TEST-OPTIONS.

Das Kommando TRACE-PROCEDURE wird eingesetzt, um den Prozedurlauf schrittweise verfolgen zu können. Das Kommando MODIFY-PROCEDURE-TEST-OPTIONS wird eingesetzt, wenn die Einstellung für das Protokollieren der Prozedur verändert werden soll, zum Beispiel um das Protokollieren auf bestimmte zu testende Teile der Prozedur zu beschränken.

In den drei folgenden Abschnitten werden zunächst die Anwendungsmöglichkeiten der beiden SDF-P-Kommandos nacheinander beschrieben. Im vierten Abschnitt schließlich werden die Möglichkeiten aufgezeigt, die dem Programmierer bei der Prozedurunterbrechung zur Verfügung stehen.

11.1 Prozedurlauf schrittweise verfolgen

Mit dem Kommando TRACE-PROCEDURE kann ein Prozedurlauf schrittweise verfolgt werden. Allerdings kann dieses Kommando nur für Prozeduren genutzt werden, die im Prozedurkopf, genauer im Kommando SET-PROCEDURE-OPTIONS, als unterbrechbar definiert sind (Operand INTERRUPT-ALLOWED = *YES).

Will der Anwender eine Prozedur schrittweise verfolgen, muss er vor dem Aufruf der Prozedur, also vor CALL- bzw. INCLUDE-PROCEDURE, das Kommando TRACE-PROCEDURE geben. Die Prozedur wird dann nach Anzahl der in TRACE-PROCEDURE angegebenen STEPS unterbrochen. Mit erneuter Angabe des Kommandos TRACE-PROCEDURE wird der Prozedurlauf fortgesetzt.

Im Operanden STEPS sollte angegeben werden, wie viele Kommandos ausgeführt werden sollen, bevor die Prozedur wieder unterbrochen wird. Beim ersten Aufruf von TRACE-PROCEDURE ist STEPS = 1 voreingestellt, d.h. die Prozedur wird nach jedem Kommando unterbrochen. Wird für STEPS ein anderer Wert angegeben, bleibt dieser veränderte Wert für den gesamten Prozedurlauf erhalten.

Wenn die Prozedur unterbrochen ist, kann der Programmierer zum Beispiel den Prozedurstatus abfragen oder mit dem Kommando MODIFY-PROCEDURE-TEST-OPTIONS die Einstellung für das Protokollieren oder die maximale Anzahl von Rückwärtssprüngen ändern (siehe unten). Danach wird die Prozedur wieder unterbrochen.

Anschließend kann der Prozedurlauf mit dem Kommando TRACE-PROCEDURE wieder schrittweise fortgesetzt werden. Die Prozedur kann auch mit dem Kommando RESUME-PROCEDURE fortgesetzt werden. Sie wird dann bis zum Ende abgearbeitet.

Wenn in der Prozedur ein Fehler aufgetreten ist, kann der Programmierer in der Unterbrechung zum Beispiel Variableninhalte ändern, Variablen deklarieren usw.

Protokollierung

Die Ausführung der Kommandos, die in der Prozedur nach Eingabe des TRACE-PROCEDURE-Kommandos durchlaufen werden, wird immer auf SYSOUT protokolliert, wenn Protokollieren für die Prozedur zulässig ist, unabhängig von der Einstellung des LOGGING-Operanden in den Kommandos CALL- bzw. INCLUDE-PROCEDURE oder MODIFY-PROCEDURE-TEST-OPTIONS. Diese Einstellungen werden ignoriert.

11.2 Protokollieren ändern

Wenn während der Testphase Einstellungen für das Protokollieren geändert werden sollen, kann im Dialog das Kommando MODIFY-PROCEDURE-TEST-OPTIONS aufgerufen werden.

Immer gilt: Protokollieren kann nur dann eingestellt werden, wenn für die Prozedur Protokollieren erlaubt ist (in den Kommandos SET-PROCEDURE-OPTIONS oder MODIFY-PROCEDURE-OPTIONS).

Wie in den übrigen Kommandos, in denen das Protokollieren eingestellt werden kann, wird es auch im Kommando MODIFY-PROCEDURE-TEST-OPTIONS durch den Operanden LOGGING gesteuert. Dabei kann Protokollieren für Kommandos und Daten getrennt ein- bzw. ausgeschaltet werden.

11.3 Endlosschleifen verhindern

Das Kommando MODIFY-PROCEDURE-TEST-OPTIONS bietet mit dem Operanden BACK-BRANCH-LIMIT die Möglichkeit, Endlosschleifen zu verhindern.

BACK-BRANCH-LIMIT setzt eine obere Grenze für die Zahl der Rückwärtssprünge innerhalb einer Prozedur. Als Rückwärtssprünge gelten zum einen die Sprünge vom Ende einer Schleife zum Anfang, das heißt von END-WHILE zu WHILE, von END-FOR zu FOR, von UNTIL zu REPEAT. Zum anderen zählen dazu auch Sprünge, die mit dem Kommando GOTO durchgeführt werden. Rückwärtssprünge mit dem Kommando SKIP-COMMANDS werden nicht gezählt.

11.4 Prozedur unterbrechen

Wenn Prozedurunterbrechung erlaubt ist (INTERRUPT-ALLOWED = *YES im Kommando SET-PROCEDURE-OPTIONS oder MODIFY-PROCEDURE-OPTIONS), kann der Prozedurlauf mit der Taste K2 unterbrochen werden.

Während der Unterbrechung kann der Prozedurstatus überprüft werden, die Prozedurumgebung verändert werden usw. und der Prozedurlauf mit dem Kommando RESUME-PROCEDURE oder schrittweise mit dem Kommando TRACE-PROCEDURE fortgesetzt werden.

Der Prozedurstatus kann zum Beispiel mit den vordefinierten Funktionen überprüft werden. So kann zum Beispiel die Schachtelungstiefe überprüft werden mit der Funktion PROC-LEVEL( ). Die Sysfile-Umgebung kann überprüft werden mit den Funktionen SYSCMD( ), SYSDTA( ) usw. Systemdateien können dann zum Beispiel umgelenkt werden mit den Kommandos ASSIGN-SYSDTA etc.

Variableninhalte können mit dem Kommando SHOW-VARIABLE ausgegeben werden, Strukturlayouts mit dem Kommando SHOW-STRUCTURE-LAYOUT.

Ob Variablen deklariert sind, kann mit der Funktion IS-DECLARED( ) abgefragt werden, ob eine deklarierte Variable bereits einen Inhalt hat mit der Funktion IS-INITIALIZED( ).

Haben die Variablen nicht den Inhalt, der zum aktuellen Zeitpunkt für den korrekten Ablauf der Prozedur benötigt wird, kann der Programmierer ihnen den korrekten Inhalt mit dem Kommando SET-VARIABLE zuweisen.

Wenn auf Elemente zusammengesetzter Variablen zugegriffen wird, kann zum Beispiel geprüft werden, ob die Elementnamen korrekt sind, ob die Elemente vorhanden sind usw. Dazu stehen unter anderem die vordefinierten Funktionen FIRST-VARIABLE-NAME( ) und NEXT-VARIABLE-NAME( ) zur Verfügung.
An dieser Stelle soll nur ein Hinweis darauf gegeben werden, wie der Programmierer während der Testphase im Dialog die Prozedurumgebung abfragen und auch verändern kann. Während der Prozedurunterbrechung können nicht nur SDF-P-Kommandos und -Funktionen aufgerufen werden, sondern auch BS2000-Kommandos. Die SDF-P-Funktionen sind im Detail im Kapitel „Vordefinierte Funktionen“ auf Seite 353ff beschrieben, die SDF-P-Kommandos Kapitel „SDF-P-Kommandos“ auf Seite 545ff. Die BS2000-Kommandos sind im Handbuch „Kommandos, Bd. 1-5“ [3] beschrieben.

11.5 Ablaufumgebung simulieren

Die gesamte Funktionalität von SDF-P ist in den zwei Subsystemen SDF-P und SDFPBASY (Release-Unit SDF-P-BASYS) realisiert. Die volle Funktionalität steht nur mit dem kostenpflichtigen Subsystem SDF-P zur Verfügung (siehe auch Abschnitt „Kurzbeschreibung des Produkts“ auf Seite 18).

Wird eine Prozedur in einem System erstellt, in dem SDF-P geladen ist, steht die volle SDF-P-Funktionalität zur Verfügung. Soll diese Prozedur auch in einem System, in dem nur die SDF-P-BASYS-Funktionalität zur Verfügung steht, eingesetzt werden, muss auch die Ablauffähigkeit für diese Umgebung geprüft werden. Diese Ablaufumgebung kann innerhalb der aufrufenden Task mit folgendem Kommando simuliert werden ohne das Subsystem SDF-P zu entladen:

```
/MODIFY-PROCEDURE-TEST-OPTIONS FUNCTIONALITY=*BASIC
```

Danach steht dem Aufrufer nur noch die SDF-P-Funktionalität zur Verfügung. Die einzige Ausnahme ist das Kommandos MODIFY-PROCEDURE-TEST-OPTIONS. Es wird auch im Simulationsmodus ausgeführt, damit wieder zur vollen SDF-P-Funktionalität zurückgekehrt werden kann:

```
/MODIFY-PROCEDURE-TEST-OPTIONS FUNCTIONALITY=*FULL
```

12 Nicht-S-Prozeduren umstellen

Nicht-S-Prozeduren können ohne Umstellung die Leistungen von SDF-P nicht nutzen.

Die Umstellung von Nicht-S-Prozeduren in S-Prozeduren kann schrittweise durchgeführt werden. Dabei dienen die Schritte 1 bis 6 lediglich dazu, die (ehemals) Nicht-S-Prozeduren als S-Prozeduren ablauffähig zu machen.

Mit SDF-CONV ist es möglich, Nicht-S-Prozeduren automatisch in S-Prozeduren umzuwandeln (Operand PROCEDURE-FORMAT). Dabei kann bestimmt werden, ob die Kommandosprache der Eingabeprozedur beibehalten oder in die SDF-Kommandosprache übersetzt werden soll, ob Datenzeilen in Anweisungszeilen umgewandelt werden sollen und ob die Ausgabe von SDF-Kommandos in der umgewandelten Prozedur geblockt erfolgen darf. (Näheres siehe Handbuch „SDF-CONV“ [17].)

12.1 Nicht-S-Prozedur

Die folgende Liste gilt für Prozeduren, die im Vordergrund aufgerufen werden, das heißt, für Prozeduren, die im Dialogbetrieb ablaufen sollen oder von anderen Prozeduren aufgerufen werden.

1. Prozedurkopf und Prozedurende

   a) SDF-P-Prozedurkopf erzeugen:
   Das Kommando BEGIN-PROCEDURE bzw. PROCEDURE entfernen.
   Die Prozedur hat dann implizit einen Prozedurkopf. Es gelten dann für die Prozedureigenschaften die Voreinstellungen des Kommandos SET-PROCEDURE-OPTIONS. Prozedurparameter können nicht übergeben werden.

   b) Prozedur korrekt beenden:
   Das Kommando END-PROCEDURE bzw. ENDP entfernen.
   Kommando EXIT-PROCEDURE einsetzen.

2. Wenn nötig: Prozedurparameter deklarieren:

   DECLARE-PARAMETER-Block erzeugen (oder Kommando DECLARE-PARAMETER aufrufen).
   Wenn an die S-Prozedur Prozedurparameter übergeben werden sollen, müssen diese im Prozedurkopf deklariert werden.
   Dabei sollte jeder Parameter separat mit einem Aufruf des Kommandos DECLARE-PARAMETER deklariert werden. Diese Kommandoaufrufe müssen dann in einen DECLARE-PARAMETER-Block eingebunden werden, der mit dem Kommando BEGIN-PARAMETER-DECLARATION eingeleitet und mit dem Kommando END-PARAMETER-DECLARATION abgeschlossen wird.

3. Prozedurparameter initialisieren:Nicht-S-Prozeduren (umstellen):

   Operand INITIAL-VALUE im Kommando DECLARE-PARAMETER mit einem Wert versehen.
   Wenn ein Prozedurparameter mit dem Defaultwert INITIAL-VALUE = *NONE deklariert wird, muss ihm beim Prozeduraufruf ein Wert übergeben werden. Andernfalls wird eine Fehlermeldung ausgegeben.
   Wenn ein Prozedurparameter mit INITIAL-VALUE = *PROMPT deklariert wird, wird der Wert nach dem Prozeduraufruf im Dialog abgefragt. Ist keine Dialogabfrage möglich, wird dem Prozedurparameter implizit die leere Zeichenkette zugewiesen. Wird dem Prozedurparameter mit INITIAL-VALUE ein Wert zugewiesen, braucht ihm beim oder nach dem Prozeduraufruf kein Wert übergeben zu werden. Der definierte Anfangswert (INITIAL-VALUE) wird dann standardmäßig eingesetzt.

4. Wenn nötig, Jobvariablenersetzung einstellen:

   Standardmäßig gibt es in S-Prozeduren keine Jobvariablenersetzung. Diese ist nur dann möglich, wenn im Prozedurkopf mit dem Kommando SET-PROCEDURE-OPTIONS der Operand JV-REPLACEMENT auf den Wert AFTER-BUILTIN-FUNCTION gesetzt wird (oder später im Prozedurrumpf im Kommando MODIFY-PROCEDURE-OPTIONS). Es wird aber empfohlen, die Voreinstellung nicht zu ändern. Stattdessen sollte &(jobvar) ersetzt werden durch &(JV('jobvar')).

5. Wenn nötig, Prozedureigenschaften einstellen:

   Mit dem Kommando SET-PROCEDURE-OPTIONS im Prozedurkopf Prozedureigenschaften definieren, die von den Voreinstellungen abweichen sollen. Dies gilt zum Beispiel für das Escape-Zeichen (= Fluchtsymbol) in Datensätzen oder für das Verhalten beim Auftreten von Fehlern in Datensätzen. Das Escape-Zeichen wird mit dem Operanden DATA-ESCAPE-CHARACTER definiert, das Verhalten bei Fehlern mit dem Operanden DATA-ERROR-HANDLING.

6. Prozeduraufruf umstellen:

   a) CALL-Kommando:
   Durch CALL-PROCEDURE oder INCLUDE-PROCEDURE ersetzen. Das ISP-Kommando CALL wird von SDF-P kompatibel unterstützt; intern wird es auf das erweiterte CALL-PROCEDURE-Kommando abgebildet. Daher gelten bei Prozeduraufruf mit CALL die Voreinstellungen des Kommandos CALL-PROCEDURE. Prozeduraufruf mit CALL sollte dennoch durch CALL-PROCEDURE oder INCLUDE-PROCEDURE ersetzt werden.

   b) CALL-PROCEDURE-Kommando anpassen:
   Das Kommando CALL-PROCEDURE wurde für SDF-P erweitert. Wenn diese Erweiterungen im Kommandoaufruf nicht berücksichtigt sind, gelten jeweils die Voreinstellungen. Sollen andere Einstellungen gelten, müssen die entsprechenden Operanden in den Kommandoaufruf aufgenommen werden.

   c) Prozeduraufruf mit DO-Kommando:
   Auf CALL-PROCEDURE oder INCLUDE-PROCEDURE umstellen:
   Prozeduraufruf mit dem Kommando DO wird zwar weiterhin unterstützt; da mit DO jedoch keine echte Aufrufschachtelung möglich ist, sollten Prozeduren nur noch mit den Kommandos CALL-PROCEDURE oder INCLUDE-PROCEDURE aufgerufen werden.
   Bei einer Umstellung des Prozeduraufrufs von DO nach CALL- oder INCLUDE-PROCEDURE ist das unterschiedliche Beendigungsverhalten zu beachten.

7. Gültige Sprungmarken erzeugen:

   a) Nicht-S-Marken (Format: .marke) ersetzen durch S-Marken (marke:):
   SDF-P unterstützt Marken im Nicht-S-Format nur in der obersten Blockebene, nicht in geschachtelten Blöcken. Nicht-S-Marken können nur in Sprüngen mit dem Kommando SKIP-COMMANDS (SKIP, SKIPJV, SKIPUS) sowie in Kommandos zur bedingungsabhängigen Auftragssteuerung (MODIFY-JV-CONDITIONALLY, WAIT-EVENT, ADD-CJC-ACTION, WAIT, ON) adressiert werden.

   b) Kommando SKIP-COMMANDS bzw. SKIP, SKIPJV und SKIPUS durch Kontrollflusskommandos ersetzen:
   Wenn mit SKIP-COMMANDS unbedingte Sprünge durchgeführt wurden, kann SKIP-COMMANDS durch das Kommando GOTO ersetzt werden: Sprünge mit SKIP-COMMANDS sind nur in der Schachtelungstiefe 0 möglich, das heißt, nicht in Kommandoblöcken. Sprünge innerhalb von Kommandoblöcken oder in umgebende Kommandoblöcke werden mit GOTO durchgeführt.
   Wenn mit SKIP-COMMANDS bedingte Sprünge durchgeführt wurden, kann SKIP-COMMANDS durch einen IF-Block (evtl. auch IF-BLOCK-ERROR-Block oder IF-CMD-ERROR-Block) ersetzt werden.

8. Fehlerbehandlung einfügen:

   SET-JOB-STEP-Kommandos bzw. STEP durch IF-BLOCK-ERROR oder IF-CMD-ERROR ersetzen:
   In S-Prozeduren wird Fehlerbehandlung in Fehlerbehandlungsblöcken durchgeführt, die mit den Kommandos IF-BLOCK-ERROR oder IF-CMD-ERROR eingeleitet werden. Diesen Fehlerbehandlungsblöcken darf kein SET-JOB-STEP- oder STEP-Kommando vorangehen, da SET-JOB-STEP und STEP die Fehlersituation aufheben. IF-BLOCK-ERROR oder IF-CMD-ERROR kann dann die ursprüngliche Fehlersituation nicht mehr erkennen.
   Zu beachten ist, dass IF-BLOCK-ERROR keine Schalter zurücksetzt!

9. Gleichheitszeichen als erstes Zeichen im Kommandoaufruf löschen:

   Bei einigen Kommandos darf auf den Kommandonamen ein beliebiges Zeichen folgen. Wenn dieses erste Zeichen nach dem Kommandonamen ein Gleichheitszeichen ist, interpretiert SDF-P die Kommandozeile als Wertzuweisung an eine Variable, deren Variablenname der Kommandoname ist.

10. Auftragsschalter durch Variablen ersetzen:

    Der Ablauf von S-Prozeduren sollte nicht über Auftragsschalter gesteuert werden. SDF-P bietet zur Ablaufsteuerung die Kontrollflusskommandos, mit denen sich Verzweigungen und Schleifen programmieren lassen. Bei der Abfrage der entsprechenden Bedingungen werden Variablen eingesetzt.

    Hinweis

    Bei Abhängigkeiten zu Nicht-S-Prozeduren, die die Steuerung über Auftragsschalter nutzen,kann es notwendig sein, dass Auftragsschalter weiterhin abgefragt oder gesetzt werden müssen. In diesem Fall können Auftragschalter auch über das implizit deklarierte Array SYSSWITCH abgefragt oder modifiziert werden (siehe Abschnitt „Reservierte Variablennamen“ auf Seite 159).

11. Wenn nötig: Anweisungen kennzeichnen:

    Prozedurzeilen, die Anweisungen enthalten, müssen mit zwei Schrägstrichen beginnen.

12.2 Enter-Job

Die folgende Liste gilt für Prozeduren (Enter-Dateien), die mit dem Kommando ENTER-JOB aufgerufen werden, das heißt für Prozeduren, die im Stapelbetrieb ablaufen sollen.

1. Prozedurkopf und Prozedurende

   a) SDF-P-Prozedurkopf erzeugen:
   Das Kommando SET-LOGON-PARAMETERS bzw. LOGON entfernen.
   Bei S-Prozeduren wird nicht zwischen Hintergrund- und Vordergrund-Prozeduren unterschieden. Daher darf eine S-Prozedur kein SET-LOGON-PARAMETERS bzw. LOGON-Kommando enthalten.
   Wenn das SET-LOGON-PARAMETERS bzw. LOGON-Kommando entfernt ist, hat die Prozedur implizit einen Prozedurkopf. Für die Prozedureigenschaften gelten die Voreinstellungen.

   b) Prozedur korrekt beenden:
   Das Kommando EXIT-JOB (mit MODE=*NORMAL) bzw. LOGOFF entfernen.
   Das Kommando EXIT-PROCEDURE einsetzen.
   Da in S-Prozeduren nicht zwischen Hintergrund- und Vordergrund-Prozeduren unterschieden wird, muss auch das Kommando EXIT-JOB bzw. LOGOFF am Prozedurende entfernt werden.

   c) Prozedur abnormal beenden:
   Innerhalb eines Enter-Jobs führt das Beenden einer Prozedur mit /EXIT-PROCEDURE ERROR=*YES in einer überwachenden MONJV trotzdem zum Endestatus $T („normales Ende“). Enthält die bisherige Enter-Datei das Kommando EXIT-JOB mit MODE=*ABNORMAL bzw. ABEND, müssen deshalb folgende Fälle unterschieden werden:
   – Soll der Auftrag nicht über eine MONJV überwacht werden oder ist der MONJV-Endestatus $T gewünscht, so können auch diese Kommmandos durch /EXIT-PROCEDURE ERROR=*YES ersetzt werden.
   – Soll der Auftrag mit einer MONJV überwacht und der MONJV-Endestatus $A erreicht werden, so muss das Kommando EXIT-JOB mit MODE=*ABNORMAL verwendet werden.

   Hinweis

   Es ist zu beachten, dass beim Beenden einer Hintergrund-Prozedur mit /EXIT-JOB die S.-Dateien nicht gelöscht werden.

2. Wenn eine Hintergrund-Prozedur von einer anderen Prozedur aus aufgerufen wird:

   Prozeduraufruf anpassen bzw. das Kommando ENTER-JOB durch ENTER-PROCEDURE ersetzen.
   (Prozeduren, die unter SDF-P als Hintergrund-Prozedur laufen sollen, werden mit dem Kommando ENTER-PROCEDURE aufgerufen, das intern wiederum ein ENTER-JOB-Kommando absetzt.)

3. Für die weiteren Umstellungsschritte gilt die Beschreibung zur Umstellung von Nicht-S-Prozeduren.

12.3 Kompatibilität der Kommandos zur Prozedursteuerung

In diesem Abschnitt werden die Einschränkungen beschrieben, die für Nicht-SDF-P-Kommandos in S-Prozeduren gelten sowie die Einschränkungen für SDF-P-Kommandos beim Prozedurablauf.

(Sprung-)Marken und AID-Sequenzen

(Sprung-)Marken

Marken des Formats .marke können nur in Verbindung mit den SKIP-Kommandos und den Kommandos zur bedingungsabhängigen Auftragssteuerung verwendet werden (siehe unten).
Marken des Formats marke: können nur mit SDF-P-Kommandos verwendet werden.

AID-Sequenzen

AID-Sequenzen sind Folgen von Kommandos an AID, die jeweils durch ein Semikolon voneinander getrennt sind.
In diesen Sequenzen dürfen keine SDF-P-Kontrollflusskommandos verwendet werden.
AID-Kommandos, denen eine Kommando- bzw. Unterkommandoliste folgt, dürfen nicht durch Variablenersetzung erzeugt werden.

Nicht-SDF-P-Kommandos

ADD-CJC-ACTION (bzw. ON)
In ADD-CJC-ACTION-Blöcken sind nur ENTER-JOB-, ENTER-PROCEDURE- und MODIFY-JV-Kommandos erlaubt.

BEGIN-PROCEDURE (bzw. PROCEDURE)
Wird nicht unterstützt.

CALL-PROCEDURE (bzw. CALL)
Ein Semikolon außerhalb von Klammern (bzw. außerhalb von Kommentaren) wird als Kommandotrenner interpretiert.

CANCEL-PROCEDURE
Wenn das Kommando CANCEL-PROCEDURE eingesetzt wird, um eine Fehlersituation zu beenden, müssen ihm folgende Kommandos vorausgehen:

```
IF-BLOCK-ERROR
END-IF
```

DO (ISP-Kommando)
Darf nicht verwendet werden beim Aufruf von Prozeduren, in denen Prozedurparameter mit TRANSFER-TYPE = *BY-REFERENCE deklariert werden.
Ein Semikolon außerhalb von Klammern wird als Kommandotrenner interpretiert.

EXIT-JOB (bzw. ABEND)
Wenn das Kommando EXIT-JOB eingesetzt wird, um in einer Fehlersituation den Auftrag zu beenden, so muss es in folgende Kommandos eingeschlossen werden:

```
IF-BLOCK-ERROR
END-IF
```

INTR (ISP-Kommando)
Ein Semikolon außerhalb von Klammern wird als Kommandotrenner interpretiert.

LOGOFF
Wenn das Kommando LOGOFF eingesetzt wird, um in einer Fehlersituation den Auftrag zu beenden, so muss es in folgende Kommandos eingeschlossen werden:

```
IF-BLOCK-ERROR
END-IF
```

LOGON
Wird nicht unterstützt.

PAUSE (ISP-Kommando)
Das Gleichheitszeichen darf nicht erstes signifikantes Zeichen sein. Das Kommando wird sonst als Wertzuweisung interpretiert (SET-VARIABLE ohne Kommandonamen).
Ein Semikolon außerhalb von Klammern wird als Kommandotrenner interpretiert.

REMARK
Wie bei SDF-Kommandos üblich, muss beachtet werden, dass
– das Gleichheitszeichen nicht das erste signifikante Zeichen sein darf. Das Kommando wird sonst als Wertzuweisung interpretiert (SET-VARIABLE ohne Kommandonamen).
– ein Semikolon außerhalb von Klammern als Kommandotrenner interpretiert wird.
– Klammern, einfache und doppelte Anführungszeichen nicht einzeln, sondern nur in Paaren auftreten dürfen.

SET-JOB-STEP (STEP)
Darf nicht durch Variablenersetzung entstehen.

SET-LOGON-PARAMETERS
Wird nicht unterstützt.

SKIP-COMMANDS (ISP-Kommandos SKIP, SKIPJV, SKIPUS)
Kann nur in Schachtelungstiefe 0 hineinspringen, das heißt, nur in die oberste Blockebene.

TYPE (ISP-Kommando)
Das Gleichheitszeichen darf nicht erstes signifikantes Zeichen sein. Das Kommando wird sonst als Wertzuweisung interpretiert (SET-VARIABLE ohne Kommandonamen).
Ein Semikolon außerhalb von Klammern wird als Kommandotrenner interpretiert.

WAIT-EVENT (bzw. WAIT), MODIFY-JV-CONDITIONALLY
Da hier Sprungkommandos nur mit Nicht-S-Marken möglich sind, sollte das Kommando WAIT-EVENT nicht innerhalb von Blöcken verwendet werden

Selbstdefinierte Kommandos

Mit SDF-A können eigene Kommandos definiert werden. Solche Kommandos können mit dem Operandenwert „Kommandorest“ definiert sein, in dem semantisch Semikolons zugelassen sind.

Werden solche Kommandos in S-Prozeduren eingesetzt, ist - wie in normalen SDF-Kommandos üblich - zu beachten, dass

– das Gleichheitszeichen nicht das erste signifikante Zeichen sein darf. Das Kommando wird sonst als Wertzuweisung interpretiert (SET-VARIABLE ohne Kommandonamen).
– ein Semikolon außerhalb von Klammern als Kommandotrenner interpretiert wird.
– Klammern, einfache und doppelte Anführungszeichen nicht einzeln, sondern nur in Paaren auftreten dürfen.

SDF-P-Kommandos

EXIT-PROCEDURE
Wenn das Kommando EXIT-PROCEDURE eingesetzt wird, um in einer Fehlersituation den Auftrag zu beenden, so muss es in folgende Kommandos eingeschlossen werden:
```
IF-BLOCK-ERROR
END-IF
```

MODIFY-PROCEDURE-TEST-OPTIONS
Hat keine Wirkung auf Nicht-S-Prozeduren.

Einschränkungen bei Datensätzen

Handelt es sich bei der S-Prozedur um eine ISAM-Datei, so werden die darin enthaltenen Datensätze an das lesende Programm ohne ISAM-Schlüssel übergeben.

12.4 Umstellungsbeispiele

Beispiel 1: Speicherplatz reorganisieren

Diese Prozedur reorganisiert den Speicherplatz für alle Dateien einer Benutzerkennung. Wenn eine Datei eine PLAM-Bibliotkek ist, werden alle ihre Elemente in eine neue Bibliothek kopiert.

Die Benutzerkennung muss mit führendem $-Zeichen und abschließendem Punkt angegeben werden. Es gibt keine Defaultwerte.

a) Nicht-S-Prozedur

```
/BEGIN-PROCEDURE LOGGING=N,    -
/                PARAMETERS=YES(   -
/                  PROCEDURE-PARAMETERS=(  -
/                     &USERID=),   -
/                  ESCAPE-CHARACTER='&')
/REMARK +-------------------------------------------------------------------+"
/REMARK |                                                                   |
/REMARK | This procedure compacts all files contained on a user-id.         |
/REMARK | If the file is a PLAM library then all elements are               |
/REMARK | duplicated in a new library.                                      |
/REMARK | The user-id must be given with the leading dollar sign            |
/REMARK | and with the trailing point.                                      |
/REMARK | There is no default value.                                        |
/REMARK |                                                                   |
/REMARK +-------------------------------------------------------------------+"
/REMARK &USERID
/ASSIGN-SYSOUT TO=*DUMMY
/SHOW-FILE-ATTR &USERID,INFO=NAME-AND-SPACE,   -
/               OUTPUT=#LST(FORM-NAME=FILE-NAME)
/ASSIGN-SYSOUT TO=*PRIMARY
/ASSIGN-SYSDTA TO=*SYSCMD
/MOD-JOB-SWITCHES ON=(1,4,5)
/START-EXECUTABLE-PROGRAM $EDT
@@READ '#LST'
@@COL1ON&INSERT'/CALL-PROC #PROC2,P-P=('
@@SUFFIX&WITH')'
@@RENUM
@@CRO.001W'/BEGIN-PROCEDURE LOGGING=N'
@@CRO.002W'/ASSIGN-SYSOUT TO=#OUTBEFORE'
@@CRO.003W'/SHOW-FILE-ATT &USERID,INFO=SPACE-SUMMARY'
@@CRO.004W'/ASSIGN-SYSOUT TO=*PRIMARY'
@@CR$+.01W'/ASSIGN-SYSOUT TO=#OUTAFTER'
@@CR$+.01W'/SHOW-FILE-ATT &USERID,INFO=SPACE-SUMMARY'
@@CR$+.01W'/ASSIGN-SYSOUT TO=*PRIMARY'
```

```
@@CR$+.O1W'/END-PROCEDURE'
@@WR'#PROC1' OVER
@@DELETE
@@CR$+.O1W'/BEGIN-PROC LOGGING=N,  -'
@@CR$+.O1W'/           PARAMETERS=YES(  -'
@@CR$+.O1W'/               PROC-PARAM=(&FILE),-'
@@CR$+.O1W'/               ESCAPE-CHAR=''&'')'
@@CR$+.O1W'/COPY-FILE &FILE,&FILE..WORK,PROTECTION=SAME'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=ERASE'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=END'
@@CR$+.O1W'/.ERASE DELETE-FILE &FILE,OPTION=DATA'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=FILE'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/DELETE-FILE &FILE..WORK'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=END'
@@CR$+.O1W'/.FILE SET-JOB-STEP'
@@CR$+.O1W'/FILE &FILE,SPACE=(100,20),FCBTYPE=PAM'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=LMS'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/COPY-FILE &FILE..WORK,&FILE'
@@CR$+.O1W'/DELETE-FILE &FILE..WORK,IGNORE-PROTECTION=ACCESS'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=END'
@@CR$+.O1W'/.LMS SET-JOB-STEP'
@@CR$+.O1W'/ASSIGN-SYSDTA TO=*SYSCMD'
@@CR$+.O1W'/MOD-JOB-SWITCHES ON=(1,2,3,4,5)'
@@CR$+.O1W'/START-EXECUTABLE-PROGRAM $LMS,MONJV=#BIDON'
@@CR$+.O1W'LIB &FILE..WORK,IN'
@@CR$+.O1W'LIB &FILE,OUT,NEW'
@@CR$+.O1W'DUP* */*'
@@CR$+.O1W'END'
@@CR$+.O1W'/MOD-JOB-SWITCHES OFF=(1,2,3,4,5)'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/SKIP-COMMANDS IF=JV(((#BIDON,4,4) = ''0000'')),-'
@@CR$+.O1W'/               TO-LABEL=AVEND'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/COPY-FILE &FILE..WORK,&FILE'
@@CR$+.O1W'/DELETE-FILE &FILE..WORK'
@@CR$+.O1W'/SET-JOB-STEP'
@@CR$+.O1W'/SKIP-COMMANDS TO-LABEL=END'
@@CR$+.O1W'/.AVEND SET-JOB-STEP'
@@CR$+.O1W'/DELETE-FILE &FILE..WORK'
@@CR$+.O1W'/.END SET-JOB-STEP'
@@CR$+.O1W'/MOD-FILE-ATT &FILE,SUPPORT=ANY-DISK(RELEASE(9999))
@@CR$+.O1W'/W-TEXT ''File &FILE compacted'''
@@CR$+.O1W'/SET-JOB-STEP'
```

```
@@CR$+.01W'/END-PROCEDURE'
@@WR'#PROC2'OVER
@@H
/CALL-PROCEDURE #PROC1
/START-EXECUTABLE-PROGRAM $EDT
@@REA'#OUTBEFORE'
@@REA'#OUTAFTER'
@@SET #I5 = SUBSTR 1:38-42:
@@SET #I6 = SUBSTR 2:38-42:
@@SET #I7 = #I5 - #I6
@@DELETE
@@SET #S5 = CHAR #I5
@@SET #S6 = CHAR #I6
@@SET #S7 = CHAR #I7
@@CR #S1:'Space before : ',#S5
@@CR #S2:'Space after  : ',#S6
@@CR #S3:'               ------------'
@@CR #S4:'Space won    : ',#S7
@@PRINT #S1 N
@@PRINT #S2 N
@@PRINT #S3 N
@@PRINT #S4 N
@@H
/MOD-JOB-SWITCHES OFF=(1,4,5)
/ASSIGN-SYSDTA TO=*SYSCMD
/END-PROC
```

b) S-Prozedur

```
/&* +-----------------------------------------------------------------+
/&* |                                                                 |
/&* | This procedure compacts all files contained on a user-id.       |
/&* | If the file is a PLAM library then all elements are             |
/&* | duplicated in a new library.                                    |
/&* | The user-id is given in parameters with or without the          |
/&* | leading dollar sign and with or without the trailing point.     |
/&* | If no parameter is given then the current user-id is taken.     |
/&* |                                                                 |
/&* +-----------------------------------------------------------------+
/DECLARE-PARAMETER USER-ID(INITIAL-VALUE=USER-ID,TRANSFER-TYPE=BY-VALUE)
/IF (SUBSTR(USER-ID,1,1)<>'$')
/   "THEN" USER-ID = '$' // USER-ID
/END-IF
/IF (SUBSTR(USER-ID,LENGTH(USER-ID),1)<>'.')
/   "THEN" USER-ID = USER-ID // '.'
/END-IF
/DECLARE-VARIABLE FS(TYPE=STRUCT(*DYNAMIC)),MULT-ELEM=LIST
```

```
/DECLARE-VARIABLE VARLOOP(TYPE=STRUCT(*DYNAMIC))
/EXEC-CMD CMD=(SHOW-FILE-ATTR &USER-ID,INFO=NAME-AND-SPACE),-
/STRUCTURE-OUTPUT=FS,TEXT-OUTPUT=*NONE
/DECLARE-VARIABLE SPACEBEFORE(INIT=0,TYP=*INTEGER)
/FOR VARLOOP = *LIST(FS)
/    SPACEBEFORE = SPACEBEFORE + VARLOOP.F-SIZE
/    IF (IS-LIBRARY(VARLOOP.F-NAME))
/       "THEN" LIBBLOCK: BEGIN-BLOCK DATA-INSERTION=YES
/             COPY-FILE FROM-FILE=&(VARLOOP.F-NAME),-
/                       TO-FILE=&(VARLOOP.F-NAME).WORK,PROTECTION=SAME
/             IF-BLOCK-ERROR
/                   EXIT-BLOCK LIBBLOCK
/             END-IF
/             MODIFY-JOB-SWITCHES ON=(1,4)
/             ASSIGN-SYSDTA TO=*SYSCMD
/             DELETE-FILE &(VARLOOP.F-NAME)
/             IF-BLOCK-ERROR
/               DELETE-FILE &(VARLOOP.F-NAME).WORK,OPTION=DATA
/               EXIT-BLOCK LIBBLOCK
/             END-IF
/             START-EXE FROM-FILE=$LMS
/             SEND-DATA 'LIB &(VARLOOP.F-NAME).WORK,IN'
/             SEND-DATA 'LIB &(VARLOOP.F-NAME).NEW.OUT'
/               SEND-DATA 'DUP* */*'
/               SEND-DATA 'END'
/             ASSIGN-SYSDTA TO=*PRIMARY
/             MODIFY-JOB-SWITCHES OFF=(1,4)
/             IF-BLOCK-ERROR
/                COPY-FILE &(VARLOOP.F-NAME).WORK,-
/                          &(VARLOOP.F-NAME)
/                ELSE
/               WR-TEXT 'Library &(VARLOOP.F-NAME) compacted'
/             END-IF
/             DELETE-FILE &(VARLOOP.F-NAME).WORK,IGNORE-PROTECTION=ACCESS
/             MOD-FILE-ATTR &(VARLOOP.F-NAME),-
/                           SUPPORT=ANY-DISK(SPACE=RELEASE(10000))
/              END-BLOCK
/         ELSE
/              MOD-FILE-ATTR &(VARLOOP.F-NAME),-
/                            SUPPORT=ANY-DISK(SPACE=RELEASE(10000))
/              IF-BLOCK-ERROR
/                  ELSE
/                  WR-TEXT 'File &(VARLOOP.F-NAME) compacted'
/              END-IF
/    END-IF
/END-FOR
/IF-BLOCK-ERROR
/END-IF
```

```
/DECLARE-VARIABLE FS2(TYP=STRING),MULT-EL=LIST
/EXEC-CMD CMD=(SHOW-FILE-ATTR &USER-ID,INFO=SPACE-SUMMARY),-
/         TEXT-OUTPUT=FS2
/SPACEAFTER = INTEGER (SUBSTR (FS2#,37,5))
/WR-TEXT 'Space before : &SPACEBEFORE'
/WR-TEXT 'Space after  : &SPACEAFTER'
/WIN = SPACEBEFORE - SPACEAFTER
/WR-TEXT '                -----'
/WR-TEXT 'Spaced won : &WIN'
/EXIT-PROC
```

Beispiel 2: Dateien mithilfe von Mustern kopieren

In der folgenden Prozedur werden zunächst Dateien über Muster ausgewählt. Anschließend werden sie unter eine neue Katalogkennung, eine neue Benutzerkennung oder ein neues Präfix kopiert.

Zur Veranschaulichung:
`CALL-PROC copy, PROC-PAR = (SOURCE=xxx, TARGET=yyy)`

für xxx kann stehen:
`:CAT: ,$USERID., *STRING*` oder alles, was von SHOW-FILE-ATTR akzeptiert wird.

für yyy kann stehen:
`:CAT: ,$USERID2.` oder jedes Präfix mit abschließendem Punkt.

a) Nicht-S-Prozedur

```
/BEGIN-PROCEDURE LOGGING=N,                          -
/                PARAMETERS=YES(                     -
/                  PROCEDURE-PARAMETERS=(            -
/                       &SOURCE=,                    -
/                       &TARGET=),                   -
/                  ESCAPE-CHARACTER='&')
/REMARK +-------------------------------------------------------------+
/REMARK |                                                             |
/REMARK | This procedure enables the user to copy some files selected |
/REMARK | with wildcards under a new cat-id, a new user-id or a new   |
/REMARK | prefix.                                                     |
/REMARK | Examples :                                                  |
/REMARK |   CALL-PROC copy,PROC-PAR=(SOURCE=xxx,TARGET=yyy)           |
/REMARK |   where xxx could be :CAT: , $USERID. , *STRING* or         |
/REMARK |                        everything accepted by SHOW-FILE-ATT |
/REMARK |         yyy could be :CAT2: , $USERID2. or any PREFIX with  |
/REMARK |                        a trailing point                     |
/REMARK |                                                             |
/REMARK +-------------------------------------------------------------+
/REMARK &SOURCE
/REMARK &TARGET
/ASSIGN-SYSOUT TO=*DUMMY
/ASSIGN-SYSLST TO=#LST
/SHOW-FILE-ATTR &SOURCE,LIST=((SYSLST),FILENAM)
/ASSIGN-SYSLST TO=*PRIMARY
/ASSIGN-SYSOUT TO=*PRIMARY
/ASSIGN-SYSDTA TO=*SYSCMD
/MODIFY-JOB-SWITCHES ON=(1,4,5)
/START-EXECUTABLE-PROGRAM $EDT
@@PROC 1
@@REA '#LST'
```

```
@@PROC 2
@@COPY &(1)
@@ON&FIND '.',1 DELETE PREFIX
@@DELETE&:1-1
@@PREFIX & WITH '/            TO-FILE=&TARGET'
@@RENUM 1.5(1)
@@END
@@SUFFIX & WITH '.-'
@@PREFIX & WITH '/COPY-FILE FROM-FILE  ='
@@COPY &(2)
@@RENUM
@@CR 0.01W'/BEGIN-PROCEDURE LOGGING=N'
@@CR$+.01W'/END-PROCEDURE'
@@WR'#LST' OVER
@@H
/MODIFY-JOB-SWITCHES OFF=(1,4,5)
/ASSIGN-SYSDTA TO=*PRIMARY
/CALL-PROC #LST
/END-PROCEDURE
```

b) S-Prozedur

```
/BEGIN-PARAMETER-DECLARATION
/  DECLARE-PARAMETER SOURCE(INIT=*PROMPT, TYPE=*STRING)
/  DECLARE-PAREMETER TARGET(INIT=*PROMPT, TYPE=*STRING)
/END-PARAMETER-DECLARATION
/&*+-------------------------------------------------------------------+
/&*|                                                                   |
/&*| This procedure enables the user to copy some files selected with  |
/&*| wildcards under a new cat-id, a new user-id, or a new prefix.     |
/&*| Examples :                                                        |
/&*|  CALL-PROC copy,PROC-PAR=(SOURCE=xxx,TARGET=yyy)                  |
/&*|  where xxx could be :CAT: , &USERID. , *STRING* or everything     |
/&*|                           accepted by SHOW-FILE-ATTRIBUTES        |
/&*|          yyy could be :CAT2: , &USERID. or any PREFIX with a      |
/&*|                           trailing point.                         |
/&*|                                                                   |
/&*+-------------------------------------------------------------------+
/DECLARE-VARIABLE FS(TYPE=STRUCT(*DYNAMIC)),MULT-ELEM=LIST
/DECLARE-VARIABLE VARLOOP(TYPE=STRUCT(*DYNAMIC))
/DECLARE-VARIABLE TARGET2(TYPE=STRING)
/EXEC-CMD CMD=(SHOW-FILE-ATTR &SOURCE,-
/INFO=NAME-AND-SPACE),-
/STRUCTURE-OUTPUT=FS,TEXT-OUT=*NONE
/FOR VARLOOP = *LIST(FS)
/    TARGET2 = TARGET // -
/              VARLOOP.SHORT-F-NAME
```

```
/    COPY-FILE FROM = &(VARLOOP.F-NAME),-
/              TO  = &(TARGET2)
/    IF-BLOCK-ERROR
/      "THEN" WR-TEXT 'File &(VARLOOP.F-NAME) not copied'
/       ELSE
/            WR-TEXT 'File &(VARLOOP.F-NAME) copied on &TARGET2'
/    END-IF
/END-FOR
/EXIT-PROC
```

13 Programmschnittstellen

In diesem Kapitel werden die Programmschnittstellen für die Systembetreuung und für den nicht-privilegierten Anwender beschrieben.

13.1 Programmschnittstellen für die Systembetreuung

Hier sind die Programmschnittstellen dargestellt, mit denen die Systembetreuung eigene Funktionen schreiben kann sowie die Exitroutinen, die sie benutzen kann.

13.1.1 Assembler-Makros zur Erzeugung eigener Funktionen

Der Systembetreuung stehen zur Erzeugung von eigenen Funktionen Assembler-Makros zur Verfügung, die im Folgenden beschrieben werden.

BIFDEF

Der Makro BIFDEF generiert den Tabelleneintrag, der den Namen der Funktion mit den Adressen des Ausführungmodul-Eintrags und des Syntaxbeschreibung-Eintrags verbindet.

Zu weiteren Informationen siehe Abschnitt „Systemverwalter-Funktionen“ auf Seite 248.

Operation	Operanden
BIFDEF	MF = L
	NAME = <name 1..20>
	SYNTAX = <name 1..8>
	CODE = <name 1..8>

Operandenbeschreibung

MF = L
LIST-Form des Makroaufrufs.

NAME = <name 1..20>
Name der Systemverwalter-Funktion (Erster Buchstabe sollte ein „X“ sein).

SYNTAX = <name 1..8>
Name des Verweispunkts für die Syntaxbeschreibung (der schon im Makro BIFDESC angegeben wurde).

CODE = <name 1..8>
Name des Ausführungsmodul-Eintrags für die Funktion (die von der Systembetreuung geschrieben wurde).

BIFDESC

Der Makroaufruf BIFDESC enthält die Syntaxbeschreibung von Systemverwalter-Funktionen.

BIFDESC definiert eine statische Struktur, die ausschließlich vom Subsystem SDF-P-BIF benutzt wird.

Zu weiteren Informationen siehe Abschnitt „Systemverwalter-Funktionen“ auf Seite 248.

Operation	Operanden
BIFDESC	NAME = <name 1..20>
	,ENTRYN = <name 1..8> / (*CSECT,<name 1..8>)
	,PARLIST = *NONE / list-poss(2000): (parameter-specification)
	,PARFORM = *BY-VALUE / *STRING
	,VALTYPE = *STRING / *INTEGER / *BOOLEAN / *ANY

Operandenbeschreibung

NAME = <name 1..20>
Name der Systemverwalter-Funktion (Erster Buchstabe sollte ein „X“ sein).

ENTRYN
Name des Verweispunkts für die Syntaxbeschreibung.

= <name 1..8>
Bezeichnet nur den Verweispunkt für die Syntaxbeschreibung. In diesem Fall wird keine CSECT generiert.

= (*CSECT,<name 1..8>)
Gibt den Verweispunkt für die Syntaxbeschreibung an und löst anschließend die Generierung der CSECT aus.

PARLIST
Gibt eine Liste von Operanden an.

= *NONE
Es gibt keine Operanden.

= list-poss(2000): (parameter-specifikation)
Gibt Spezifikationen für Listen von Operanden an. Auch wenn nur eine Spezifikation angegeben wird, müssen Klammern benutzt werden.

parameter-specifikation = parameter-name,parameter-type[,default-value [,keyword-list]]
Gibt Operandennamen, Operandentypen und optionale Defaultwerte und Keywords (Schlüsselwortwerte) an.

parameter-name = <name 1..20>
Name des Operanden.

parameter-type = *STRING / *INTEGER / *BOOLEAN / *ANY / *KEYWORD
Typ des Operanden.

default-value = <integer -2^{31}..2^{31}-1> / <c-string 0..4096> / TRUE / FALSE / ON / OFF / YES / NO / *<name 1..30>
Defaultwert, der benutzt wird, wenn der Operand nicht vom Anwender angegeben wird. Wenn der Defaultwert mit „*“ beginnt, ist er vom Typ KEYWORD. Wenn er in Hochkommas eingeschlossen ist, ist er vom Typ STRING.

keyword-list = list-poss(2000): (keyword)
Liste von akzeptierten Keywords, muss auch dann in Klammern eingeschlossen werden, wenn nur ein Keyword angegeben wird.

keyword = *<name 1..30>
Name des Keywords mit führendem Stern.

PARFORM =
Form des Operanden

= *BY-VALUE
Der Operand ist ein Wert.

= *STRING
Der Operand ist eine Zeichenkette.

VALTYPE = *STRING / *INTEGER / *BOOLEAN / *ANY
Typ des Rückgabewerts

BIFMDL1

Der Makroaufruf BIFMDL1 generiert die DSECT für die Werte der Systemverwalter-Funktionen. Es wird die Struktur jedes Elements der Operandenliste für das Ausführungsmodul beschrieben.

Zu weiteren Informationen siehe Abschnitt „Systemverwalter-Funktionen“ auf Seite 248.

Operation	Operanden
BIFMDL1	MF = D ,PREFIX = B / prefix ,MACID = IF1 / macid

MF = D
DSECT-Form des Makroaufrufs: erzeugt eine DSECT für die Operandenliste.

PREFIX = B / prefix
Definiert das erste Zeichen der generierten Namen. Voreinstellung: B.

MACID = IF1 / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: IF1.

DSECT

```
         BIFMDL1 MF=D
BIF1     DSECT
               *,##### PREFIX=B, MACID=IF1 #####
BIF1VLG  DS    F             VALUE LENGTH
BIF1VPT  DS    A             VALUE POINTER
BIF1VTY  DS    X             VALUE TYPE
BIF1STR  EQU   X'01'         -- VALUE_STRING
BIF1INT  EQU   X'02'         -- VALUE_INTEGER
BIF1BOOL EQU   X'03'         -- VALUE_BOOLEAN
BIF1KEYW EQU   X'04'         -- VALUE_KEYWORD
BIF1RES1 DS    XL1           RESERVED
BIF1RES2 DS    XL1           RESERVED
BIF1RES3 DS    XL1           RESERVED
BIF1#    EQU   *-BIF1VLG     LENGTH
```

Diese DSECT kann bei jedem Element der Operandenliste angewendet werden. Durch sie werden sowohl Operanden als auch Returncodes beschrieben.

Das Feld, auf das BIF1VPT (Value pointer) zeigt, ist:

- für Stringwerte: der String selbst
- für Integerwerte:
  - wenn PARFORM = *BY-VALUE: ein „Vollwort“, das einen Integerwert repräsentiert
  - wenn PARFORM = *STRING: ein String, der die EBCDIC-Repräsentation eines Integerwerts enthält (von einem bis 11 Zeichen)
- für Booleanwerte:
  - wenn PARFORM = *BY-VALUE: X'00' für FALSE oder X'01' für TRUE
  - wenn PARFORM = *STRING: ein Stringwert, der 'FALSE' oder 'TRUE' (vier oder fünf Zeichen)
- für Keyword-Werte: ein Stringwert mit einem führendem Stern.

Hinweis

Das Format der Returncode-Werte hängt vom Operanden PARFORM ab.

BIFMDL2

Der Makroaufruf BIFMDL2 generiert die DSECT für den Returncode der Systemverwalter-Funktionen. Es wird der Returncode beschrieben, der vom Ausführungsmodul zurückgegeben wird.

Zu weiteren Informationen siehe Abschnitt „Systemverwalter-Funktionen“ auf Seite 248.

Operation	Operanden
BIFMDL2	MF = D ,PREFIX = B / prefix ,MACID = IF2 / macid

Operandenbeschreibung

MF = D
DSECT-Form des Makroaufrufs: erzeugt eine DSECT für die Operandenliste.

PREFIX = B / prefix
Definiert das erste Zeichen der generierten Namen. Voreinstellung: B

MACID = IF2 / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: IF2.

DSECT

```
BIF2D      DSECT
BIF2SC2    DS       X           Subcode2
BIF2SC1    DS       X           Subcode1
BIF2MID    DS       CL7         Msg-id
BIF2#      EQU      *-BIF2SC2
```

13.1.2 Exitroutinen

Die Systembetreuung muss das Verhalten von SDF-P berücksichtigen:

– bei der Belegung des Registers 12 im TP-Modus: Register 12 muss die Adresse des Programm-Managers enthalten
– bei der Anwendung der System Exits 080 und 081 (SYSCMD-Exits)

Die SYSCMD-Exits sind ausführlich beschrieben im Handbuch „System Exits“ [24]. Darüber hinaus wirkt sich SDF-P bei den SYSCMD-Exits 080 und 081 auf das Umwandeln von SDF-Kommandos aus und auf den Zeitpunkt, an dem der SYSCMD-Exit aktiviert wird.

Kommandoumwandlung in SYSCMD-Exit

SDF-P-Kommandos dürfen in SYSCMD-Exit-Routinen nicht verändert werden; Veränderungen von SDF-P-Kommandos führen zum Fehler oder werden nicht erkannt.

Wird in einer Exit-Routine ein SDF-Kommandoaufruf, dem in einer Prozedur der Kommandoaufruf IF-CMD-ERROR folgt, in mehrere Kommandos aufgeteilt (1:n-Wandlung), so gilt IF-CMD-ERROR für alle erzeugten Kommandos.

Wenn in einer Exit-Routine ein Kommando einmal nicht verändert wird, wird diese Exit-Routine für das Kommando nicht mehr aktiviert.

Aktivierung von SYSCMD-Exits

Wird ein SYSCMD-Exit während des Ablaufs einer Prozedur aktiviert, so hat er keine Wirkung auf die laufende Prozedur.

Tritt während des Prozedurablaufs ein Fehler auf, wird im Fehlermodus kein SYSCMD-Exit aktiviert.

Bei Sprüngen in der Prozedur mit dem Kommando SKIP-COMMANDS wird ebenfalls kein SYSCMD-Exit aktiviert; das Sprungziel wird nur einmal zur Verfügung gestellt.

Ein SYSCMD-Exit zur Kommandoumwandlung wird nicht mehr aktiviert, wenn beim ersten Aufruf der Exit-Routine das Kommando nicht verändert wurde.

13.2 Programmschnittstellen für den Anwender

SDF-P bietet dem Assembler-Programmierer folgende Schnittstellen:

Makro	**Funktion**
CLIEXPR	SDF-P-Ausdrücke auswerten
CLIGET	Unterbrechungsschutz für eine Prozedur abfragen
CLISET	expliziten Schutz vor Programmunterbrechungen vereinbaren
CMD	SDF-P-Kommandos aus einem Programm absetzen
GETVAR	Lesen von einfachen und zusammengesetzten Variablen
PUTVAR	Schreiben von einfachen Variablen
SHOWSSA	über Variablenstrom-Zuweisung informieren
TRANSVV	Variablen mittels Variablenstrom übertragen
VARINF	zusammengesetzte Variablen bearbeiten (modifizieren)

Über die Makros PUTVAR, GETVAR, SHOWSSA, TRANSVV und VARINF können S-Variablen angesprochen werden, das heißt die Variablen, die auch über die SDF-P-Kommandoschnittstelle angesprochen werden können.

Geltungsbereich von Variablen

Es gibt für S-Variablen zwei Geltungsbereiche : prozedurlokal und taskglobal.

Der Geltungsbereich „prozedurlokal“ wird auf Kommandoebene im Operanden SCOPE mit SCOPE = *PROCEDURE / *CURRENT eingestellt oder angesprochen. Auf Programmebene wird dieser Geltungsbereich mit SCOPE=*VISIBLE eingestellt oder angesprochen. Zu beachten ist, dass prozedurlokale Variablen nur bis Prozedurende existieren, das Programm aber noch weiterlaufen kann. Auf prozedurlokale Variablen kann dann innerhalb des Programms nicht mehr zugegriffen werden, es kommt zum Fehler.

Der Geltungsbereich „taskglobal“ wird auf Kommandoebene im Operanden SCOPE mit SCOPE = *TASK eingestellt oder angesprochen. Auf Programmebene wird dieser Geltungsbereich erweitert um SCOPE=*TASKONLY. Während der gesamten Laufzeit des Programms kann auf taskglobale Variablen zugegriffen werden, unabhängig davon, ob sie in der umgebenden Prozedur sichtbar sind.

CLIEXPR

Der Makro CLIEXPR wertet arithmetische, logische und String-Ausdrücke aus. Der Ausdruck wird in einem Input-Feld übergeben, das Ergebnis wird in ein Output-Feld zurückgeschrieben. Es kann spezifiziert werden, wie das Ergebnis zurückgeliefert werden soll (Dualzahl, boolsche Konstante, String).

Der Makro kann auch mit MF=M aufgerufen werden. Nähere Erläuterungen zum Operanden MF=... siehe Handbuch „Makroaufrufe an den Ablaufteil“ [7].

Operation	**Operanden**
CLIEXPR	MF = E ,PARAM = <name 1..27> / (<integer 1..15>)
	MF = D [,PREFIX = C / prefix]
	MF = C ,PREFIX = C / prefix [,MACID = LIE / macid]
	MF = L ,INPUT@ = <pointer> ,INPUTL = <integer 0..2147483647> ,OUTPUT@ = <pointer> ,OUTPUTL = <integer 0..2147483647> ,VFORM = *BY-VALUE / *STRING ,OTYPE = <pointer> ,OACTL = <pointer> ,PROT@ = NULL / <pointer> ,PROTL = 0 / <integer 0..2147483647> ,OPROTL = NULL / <pointer>

Operandenbeschreibung

Die in der Beschreibung benutzte Bezeichnung <pointer> bedeutet immer eine Adressangabe in der Form **A(**symbolische Adresse**)** oder Register mit der Adresse. Diese Registerangabe ist nur in Verbindung mit MF=M möglich.

MF = E
E-Form; erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste. (Makroaufruf mit MF=L).

= <name 1..27>
Bezeichnet die symbolische Adresse der Operandenliste

= (<integer 1..15>)
Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form; es wird eine DSECT zur Operandenliste generiert. Jedes Feld hat einen Feldnamen und erläuternde Equates, falls erforderlich.

PREFIX = C / prefix
Bezeichnet das erste Zeichen der Feldnamen; Voreinstellung = C.

MF = C
C-Form; es wird nur der Datenbereich (Operandenliste) generiert. Jedes Feld hat einen Feldnamen und erläuternde Equates, falls erforderlich. Der Standardheader muss vom Anwender initialisiert werden.

PREFIX = C / prefix
Bezeichnet das erste Zeichen der Feldnamen; Voreinstellung = C.

MACID = LIE / macid
Bezeichnet das zweite, dritte und vierte Zeichen der Feldnamen; Voreinstellung = LIE.

MF = L
List-Form; es wird nur der Datenbereich (Operandenliste) generiert, unter Beachtung der im Makroaufruf angegebenen Operandenwerte. Der Datenbereich enthält keine Feldnamen und keine erläuternden Equates. Der Standardheader ist initialisiert.

INPUT@ = <pointer>
Adresse des Feldes, das den auszuwertenden Ausdruck enthält. Der Ausdruck muss als String-Ausdruck angegeben werden. Das Feld muss auf Wortgrenze ausgerichtet sein.

INPUTL = <integer 0..2147483647>
Länge des Feldes mit dem auszuwertenden Ausdruck.

OUTPUT@ = <pointer>
Adresse des Feldes, in das das Ergebnis der Auswertung geschrieben werden soll. Das Feld muss auf Wortgrenze ausgerichtet sein.

OUTPUTL = <integer 0..2147483647>
Länge des Feldes für das Ergebnis. Die (wirkliche) Länge des Ergebnisses wird in das bei OACTL=... angegebene Feld eingetragen.

VFORM =
Bezeichnet die Art, wie das Ergebnis dargestellt werden soll (Dualzahl, boolsche Konstante oder String).

VFORM = *BY-VALUE
Integer-Zahlen werden als Dualzahl (4 Byte - Zahl) dargestellt.
Boolsche Konstanten werden durch X'00' (für FALSE) oder X'01' (für TRUE) dargestellt.

VFORM = *STRING
Integer-Zahlen werden als Folge von Dezimalziffern dargestellt.
Boolsche Konstanten werden durch den String 'FALSE' oder 'TRUE' dargestellt.

PROT@ = NULL / <pointer>
Adresse des Feldes, in das SDF-P-Meldungen geschrieben werden sollen. Mehrere Meldungen werden hintereinander geschrieben. Jeder Eintrag beginnt mit einem 2-Byte-Längenfeld, gefolgt von 2 Byte mit Füllzeichen und dem anschließenden Meldungstext. Voreinstellung: Ausgabe nach SYSOUT
Hinweis
– Es werden nur Meldungen der Meldungsklasse SDP eingetragen; alle anderen Meldungen werden nach SYSOUT ausgegeben.
– Die Meldungsform (Sprache, Kurz- oder Langform, usw.) ist abhängig von den Einstellungen des Kommandos /MODIFY-MSG-ATTRIBUTES.

PROTL = 0 / <integer 0..2147483647>
Länge des Feldes für die Meldungen. Überschreitet die Meldungsausgabe die angegebene Feldlänge, wird die Meldung nicht abgeschnitten, sondern die letzte Meldung nicht eingetragen. Voreinstellung: kein Eintrag.
Die (real) benötigte Länge wird in das bei OPROTL=... angegebene Feld eingetragen.

OACTL = <pointer>
Adresse eines Feldes, in das die wirkliche Länge des Ergebnisses eingetragen wird. Das Feld muss 4 Byte lang und auf Wortgrenze ausgerichtet sein.

OTYPE = <pointer>
Adresse eines Feldes, in das der Typ des Ergebnisses eingetragen wird. Das Feld muss 1 Byte lang sein. Die Einträge beginnen mit dem bei PREFIX=.. und MACID=.. angegebenen Zeichen. Bedeutung der Einträge:

Eintrag	Bedeutung (Typ)
<prefix, macid>VSTR	String
<prefix, macid>VINT	Integerzahl
<prefix, macid>VBOO	boolsche Konstante

OPROTL = NULL / <pointer>
Adresse eines Feldes, in das die wirkliche Meldungslänge eingetragen wird. Das Feld muss 4 Byte lang sein. Voreinstellung: kein Eintrag.

Hinweise

- Die Ergebnisse sind immer einfache Werte (Basisterme). Es werden keine zusammengesetzten Ausdrücke zurückgeliefert.
- Der auszuwertende Ausdruck darf keine &-Ersetzungen enthalten.
- In das bei OTYPE=.. angegebene Feld wird immer der wirkliche Ergebnistyp eingetragen, auch wenn VFORM=*STRING angegeben wurde. Der Benutzer kann dadurch eine Ziffernfolge von einer Integerzahl oder den String „FALSE“ von dem boolschen Wert FALSE unterscheiden.

Returncodes

Die folgende Tabelle listet die Returncodes in hexadezimaler Schreibweise auf

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Normale Ausführung
01	00	0000	Überlauf: PROT-Feld (Warnung)
00	40	0001	Syntaxfehler im auszuwertenden Ausdruck
01	40	0001	Überlauf PROT-Feld
00	40	0002	Fehler beim Auswerten des Ausdrucks
01	40	0002	Überlauf: PROT-Feld
00	40	0003	OUTPUT-Feld zu klein
00	01	0004	INPUT-Feld nicht spezifiziert oder nicht ausgerichtet
01	01	0004	OUTPUT-Feld nicht spezifiziert oder nicht ausgerichtet
02	01	0004	Protokoll-Feld (nicht ausgerichtet)
03	01	0004	andere Felder (nicht ausgerichtet)
04	01	0004	Feldadresse angegeben, aber kein Zugriff möglich
00	40	0005	Kein ausreichend freier Platz im Adressraum des Aufrufers
01	20	0006	Systemfehler
00	40	0007	Ungültiges Prozedurformat; Makroausführung abgebrochen
00	01	FFFF	Falsche Angabe für UNIT oder FUNCTION im Standardheader
00	02	FFFF	Die angeforderte Funktion wird nicht unterstützt
00	03	FFFF	Falsche Versionsangabe im Standardheader

Layout der Dsect (Operandenliste)

```
   CLIEXPR MF=D,PREFIX=N
1          MFTST MF=D,PREFIX=N,MACID=LIE,ALIGN=F,
1                DMACID=LIE,SUPPORT=(E,D,C,M,L),DNAME=LIEMDL
2 NLIEMDL  DSECT ,
2                *,##### PREFIX=N, MACID=LIE #####
1 *   Which type has the output
1 NLIEVSTR EQU   1                        *STRING
1 NLIEVINT EQU   2                        *INTEGER
1 NLIEVBOO EQU   3                        *BOOLEAN
1 *
1 *   parameterarea description
1 NLIEHDR  FHDR  MF=(C,NLIE),EQUATES=NO                  Standardheader
2 NLIEHDR  DS    0A
2 NLIEFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 NLIEIFID DS    0A              0   INTERFACE IDENTIFIER
2 NLIEFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 NLIEFCT  DS    AL1             2   FUNCTION NUMBER
2 NLIEFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 NLIERET  DS    0A              4   GENERAL RETURN CODE
2 NLIESRET DS    0AL2            4   SUB RETURN CODE
2 NLIESR2  DS    AL1             4   SUB RETURN CODE 2
2 NLIESR1  DS    AL1             5   SUB RETURN CODE 1
2 NLIEMRET DS    0AL2            6   MAIN RETURN CODE
2 NLIEMR2  DS    AL1             6   MAIN RETURN CODE 2
2 NLIEMR1  DS    AL1             7   MAIN RETURN CODE 1
2 NLIEFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *   main return codes
1 NLIESUCC EQU   0                        No error detected
1 NLIESYNT EQU   1                        Syntax error
1 NLIEEVAL EQU   2                        Semantic error
1 NLIETRUN EQU   3                        Output buffer too small
1 NLIEAREA EQU   4                        Buffer missing or not aligned
1 *                                       or not accessible
1 NLIEREQM EQU   5                        Out of memory
1 NLIEDUMP EQU   6                        Invalid SDF-P-BASYS
1 *                                       processing
1 NLIECTXT EQU   7                        Old procedure context
1 *
1 NLIEIPTR DS    A                        SDF-P expression
```

```
 1 NLIEOPTR DS    A                        Resulting Value
 1 NLIEPPTR DS    A                        Resulting Protocol
 1 NLIEILEN DS    F                        SDF-P expression
 1 NLIEOMAX DS    F                        Value attribute (maximum
 1 *                                       length)
 1 NLIEPMAX DS    F                        Protocol attribute (maximum
 1 *                                       length)
 1 NLIEFORM DS    FL1                      Value attribute ( string
 1 *                                       generation )
 1 *   Wished output form
 1 NLIEFVAL EQU   0                        *BY-VALUE
 1 NLIEFSTR EQU   1                        *STRING
 1 *
 1 NLIERES1 DS    CL7                      Alignement
 1 NLIEOLEN DS    A                        Value length as FW-aligned 4
 1 *                                       bytes field
 1 NLIEOTYP DS    A                        Value type as 1 byte field
 1 NLIEPLEN DS    A                        Protocol length as FW-aligned
 1 *                                       4 bytes field
 1 NLIE#    EQU   *-NLIEHDR
```

Beispiel

```
CLIEXPR  START
         BALR  3,0
         USING *,3
         CLIEXPR MF=E,PARAM=OPLISTE
WROUT    WROUT  AUSG,TERM,PARMOD=31
TERM     TERM
*****    DEFINITIONEN   *****
OPLISTE  CLIEXPR MF=L,INPUT@=A(IF),INPUTL=10,OUTPUT@=A(OF),OUTPUTL=10,V-
               FORM=*BY-VALUE,OACTL=A(H1),OTYPE=A(H2)
         DS    OF
IF       DC    CL10'(8+3)'
         DS    OF
AUSG     DC    Y(AUSGE-AUSG)
         DS    3X
         DC    C'AUSGABE:  '
OF       DS    cl10
AUSGE    EQU   *
         DS    CL10
         DS    OF
H1       DS    CL4
         DS    OF
H2       DS    CL1
         END
```

CLIGET

Mit dem Makroaufruf CLIGET kann ein Programm abfragen, ob ein Schutz vor impliziten Unterbrechungen erforderlich ist.
CLIGET liefert die Einstellung im Operanden INTERRUPT-ALLOWED, die in den Kommandos SET-PROCEDURE-OPTIONS, MODIFY-PROCEDURE-OPTIONS oder BEGIN-PROCEDURE gesetzt wurde.
Zu weiteren Informationen siehe Abschnitt „Nichtunterbrechbarkeit“ auf Seite 129.

Operation	Operanden
CLIGET	MF = E ,PARAM = <name 1..8> / (integer 1..15)
	MF = D [,PREFIX = C / prefix]
	MF = C [,PREFIX = C / prefix] [,MACID = LIS / macid]
	MF = L

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs: erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form des Makroaufrufs: erzeugt eine DSECT für die Operandenliste. Die generierten Namen beginnen mit dem Buchstaben C; sie können mit PREFIX verändert werden.

PREFIX = C / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben C. Dieser Buchstabe kann durch den Parameter prefix geändert werden.

MF = C
C-Form des Makroaufrufs: erzeugt eine Operandenliste, deren symbolische Namen mit der Zeichenfolge CLIG beginnen. Sie können durch PREFIX und MACID verändert werden.

PREFIX = C / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben C. Dieser Buchstabe kann durch den Parameter prefix geändert werden.

MACID = LIG / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: LIG

MF = L
LIST-Form des Makroaufrufs: erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form); der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

Ausgabe-Parameter

Die Ausgabe-Parameter werden in die vorgegebenen Felder in der Operandenliste zurückgegeben. Das Aufruferprogramm muss die Operandenliste bei einer Ersetzung mit korrespondierenden Namen lesen.

&P.INTA=INTERRUPT-ALLOWED
Ein Byte langes Feld, in das die Prozeduroption „INTERRUPT-ALLOWED“ durch das Makro zurückgegeben wird.
&P.INTN bedeutet NO bzw. INTERRUPT-ALLOWED = *NO: Das Programm muss vor einer impliziten Unterbrechung geschützt werden.
&P.INTY bedeutet YES bzw. INTERRUPT-ALLOWED = *YES: Das Programm muss vor einer impliziten Unterbrechung nicht geschützt werden.

Hinweis

Bei &P.INTN wird

- K2 zurückgewiesen, wenn nicht K2-STXIT aktiviert ist.
- //EXECUTE-SYSTEM-CMD und //HOLD-PROGRAM zurückgewiesen, wenn SYSSTMT ungleich SYSCMD ist (d.h. entweder Datensichtstation oder Datei oder andere Zuweisung).

Jede andere sicherheitsrelevante Aktion (CMD, BKPT, K2-STXIT, SVC usw.) ist unter der Verantwortung des Aufruferprogramms. D.h. das Programm darf diese Makros nicht aktivieren und Unterbrechungen verursachen, wenn diese Aktionen vom Endbenutzer verlangt werden.

Returncodes

Die Tabelle auf der folgenden Seite listet die Returncodes in hexadezimaler Schreibweise auf. Register von Benutzerprogrammen werden nicht verändert.

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Makroaufruf war erfolgreich; kein Fehler
00	01	0001	Parameter-Fehler; Parameter zu kurz
00	20	0004	Systemfehler
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste

CLISET

Der Makroaufruf CLISET schützt Programme explizit vor Unterbrechungen.
Zu weiteren Informationen Abschnitt „Nichtunterbrechbarkeit“ auf Seite 129.

Operation	Operanden
CLISET	MF = E
	,PARAM = <name 1..8> / (integer 1..15)
	MF = D
	[,PREFIX = C / prefix]
	MF = C
	[,PREFIX = C / prefix]
	[,MACID = LIS / macid]
	MF = L
	[,EXNINT = *U / *Y/ *N]

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs; erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form des Makroaufrufs; erzeugt eine DSECT für die Operandenliste. Die generierten Namen beginnen mit dem Buchstaben C; sie können mit PREFIX verändert werden.

PREFIX = C / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben C. Dieser Buchstabe kann durch den Parameter prefix geändert werden.

MF = C
C-Form des Makroaufrufs; erzeugt eine Operandenliste, deren symbolische Namen mit der Zeichenfolge CLIS beginnen. Sie können durch PREFIX und MACID verändert werden.

PREFIX = C / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben C. Dieser Buchstabe kann durch den Parameter prefix geändert werden.

MACID = LIS / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: LIS

MF = L
LIST-Form des Makroaufrufs: erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form). Der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

EXNINT
Mit dieser Option wird für ein Programm der explizite Nichtunterbrechungs-Modus eingestellt.

= *U
Die vorangegangene Einstellung wird nicht verändert. Beim ersten Aufruf wird hier *N angenommen.

= *Y
Das Programm ist explizit vor Unterbrechungen geschützt.

= *N
Das Programm ist nicht explizit vor Unterbrechungen geschützt.

Hinweise

- Die Option EXNINT kann nicht gestapelt werden, wenn CLISET mehrere Male aufgerufen wird.
- EXNINT =*U ist eine Dummy-Operandeneinstellung. Es setzt keine Informationen in SDF-P ab und löst auch keinen Returncode mit Ausnahme von „Kein Fehler“ aus.
- EXNINT=*Y bewirkt folgende Systemänderungen:
  - K2 zurückgewiesen, wenn nicht K2-STXIT aktiviert ist.
  - //EXECUTE-SYSTEM-CMD wird zurückgewiesen.
  - Durch die Anweisung und das Kommando HOLD-PROGRAM wird EOF zurückgegeben.
- Jede andere sicherheitsrelevante Aktion (CMD, BKPT, K2-STXIT, SVC usw.) ist unter der Verantwortung des Aufrufer-Programms.

Returncodes

Die Tabelle auf der folgenden Seite listet die Returncodes in hexadezimaler Schreibweise auf. Register von Benutzerprogrammen werden nicht verändert.

Subcode2	Subcode1	Maincode	Bedeutung
01	00	0000	Keine Aktion (EXNINT=*U)
00	00	0000	Makroaufruf war erfolgreich; kein Fehler
00	01	0001	Parameter-Fehler;
01	00	0002	EXNINT=*Y wurde schon vorher gesetzt oder EXNINT=*N wurde schon vorher gesetzt
00	20	0004	Systemfehler
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste

CMD

Mit dem Makro CMD können in Assembler-Programmen Kommandos ausgeführt werden, auch SDF-P-Kommandos. Der Makro CMD ist im Handbuch „Makroaufrufe an den Ablaufteil“ [7] beschrieben.

Kommando	Funktion
ASSIGN-STREAM	S-Variablenstrom zuweisen
BEGIN-STRUCTURE	Deklaration einer statischen Struktur beginnen
CALL-PROCEDURE	Prozedur starten
CLOSE-VARIABLE-CONTAINER	Variablenbehälter schließen
DECLARE-CONSTANT	Variable mit konstantem Wert definieren
DECLARE-ELEMENT	Strukturelement deklarieren
DECLARE-VARIABLE	Variable deklarieren
DELETE-STREAM	S-Variablenstrom löschen
DELETE-VARIABLE	Variable löschen
END-STRUCTURE	Ende einer Strukturdeklaration kennzeichnen
ENTER-PROCEDURE	Prozedur als Hintergrundprozedur starten
FREE-VARIABLE	Variableninhalt löschen
IMPORT-VARIABLE	Variable importieren
INCLUDE-CMD	Kommandofolgen übergeben
INCLUDE-PROCEDURE	INCLUDE-Prozedur starten
MODIFY-PROCEDURE-OPTIONS	Prozedureigenschaften ändern
OPEN-VARIABLE-CONTAINER	Variablenbehälter öffnen
READ-VARIABLE	Variablenwerte einlesen
SAVE-VARIABLE-CONTAINER	Variablenbehälter sichern
SELECT-VARIABLE-ELEMENTS	Elemente einer Listenvariablen auswählen
SHOW-STREAM-ASSIGNMENT	Zuweisung für S-Variablenstrom ausgeben
SHOW-STRUCTURE-LAYOUT	Elementnamen eines Strukturlayouts ausgeben
SHOW-VARIABLE	Variableninhalte ausgeben
SHOW-VARIABLE-ATTRIBUTES	Variablenattribute ausgeben
SHOW-VARIABLE-CONTAINER-ATTR	Variablenbehälter-Attribute ausgeben
SORT-VARIABLE	Elemente einer Listenvariablen sortieren
TRANSMIT-BY-STREAM	Variablen mit S-Variablenstrom übertragen

GETVAR

Der Makroaufruf GETVAR liest den Inhalt einer Variablen.
GETVAR kann auf einfache Variablen und Elemente zusammengesetzter Variablen angewendet werden.

Operation	Operanden
GETVAR	MF = E ,PARAM = <name 1..8> / (<integer 1..15>)
	MF = D ,PREFIX = G / prefix
	MF = C ,PREFIX = G / prefix ,MACID = ETV / macid
	MF = L ,NAMLEN = <integer 1..255> ,NAMADR = <name 1..8> ,SCOPE = *VISIBLE / *TASKONLY ,MAXLEN = <integer 1..4096> ,VALADR = <name 1..8>

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs: erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form des Makroaufrufs; erzeugt eine DSECT für die Operandenliste. Die generierten Namen beginnen mit dem Buchstaben G; sie können mit PREFIX verändert werden.

PREFIX = G / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben G.

MF = C
C-Form des Makroaufrufs; erzeugt eine Operandenliste, deren symbolische Namen mit der Zeichenfolge GETV beginnen. Sie können durch PREFIX und MACID verändert werden.

PREFIX = G / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben G.

MACID = ETV / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: ETV

MF = L
LIST-Form des Makroaufrufs: erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form); der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

NAMLEN = <integer 1..255>
Bezeichnet die Länge des Variablennamens.

NAMADR = <name 1..8>
Bezeichnet den symbolischen Namen der Adresse des Variablennamens.

SCOPE
Bezeichnet den Geltungsbereich der Variablen.

= *VISIBLE
Die Variable ist eine prozedurlokale Variable.

= *TASKONLY
Die Variable ist eine taskglobale Variable.

MAXLEN = <integer 1..4096>
Bezeichnet die Länge des Variablenwerts.

VALADR = <name 1..8>
Bezeichnet die symbolische Adresse des Variablenwerts.

Returncodes

Die folgende Tabelle listet die Returncodes in hexadezimaler Schreibweise auf.

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Makroaufruf war erfolgreich; kein Fehler
00	01	0001	Parameter-Fehler
00	01	0002	Syntaxfehler im Variablennamen
00	40	0003	Area zu klein
00	40	0004	Variable nicht deklariert
00	40	0005	Variablenbehälter nicht verfügbar
00	40	0006	Datentyp und Variablenwert nicht vereinbar
00	40	0008	Variable hat keinen Wert
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste

PUTVAR

Der Makroaufruf PUTVAR weist einer Variablen einen Wert zu. PUTVAR kann auf einfache Variablen und Elemente zusammengesetzter Variablen angewendet werden.

Wenn eine einfache Variable bei der Zuweisung noch nicht existiert, dann wird sie angelegt, falls IMPLICIT-DECLARATION=YES und IMPDEC = *STD gilt oder im Makroaufruf IMPDEC=*YES angegeben ist.

Zusammengesetzten Variablen kann ein Wert zugewiesen werden, wenn sie vom Typ Integer, Boolean, String oder Any sind.

Operation	Operanden
PUTVAR	MF = E ,PARAM = <name 1..8> / (<integer 1..15>)
	MF = D ,PREFIX = P / prefix
	MF = C ,PREFIX = P / prefix ,MACID = UTV / macid
	MF = L ,NAMLEN = <integer 1..255> ,NAMADR = <name 1..8> ,SCOPE = *VISIBLE / *TASKONLY ,IMPDEC = *YES / *NO / *STD ,VALLEN = <integer 0..4096> ,VALADR = <name 1..8> ,VALTYPE = *INTEGER / *BOOLEAN / *STRING

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs; erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form des Makroaufrufs: erzeugt eine DSECT für die Operandenliste. Die generierten Namen beginnen mit dem Buchstaben P; sie können mit PREFIX verändert werden.

PREFIX = P / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben P.

MF = C
C-Form des Makroaufrufs: erzeugt eine Operandenliste, deren symbolische Namen mit der Zeichenfolge PUTV beginnen. Sie können durch PREFIX und MACID verändert werden.

PREFIX = P / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben P.

MACID = UTV / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: UTV

MF = L
LIST-Form des Makroaufrufs: erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form). Der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

NAMLEN = <integer 1..255>
Bezeichnet die Länge des Variablennamens.

NAMADR = <name 1..8>
Bezeichnet den symbolischen Namen der Adresse des Variablennamens.

SCOPE
Definiert den Geltungsbereich der Variablen.

= *VISIBLE
Die Variable wird als prozedurlokale Variable angelegt.

= *TASKONLY
Die Variable wird als taskglobale Variable angelegt.

IMPDEC = *YES / *NO / *STD
Legt fest, ob die Variable implizit angelegt wird, wenn sie noch nicht vorhanden ist, unabhängig von der Einstellung in der umgebenden Prozedur.

= *STD
Gibt an, dass die Attribute von IMPLICIT-DECLARATION für die gegenwärtige Prozedur verwendet werden.

VALLEN = <integer 0..4096>
Bezeichnet die Länge des Variablenwerts.

VALADR = <name 1..8>
Bezeichnet die symbolische Adresse des Variablenwerts.

VALTYPE
Legt den Datentyp der Variablen fest.

= *INTEGER
Die Variable erhält den Datentyp INTEGER; Zuweisung anderer als Integer-Werte führt zum Fehler.

= *BOOLEAN
Die Variable erhält den Datentyp BOOLEAN; Zuweisung anderer Werte als TRUE oder FALSE führt zum Fehler.

= *STRING
Die Variable erhält den Datentyp STRING.
Die Maximallänge des Strings wird mit VALLEN-LENGTH definiert.

Returncodes

Die folgende Tabelle listet die Returncodes in hexadezimaler Schreibweise auf.

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Makroaufruf war erfolgreich; kein Fehler
00	01	0001	Parameter-Fehler
00	01	0002	Syntaxfehler im Variablennamen
00	40	0004	Variable nicht deklariert
00	40	0005	Variablenbehälter nicht verfügbar
00	40	0006	Datentyp und Variablenwert nicht vereinbar
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste

SHOWSSA

Der Makroaufruf SHOWSSA zeigt die aktuelle Zuweisung des angegebenen S-Variablenstroms an. SHOWSSA ist funktionsgleich mit dem Kommando SHOW-STREAM-ASSIGNMENT. Jedoch kann mit SHOWSSA keine Liste von Strömen angegeben werden.

Operation	Operanden
SHOWSSA	MF = E ,PARAM = <name 1..8> / (<integer 1..15>)
	MF = D ,PREFIX = S / prefix
	MF = C ,PREFIX = S / prefix ,MACID = HOW / macid
	MF = L / M ,STREAM = *ALL / *STD_STREAMS / <c-string 1..20 with-wild> ,INFO = *ASSIGNMENT / *DESTINATION ,OUTPUT = *RETURN_CODE / *SYSOUT / *SYSLST

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs; erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form des Makroaufrufs; erzeugt eine DSECT für die Operandenliste. Die generierten Namen beginnen mit dem Buchstaben S; sie können mit PREFIX verändert werden.

PREFIX = S / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben S.

MF = C
C-Form des Makroaufrufs; erzeugt eine Operandenliste, deren symbolische Namen mit der Zeichenfolge SHOW beginnen. Sie können durch PREFIX und MACID verändert werden.

PREFIX = S / <name 1..1
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben S.

MACID = HOW / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Voreinstellung: HOW.

MF = L / M
LIST-Form des Makroaufrufs; erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form); der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

STREAM
Name des S-Variablenstroms, der ausgegeben werden soll. Es werden dabei keine Listen unterstützt.

= *ALL
Alle S-Variablenströme, die in der aktuellen Prozedur sichtbar sind, werden aufgelistet.

= *STD_STREAMS
Alle S-Variablenströme, die standardmäßig im System implementiert sind, werden angezeigt. Die Namen dieser Ströme beginnen mit dem Präfix „SYS“. Es werden die Namen aufgelistet, die in der Werteliste der Syntaxbeschreibung des Operanden STREAM-NAME des Kommandos ASSIGN-STREAM stehen.

= <c-string 1..20 with-wild>
S-Variablenstrom, der angezeigt werden sollen. Bei Verwendung von Musterzeichen werden alle S-Variablenströme angezeigt, die dieses Suchmuster erfüllen.

INFORMATION
Gibt an, welche Information ausgegeben werden muss.

= *ASSIGNMENT
Der „TO“-Wert des Kommandos ASSIGN-STREAM muss ausgegeben werden.
Wenn dieser Stream-Name einem anderen Stream-Namen zugewiesen ist, wird dieser Name ausgegeben.

= *DESTINATION
Es wird der Name des aktuellen Servers, der mit dem S-Variablenstrom verbunden ist, ausgegeben.
Wenn der Variablenstrom einem anderen Variablenstrom-Namen zugewiesen ist, wird die letzte Zuweisung ausgegeben.

OUTPUT
Gibt an, wohin die Information ausgegeben werden soll. Die Ausgabe in eine S-Variable ist nicht möglich.

= RETURN_CODE
Es wird nur ein Returncode zurückgeliefert; Voreinstellung. Die Angabe ist nur sinnvoll, wenn STREAM=<c-string 1..20> (d.h. ein Name ohne Musterzeichen) angegeben wurde.

= *SYSOUT
Ausgabe nach SYSOUT.

= *SYSLST
Ausgabe nach SYSLST.

Returncodes

Die folgende Tabelle listet die Returncodes in hexadezimaler Schreibweise auf.

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Makroaufruf war erfolgreich; kein Fehler
00	01	0001	Parameter-Fehler
00	40	0002	Angegebener Variablenstrom unvollständig
00	40	0003	SSTA-Fehler (SSTA zu klein, nicht initialisiert, ...)
00	40	0004	Abgebrochen durch K2 während Ausgabe auf SYSOUT
00	40	0005	Fehler während Ausgabe auf SYSOUT
00	40	0006	Fehler während Ausgabe auf SYSLST
02	01	0007	Mehr als ein Variablenstrom für OUTPUT= *RETURNCODE
00	20	0008	System-Fehler
02	00	000A	Angegebener Variablenstrom ist *DUMMY zugewiesen
02	00	000B	Angegebener Variablenstrom ist schon zugewiesen
02	00	000C	Angegebener Variablenstrom existiert nicht
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste
00	41	FFFF	SDF-P ist nicht geladen
00	81	FFFF	SDF-P arbeitet zwischenzeitlich nicht

TRANSVV

Der Makroaufruf TRANSVV führt von Client-Seite eine Variablenübertragung über den angegebenen S-Variablenstrom durch, zu dem aktuell zugewiesenen Server (Kommando ASSIGN-STREAM). TRANSVV ist funktionsgleich mit dem Kommando TRANSMIT-BY-STREAM. Es können nur S-Variablenströme angegeben werden, die auf der gleichen Prozedurstufe zugewiesen wurden, auf der das Programm gestartet wurde.

Operation	**Operanden**
TRANSVV	MF = E ,PARAM = <name 1..8> / (<integer 1..15>)
	MF = D [,PREFIX = T / prefix]
	MF = C [,PREFIX = T / prefix] [,MACID = RAN / macid]
	MF = L/M ,STREAM = <name 1..20> [,VNAME = *NONE / <name 1..8> / (integer 1..15)] [,VNAMEL = <integer 1..255>] [,VSCOPE = *VISIBLE / *TASKONLY] [,RNAME = *SAME / *NONE / <name 1..8> / (integer 1..15)] [,RNAMEL = <integer 1..255>] [,RSCOPE = *VISIBLE / *TASKONLY] [,CNAME = *NONE / <name 1..8> / (integer 1..15)] [,CNAMEL = <integer 1..255>] [,CSCOPE = *VISIBLE / *TASKONLY] [,RCNAME = *SAME / *NONE / <name 1..8> / (integer 1..15)] [,RCNAMEL = <integer 1..255>] [,RCSCOPE = *VISIBLE / *TASKONLY]

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs; erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form; es wird eine DSECT zur Operandenliste generiert. Jedes Feld hat einen Feldnamen und erläuternde Equates, falls erforderlich.

PREFIX = T / prefix
Bezeichnet das erste Zeichen der Feldnamen; Voreinstellung = T.

MF = C
C-Form; es wird nur der Datenbereich (Operandenliste) generiert. Jedes Feld hat einen Feldnamen und erläuternde Equates, falls erforderlich. Der Standardheader muss vom Anwender initialisiert werden.

PREFIX = T / prefix
Bezeichnet das erste Zeichen der Feldnamen; Voreinstellung = T.

MACID = RAN / macid
Bezeichnet das zweite, dritte und vierte Zeichen der Feldnamen; Voreinstellung = RAN.

MF = L / M
LIST-Form des Makroaufrufs; erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form); der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

STREAM = <name 1..20>
Name des S-Variablenstroms, mit dem die Variable übertragen werden soll.

VNAME
Name der S-Variablen, die zum Server übertragen werden soll.

= *NONE
Es wird keine S-Variable übertragen.

= <name 1..8>
Bezeichnet die symbolische Adresse des Feldes, das den Namen der S-Variable enthält.

= (<integer 1..15>)
Bezeichnet das Register mit der Adresse des Feldes, das den Namen der S-Variable enthält (Registernummer muss in Klammern eingeschlossen werden).

VNAMEL = <integer 1..255>
Bezeichnet die Länge des Variablennamens, der vom Aufrufer angegeben wurde.

VSCOPE
Definiert den Geltungsbereich der Variablen.

= *VISIBLE
Die Variable wird als prozedurlokale Variable angelegt.

= *TASKONLY
Die Variable wird als taskglobale Variable angelegt.

Es sind folgende Kombinationen erlaubt:

```
VNAME=*NONE
VSCOPE=*VISIBLE
```

oder:

```
VSCOPE=*VISIBLE / *TASKONLY,
VNAME =<name 1..8> / (integer 1..15),
VNAMEL=<integer 1..255>
```

RNAME
Bei der Übertragung zurückgesendete S-Variable bzw. Return-Variable.

= *SAME
Die Werte von VNAME, VNAMEL und VSCOPE werden beibehalten.

= *NONE
Es wird keine Return-Variable zurückgesendet.

= <name 1..8>
Bezeichnet die symbolische Adresse des Feldes, das den Namen der Return-Variable enthält.

= (<integer 1..15>)
Bezeichnet das Register mit der Adresse des Feldes, das den Namen der Return-Variable enthält (Registernummer muss in Klammern eingeschlossen werden).

RNAMEL = <integer 1..255>
Bezeichnet die Länge des Variablennamens, der vom Aufrufer angegeben wurde.

RSCOPE
Definiert den Pool oder den Behälter der Return-Variablen.

= *VISIBLE
Die Return-Variable wird als prozedurlokale Variable angelegt.

= *TASKONLY
Die Return-Variable wird als taskglobale Variable angelegt.

Es sind folgende Kombinationen erlaubt:

```
RNAME=*NONE
```

oder:

```
RSCOPE=*VISIBLE / *TASKONLY,
RNAME =<name 1..8> / (integer 1..15),
RNAMEL=<integer 1..255>
```

CNAME
Bei der Übertragung gesendete Kontroll-Variable.

= *NONE
Es wird keine Kontroll-Variable übertragen. Diese kann sowohl wiederum als Variable als auch als Registernummer, die in Klammern eingeschlossen werden muss, angegeben werden.

= <name 1..8>
Bezeichnet die symbolische Adresse des Feldes, das den Namen der Kontroll-Variable enthält.

= (<integer 1..15>)
Bezeichnet das Register mit der Adresse des Feldes, das den Namen der Kontroll-Variable enthält (Registernummer muss in Klammern eingeschlossen werden).

CNAMEL = <integer 1..255>
Bezeichnet die Länge des Kontroll-Variablennamens, der vom Aufrufer angegeben wurde.

CSCOPE
Definiert den Pool oder den Behälter der Kontroll-Variablen.

= *VISIBLE
Die Kontroll-Variable wird als prozedurlokale Variable angelegt.

= *TASKONLY
Die Kontroll-Variable wird als taskglobale Variable angelegt.

Es sind folgende Kombinationen erlaubt:

```
CNAME=*NONE
CSCOPE=*VISIBLE / *TASKONLY
CNAME =<name 1..8> / (integer 1..15), CNAMEL=<integer 1..255>
```

RCNAME
Bei der Übertragung zurückgesendete Kontroll-Variable bzw. Return-Kontroll-Variable.

= *SAME
Es werden die Werte von CNAME, CNAMEL und CSCOPE beibehalten.

= *NONE
Es wird keine Return-Kontroll-Variable übertragen.

= <name 1..8>
Bezeichnet die symbolische Adresse des Feldes, das den Namen der Return-Kontroll-Variable enthält.

= (<integer 1..15>)
Bezeichnet das Register mit der Adresse des Feldes, das den Namen der Return-Kontroll-Variable enthält (Registernummer muss in Klammern eingeschlossen werden).

RCNAMEL = <integer 1..255>
Bezeichnet die Länge des Return-Kontroll-Variablennamens, der vom Aufrufer angegeben wurde.

RCSCOPE
Definiert den Pool oder den Behälter der Return-Kontroll-Variablen.

= *VISIBLE
Die Return-Kontroll-Variable wird als prozedurlokale Variable angelegt.

= *TASKONLY
Die Return-Kontroll-Variable wird als taskglobale Variable angelegt.

Es sind folgende Kombinationen erlaubt:

```
RCNAME=*NONE
RCSCOPE=*VISIBLE / *TASKONLY
RCNAME =<name 1..8> / (integer 1..15), RCNAMEL=<integer 1..255>
```

Returncodes

Die folgende Tabelle listet die Returncodes in hexadezimaler Schreibweise auf.

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Übertragung erfolgreich beendet; kein Fehler
01	00	0000	Variablenstrom wurde *DUMMY zugewiesen; keine übertragung
00	01	0001	Parameter-Fehler
00	40	0002	Angegebener Variablenstrom unvollständig
00	40	0003	Angegebene Variable unvollständig
00	40	0004	RET-SSTA zu klein (reserviert für Software-Entwickler)
00	01	0005	Übertragene Daten (Benutzer- oder Kontrolldaten) sind nicht mit dem Format, das der Server verarbeiten kann, kompatibel
00	40	0006	Fehlermeldung vom Server; gespeichert in RCNAME (wenn angegeben)
02	00	0007	Warnung vom Server; gespeichert in RCNAME (wenn angegeben)
02	00	0008	Variablenstrom auf *DUMMY zurückgesetzt; Server ist nicht mehr aktiv
00	20	0009	System-Fehler
00	20	000A	Fehler bei der Server-Verbindung
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste
00	41	FFFF	SDF-P ist nicht geladen
00	81	FFFF	SDF-P arbeitet zwischenzeitlich nicht

VARINF

Mit dem Makroaufruf VARINF können Variablen analysiert werden, auch zusammengesetzte Variablen, deren Elemente selbst wieder zusammengesetzte Variablen sind.

Operation	Operanden
VARINF	MF = E ,PARAM = <name 1..8> / (<integer 1..15>)
	MF = D ,PREFIX = V / prefix
	MF = C ,PREFIX = V / prefix ,MACID = ARI / macid
	MF = L ,NAMLEN = <integer 1..255> ,NAMADR = <name 1..8> ,SCOPE = *VISIBLE / *TASKONLY ,POSIT = *CURRENT / *UP / *DOWN / *NEXT ,MAXLEN = <integer 1..4096> ,RESADR = <name 1..8>

Operandenbeschreibung

MF = E
Execute-Form des Makroaufrufs; erzeugt einen SVC.

PARAM
Bezeichnet die Adresse der Operandenliste, die für den Makroaufruf ausgewertet wird (Adresse des Makroaufrufs mit MF=L).

= <name 1..8>
Bezeichnet die symbolische Adresse der Operandenliste.

= (<integer 1..15>)
Bezeichnet das Register, das die Adresse der Operandenliste enthält.

MF = D
DSECT-Form des Makroaufrufs: erzeugt eine DSECT für die Operandenliste. Die generierten Namen beginnen mit der Zeichenfolge VARINF; sie können mit PREFIX verändert werden.

PREFIX = V / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben V.

MF = C
C-Form des Makroaufrufs: erzeugt eine Operandenliste, deren symbolische Namen mit der Zeichenfolge VARI beginnen. Sie können durch PREFIX und MACID verändert werden.

PREFIX = V / prefix
Definiert das erste Zeichen der generierten Namen.
Voreinstellung: Die generierten Namen beginnen mit dem Buchstaben V.

MACID = ARI / macid
Bis zu drei Zeichen langer String, der die Zeichen 2 bis 4 der generierten Namen ersetzt. Standard: ARI

MF = L
LIST-Form des Makroaufrufs; erzeugt die Operandenliste für den Makroaufruf mit MF=E (Execute-Form); der Makroaufruf muss über eine symbolische Adresse adressierbar sein.

NAMLEN = <integer 1..255>
Bezeichnet die Länge des Variablennamens.

NAMADR = <name 1..8>
Bezeichnet den symbolischen Namen der Adresse des Variablennamens, von dem ausgehend die Elementnamen abgefragt werden sollen.

SCOPE
Bezeichnet den Geltungsbereich der Variablen.

= *VISIBLE
Die Variable ist eine prozedurlokale Variable.

= *TASKONLY
Die Variable ist eine taskglobale Variable.

POSIT
Bestimmt das Variablenelement, dessen Name zurückgeliefert werden soll. Es erfolgt keine absolute, sondern nur eine relative Positionierung, jeweils ausgehend von dem Variablenelement, auf das zuletzt zugegriffen wurde.

= *CURRENT
Liefert den Namen des aktuellen Variablenelements, das heißt des Variablenelements, das als Ausgangspunkt für die Positionierung dient (Existenzprüfung).

= *UP
Liefert den Namen der zusammengesetzten Variablen, zu der das aktuelle Variablenelement gehört.

= *DOWN
Wenn das aktuelle Variablenelement selbst eine zusammengesetzte Variable ist, liefert POSITION=*DOWN den Namen des ersten Elements dieser zusammengesetzten Variablen.

= *NEXT
Liefert den Namen der nächsten zusammengesetzten Variablen auf der gleichen Ebene.

MAXLEN = <integer 1..4096>
Bezeichnet die maximale Länge des Feldes, in dem der Variablenname zurückgeliefert wird.

RESADR = <name 1..8>
Symbolische Adresse des Feldes, in dem der Variablenname zurückgeliefert wird.

Die folgenden Ausgabefelder sind in der Operandenliste nach dem Aufruf zu finden:

```
<PR> = <prefix><macid>

<PR>VALL : Länge des Ergebnisnamens

<PR>VTYP : Variablentyp:
        Mögliche Werte: <PR>VANY: *ANY
                        <PR>VSTR: *STRING
                        <PR>VINT: *INTEGER
                        <PR>VBOO: *BOOLEAN
                        <PR>VSTU: *STRUCTURE

<PR>MULT : MULTIPLE-ELEMENTS:
        Mögliche Werte: <PR>MNO: *NONE
                        <PR>MARR: *ARRAY
                        <PR>MLIS: *LIST

<PR>SINF : STRUCTURE INFORMATION (relevant wenn: <PR>VTYP=<PR>VSTU)
        Mögliche Werte: <PR>SDYN: *DYNAMIC
                        <PR>SCMD: *BY-SYSCMD
                        <PR>SLAY: LAYOUT
```

Returncodes

Die folgende Tabelle listet die Returncodes in hexadezimaler Schreibweise auf.

Subcode2	Subcode1	Maincode	Bedeutung
00	00	0000	Makroaufruf war erfolgreich; kein Fehler
00	01	0001	Parameter-Fehler
00	01	0002	Syntaxfehler im Variablennamen
00	40	0003	Area zu klein
00	40	0004	Variable nicht deklariert
00	40	0005	Variablenbehälter nicht verfügbar
00	40	0007	Letztes Variablenelement erreicht, kein weiteres Variablenelement vorhanden
00	01	FFFF	Unbekannte Unit- oder Funktions-Nummer
00	02	FFFF	Funktion nicht verfügbar
00	03	FFFF	Falsche Version der Operandenliste

14 Vordefinierte Funktionen

In diesem Kapitel sind die zum Lieferumfang vom SDF-P gehörenden vordefinierten Funktionen detailliert in alphabetischer Reihenfolge beschrieben.

Jeder Eintrag ist folgendermaßen aufgebaut:

- Funktionsname mit (deutschem) Kurztitel
- Zuordnung zu Anwendungsgebieten
- Beschreibung der Funktion
- Formatdarstellung für den Funktionsaufruf
- Ergebnistyp
- Beschreibung der Eingabeparameter
- Beschreibung der Ergebniswerte
- Fehlermeldungen
- Beispiele

In einigen Fällen werden den Eingabeparametern Schlüsselwörter als Werte zugewiesen; die Bedeutung dieser Schlüsselwörter ist bei der Beschreibung der Eingabeparameter erklärt.
Bei den meisten Eingabeparametern kann der Benutzer jedoch den aktuellen Wert frei definieren; es werden folgende Bezeichnungen verwendet:

- string_ausdruck: Der Programmierer kann direkt einen String angeben ('string'), den Namen einer Variablen, die einen String enthält (varname), oder einen Ausdruck, der als Ergebnis einen String liefert.
- zeichen: Wie string_ausdruck, der einzusetzende String besteht jedoch nur aus einem Zeichen.
- arithm_ausdruck: Der Benutzer kann direkt einen Integer-Wert angeben (zahl), den Namen einer Variablen, die einen Integer-Wert enthält (varname), oder einen Ausdruck, der als Ergebnis einen Integer-Wert liefert.
- kurzname( ): Funktionsnamen können nicht abgekürzt werden. Einige Funktionen können außer über den Funktionsnamen auch über einen Kurznamen aufgerufen werden. Dieser Kurzname steht direkt unter dem Funktionsnamen.

Hinweis

Folgende Syntax-Fehlermeldungen sind bei jeder vordefinierten Funktion möglich und werden nicht bei den einzelnen Funktionen nochmals aufgeführt: SDP0005, SDP0006, SDP0008, SDP0009, SDP0010, SDP0039, SDP0099, SDP0300, SDP0304, SDP0306, SDP0402, SDP0431, SDP0444.

ACCOUNT( ) Abrechnungsnummer abfragen

Anwendungsgebiet: **taskspezifische Umgebungsinformationen**

Die Funktion ACCOUNT( ) ermittelt die Abrechnungsnummer der aktuellen Task, die im SET-LOGON-PARAMETERS-Kommando angegeben wurde.

Format

ACCOUNT( )

Ergebnistyp

STRING (<string 1 .. 8>)

Eingabeparameter

Keine

Ergebnis

Maximal achtstellige Abrechnungsnummer

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = ACCOUNT
/SHOW-VARIABLE A

A = K27DKU
```

ARRAY-INDEX( ) Arrayindex abfragen

Anwendungsgebiet: **Variablenzugriff (Variablenname)**

Die Funktion ARRAY-INDEX( ) lässt sich auf Arrays anwenden. ARRAY-INDEX liefert den Wert eines Arrayindex. Dadurch können andere Funktionen anschließend über den Arrayindex explizit auf dieses Element zugreifen.

Format

ARRAY-INDEX( )
ARRAY-NAME = string_ausdruck ,INDEX = *FIRST / *LAST / *LOWER-BOUND / *UPPER-BOUND

Ergebnistyp

INTEGER

Eingabeparameter

ARRAY-NAME = string_ausdruck
Bezeichnet einen Array.

INDEX =
Gibt an, welcher Arrayindex abgefragt wird.

INDEX = *FIRST
Arrayindex des ersten Elements des Arrays, das einen gültigen Wert enthält.

INDEX = *LAST
Arrayindex des letzten Elements des Arrays, das einen gültigen Wert enthält.

INDEX = *LOWER-BOUND
Arrayindex, der bei der Variablendeklaration mit dem Kommando DECLARE-VARIABLE im Operanden MULTIPLE-ELEMENTS = *ARRAY (LOWER-BOUND =) festgelegt ist.

INDEX = *UPPER-BOUND
Arrayindex, der bei der Variablendeklaration mit dem Kommando DECLARE-VARIABLE im Operanden MULTIPLE-ELEMENTS = *ARRAY (UPPER-BOUND =) festgelegt ist.

Ergebnis

Index des Arrayelements, wird als Integer-Wert zurückgegeben.

Fehlermeldungen

```
SDP0423   VARIABLE '(&00)' KEIN ARRAY
SDP1007   NOCH KEINE VARIABLE ANGELEGT
SDP1052   AGGREGATELEMENT NICHT VORHANDEN
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

Der Array AR wird deklariert und erhält folgenden Inhalt:

```
/DECLARE-VARIABLE AR,TYPE = *STRING, MULTIPLE-ELEMENTS = *ARRAY
/AR#2 = 'abc'
/AR#3 = 'cde'
/AR#4 = ' '

/ARIND = ARRAY-INDEX('AR', *FIRST)
/SHOW-VARIABLE ARIND
ARIND = 2

/ARIND = ARRAY-INDEX('AR', *LAST)
/SHOW-VARIABLE ARIND
ARIND = 4

/ARIND = ARRAY-INDEX('AR', *LOWER-BOUND)
/SHOW-VARIABLE ARIND
ARIND = 0

/ARIND = ARRAY-INDEX('AR', *UPPER-BOUND)
/SHOW-VARIABLE ARIND
ARIND = 2147483647
```

BOOLEAN( ) Konvertieren in booleschen Wert

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion BOOLEAN( ) konvertiert den Ausdruck, der beim Funktionsaufruf angegeben wird, in einen Wert vom Typ BOOLEAN.

Format

BOOLEAN( ) BOOLE( )
EXPRESSION = ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

EXPRESSION = ausdruck
Bestimmt den Ausdruck, der konvertiert werden soll. Wenn „ausdruck“ nicht konvertiert werden konnte, wird eine Fehlermeldung ausgegeben.

Ergebnis

Datentyp	Ergebnis
ausdruck = 'TRUE' oder 'true' ausdruck = 'FALSE' oder 'false'	TRUE FALSE
ausdruck = 0 ausdruck ungleich 0	FALSE TRUE

Fehlermeldung

```
SDP0429   KONVERTIERUNG NICHT MOEGLICH
```

Beispiel

```
/A = 0
/B = BOOLEAN(EXPRESSION = A)
/SHOW-VARIABLE B
B = FALSE
```

CHARACTER-TO-INTEGER( ) Zeichen in Zahl konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion CHARACTER-TO-INTEGER( ) konvertiert *ein* Zeichen in eine Dezimalzahl, ausgehend von der Codierung des Zeichens im EBCDI-Code .

Besteht der Eingabestring aus mehreren Zeichen, wird nur das erste Zeichen umgesetzt.

Durch Kombination mit den entsprechenden String-Funktionen (z.B. SUBSTRING) können aber auch alle Zeichen eines Strings konvertiert werden.

Format

CHARACTER-TO-INTEGER( ) CHAR-TO-INT( )
STRING = string_ausdruck

Ergebnistyp

INTEGER (<integer 0..255>)

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den String, dessen erstes Zeichen konvertiert wird.
Bezeichnet „string_ausdruck“ den Leerstring, wird eine Fehlermeldung ausgegeben.

Ergebnis

Integer <integer 0..255>

Fehlermeldung

```
SDP0417   ANGEGEBENER STRING LEER. FUNKTION NICHT AUSGEFUEHRT
```

Beispiel 1: Konvertierung eines Zeichens

```
/C = CHARACTER-TO-INTEGER(STRING = 'ABC')
/SHOW-VARIABLE C
C = 193
```

Das erste Zeichen des Strings, das A, wird im EBCDI-Code in Halbbyte-Schreibweise als X'C1' dargestellt, in Binär-Schreibweise als B'11000001'. Dies entspricht dem Wert 193 im Dezimalsystem (= 128 + 64 + 1). CHARACTER-TO-INTEGER( ) konvertiert daher den Buchstaben A in die Zahl 193.

Beispiel 2: Konvertierung aller Zeichen eines Strings

```
/BEGIN-BLOCK
/   ZAHL = 1
/   ZFOLGE = 'ABC'
/   SUBST: KONVFOLGE = SUBSTRING(ZFOLGE, ZAHL)
/   CODE = CHARACTER-TO-INTEGER(STRING = KONVFOLGE)
/   SHOW-VARIABLE CODE
/   ZAHL = ZAHL+1
/   IF (ZAHL < 4)
/      GOTO SUBST
/   END-IF
/END-BLOCK
```

SHOW-VARIABLE gibt in einer Schleife nacheinander die Zahlen 193, 194 und 195 aus.

CHECK-DATA-TYPE( ) Operandenwert überprüfen

Anwendungsgebiet: **Stringfunktionen/ Prüffunktionen**

Die Funktion CHECK-DATA-TYPE( ) prüft bei Strings bzw. Operandenwerten, ob sie SDF-Datentyp-Bestimmungen erfüllen (siehe dazu „Datentypen“ auf Seite 550 und „Zusätze zu Datentypen“ auf Seite 556ff).
Dabei beschreibt CHECK-DATA-TYPE( ) den Datentyp, den der Eingabewert 'INPUT' erfüllen muss. Dieser Datentyp folgt den SDF-Datentyp-Richtlinien wie sie im Programm SDF-A bei der Anweisung //ADD-VALUE beschrieben sind (siehe Handbuch „SDF-A“ [16]). D.h. die Operanden von CHECK-DATA-TYPE( ) sind vollständig mit denen von //ADD-VALUE kompatibel. Nur Operandenkombinationen, die nach der Syntax von //ADD-VALUE erstellt sind, werden berücksichtigt. Andere Kombinationen werden ignoriert. Im Folgenden ist eine kurze Zusammenfassung der erlaubten Operandenkombinationen aufgelistet.

DATA-TYPE=	**erlaubte Operandenkombinationen**
*NOCHECK	VALUE, PATTERN
*INTEGER	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*X-STRING	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH, ODD
*C-STRING	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*NAME	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH, UNDERSCORE
*ALPHANUMERIC-NAME	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*STRUCTURED-NAME	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*FILENAME *PARTIAL-FILENAME *POSIX-FILENAME *POSIX-PATHNAME	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH, PATTERN, CAT-ID, USER-ID, VERSION, GENERATION, WILDCARD
*TIME	VALUE
*DATE	VALUE
*COMPOSED-NAME	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*TEXT	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*CAT-ID	VALUE
*KEYWORD	VALUE, KEYSTAR
*KEYWORD-NUMBER	VALUE, KEYSTAR
*VSN	VALUE
*X-TEXT	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH
*FIXED	VALUE, SHORTEST-LENGTH, LONGEST-LENGTH, DECIMAL-DIGITS-SHORTEST, DECIMAL-DIGITS-LONGEST

DATA-TYPE=	**erlaubte Operandenkombinationen**
*DEVICE	VALUE, ALIAS, VOLUME-ONLY, DEVICE-CLASS, EXCEPT-DISKS, EXCEPT-TAPES
*PRODUCT-VERSION	VALUE, CORRECTION-STATE, USER-INTERFACE

Format

(Teil 1 von 2)

CHECK-DATA-TYPE( )

INPUT = string_ausdruck

,DATA-TYPE = *NOCHECK / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME/
*STRUCTURED-NAME / *FILENAME / *FULL-FILENAME / *PARTIAL-FILENAME /
*POSIX-FILENAME / *POSIX-PATHNAME / *TIME / *DATE / *COMPOSED-NAME / *TEXT /
*CAT-ID / *KEYWORD / *KEYWORD-NUMBER / *VSN / *X-TEXT / *FIXED / *DEVICE /
*PRODUCT-VERSION

,SHORTEST-LENGTH = *ANY / arithm_ausdruck

,LONGEST-LENGTH = *ANY / arithm_ausdruck

,LONGEST-LOGICAL-LENGTH = *NONE / arithm_ausdruck

,DECIMAL-DIGITS-SHORTEST = 0 / arithm_ausdruck

,DECIMAL-DIGITS-LONGEST = 0 / arithm_ausdruck

,VALUE = *NO / list-poss: string_ausdruck

,PATTERN = *NO / string_ausdruck

,CAT-ID = *YES / *NO

,USER-ID = *YES / *NO

,VERSION = *YES / *NO

,GENERATION = *YES / *NO

,WILDCARD = *NO / *YES

,KEYSTAR = *NO / *YES

,SEPARATORS = *YES / *NO

,UNDERSCORE = *NO / *YES

,ODD = *YES / *NO

,CORRECTION-STATE = *YES / *NO / *ANY

,USER-INTERFACE = *YES / *NO / *ANY

,ALIAS = *YES / *NO

,VOLUME-ONLY = *NO / *YES

,WILDCARD-TYPE = *SELECTOR / *CONSTRUCTOR

,LOWER-CASE = *NO / *YES

,QUOTES = *OPTIONAL / *MANDATORY

,TEMPORARY-FILE = *YES / *NO

Fortsetzung ➠

(Teil 1 von 2)

,SCOPE = *ALL / *STD-DISK

,DEVICE-CLASS = *DISK / *TAPE / *DISK-OR-TAPE

,EXCEPT-DISKS = *NONE / list-poss: string_ausdruck

,EXCEPT-TAPES = *NONE / list-poss: string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

INPUT = string_ausdruck
Bezeichnet den zu überprüfenden Operandenwert.

DATA-TYPE =
Bezeichnet das Überprüfungskriterium hinsichtlich des Datentyps.

DATA-TYPE = *NOCHECK
Der Operandenwert wird nicht hinsichtlich des Datentyps überprüft. Die Überprüfung findet nur hinsichtlich des Wildcard-Suchmusters statt.
In diesem Fall darf nicht PATTERN = *NO angegeben sein.

DATA-TYPE = *INTEGER
Es wird überprüft, ob der Operandenwert vom Datentyp integer ist.

DATA-TYPE = *X-STRING
Es wird überprüft, ob der Operandenwert vom Datentyp x-string ist.

DATA-TYPE = *C-STRING
Es wird überprüft, ob der Operandenwert vom Datentyp c-string ist.

DATA-TYPE = *NAME
Es wird überprüft, ob der Operandenwert vom Datentyp name ist.

DATA-TYPE = *ALPHANUMERIC-NAME
Es wird überprüft, ob der Operandenwert vom Datentyp alphanumeric-name ist.

DATA-TYPE = *STRUCTURED-NAME
Es wird überprüft, ob der Operandenwert vom Datentyp structured-name ist.

DATA-TYPE = *FILENAME
Es wird überprüft, ob der Operandenwert vom Datentyp filename ist.

DATA-TYPE = *FULL-FILENAME
Es wird überprüft, ob der Operandenwert vom Datentyp full-filename ist.
Die Angabe *FULL-FILENAME wird nur noch aus Kompatibilität unterstützt. Ab SDF V4.1A ersetzt der Datentyp filename den Datentyp full-filename (wird an der Benutzerschnittstelle auch als filename dargestellt).

DATA-TYPE = *PARTIAL-FILENAME
Es wird überprüft, ob der Operandenwert vom Datentyp partial-filename ist.

DATA-TYPE = *POSIX-FILENAME
Es wird überprüft, ob der Operandenwert vom Datentyp posix-filename ist.

DATA-TYPE = *POSIX-PATHNAME
Es wird überprüft, ob der Operandenwert vom Datentyp posix-pathname ist.

DATA-TYPE = *TIME
Es wird überprüft, ob der Operandenwert vom Datentyp time ist.

DATA-TYPE = *DATE
Es wird überprüft, ob der Operandenwert vom Datentyp date ist.

DATA-TYPE = *COMPOSED-NAME
Es wird überprüft, ob der Operandenwert vom Datentyp composed-name ist.

DATA-TYPE = *TEXT
Es wird überprüft, ob der Operandenwert vom Datentyp text ist.

DATA-TYPE = *CAT-ID
Es wird überprüft, ob der Operandenwert vom Datentyp cat-id ist.

DATA-TYPE = *KEYWORD
Es wird überprüft, ob der Operandenwert vom Datentyp keyword ist.

DATA-TYPE = *KEYWORD-NUMBER
Es wird überprüft, ob der Operandenwert vom Datentyp keyword-number ist.

DATA-TYPE = *VSN
Es wird überprüft, ob der Operandenwert vom Datentyp vsn ist.

DATA-TYPE = *X-TEXT
Es wird überprüft, ob der Operandenwert vom Datentyp x-text ist.

DATA-TYPE = *FIXED
Es wird überprüft, ob der Operandenwert vom Datentyp fixed ist.

DATA-TYPE = *DEVICE
Es wird überprüft, ob der Operandenwert vom Datentyp device ist.

DATA-TYPE = *PRODUCT-VERSION
Es wird überprüft, ob der Operandenwert vom Datentyp product-version ist.

SHORTEST-LENGTH = *ANY / arithm_ausdruck
Irrelevant für die Datentypen date, time, cat-id, keyword und keyword-number.
Bestimmt, ob der Operandenwert eine Mindestlänge an Zeichen oder Anzahl der Bytes (für den Datentypen x-string) erfüllen muss.
Für den Datentypen integer bedeutet SHORTEST-LENGTH den niedrigsten Wert.
Für den Datentypen fixed muss SHORTEST-LENGTH mit DECIMAL-DIGITS-SHORTEST kombiniert werden.

LONGEST-LENGTH = *ANY / arithm_ausdruck
Irrelevant für die Datentypen date, time, cat-id, keyword und keyword-number.
Bestimmt, ob der Operandenwert eine Maximallänge an Zeichen oder Anzahl der Bytes (für den Datentypen x-string) erfüllen muss.
Für den Datentypen integer bedeutet LONGEST-LENGTH den niedrigsten Wert.
Für den Datentypen fixed muss LONGEST-LENGTH mit DECIMAL-DIGITS-LONGEST kombiniert werden.

LONGEST-LOGICAL-LENGTH = *NONE / arithm_ausdruck
Nur relevant in Zusammenhang mit PATTERN = string_ausdruck.
Bestimmt die Maximallänge des Operandenwerts, zu dem ein Wildcardausdruck noch als passend erkannt werden soll.

LONGEST-LOGICAL-LENGTH = *NONE
Die Maximallänge für den angegebenen Datentyp wird von SDF gesetzt.

DECIMAL-DIGITS-SHORTEST = 0 / arithm_ausdruck
Nur relevant für den Datentyp fixed.
Bestimmt, welche Mindestanzahl es für die Dezimalstellen geben darf.

DECIMAL-DIGITS-LONGEST = 0 / arithm_ausdruck
Nur relevant für den Datentyp fixed.
Bestimmt, welche Maximalanzahl es für die Dezimalstellen geben darf.

VALUE =
Bestimmt, welche Werte als Eingabe zulässig sind.

VALUE = *NO
Alle dem jeweiligen Operandentyp entsprechenden Werte sind zulässig. Einschränkungen gibt es nur, sofern diese bei der Festlegung des Operandentyps angegeben sind (z.B. Längenbegrenzungen). Für Operandenwerte des Typs keyword ist *NO nicht zulässig.

VALUE = list-poss: string_ausdruck
Die erlaubten Werte sind auf die angegebenen Werte beschränkt. Der Benutzer kann die angegebenen Werte bei der Eingabe abkürzen. Für Werte vom Typ keyword kann keine Liste von Einzelwerten verwendet werden (ein spezifisches CHECK-DATA-TYPE muss für jeden einzelnen Wert eingegeben werden).

PATTERN =
Wildcard-Suchmuster, nach dem der Operandenwert durchsucht wird.

PATTERN = *NO
Es gibt kein Wildcard-Suchmuster.

PATTERN = string_ausdruck
Der Operandenwert wird nach dem angegebenen Wildcard-Suchmuster durchsucht.

CAT-ID = *YES / *NO
Nur relevant für die Datentypen filename und partial-filename.
Bestimmt, ob die Katalogkennung als Teil eines Dateinamens angegeben werden darf.

USER-ID = *YES / *NO
Nur relevant für die Datentypen filename und partial-filename.
Bestimmt, ob die Katalogkennung als Teil eines Dateinamens angegeben werden darf.
Bestimmt, ob die Benutzerkennung als Teil eines Dateinamens angegeben werden darf.

VERSION = *YES / *NO
Nur relevant für die Datentypen filename und partial-filename.
Bestimmt, ob die Katalogkennung als Teil eines Dateinamens angegeben werden darf.
Bestimmt, ob die Versionsbezeichnung als Teil eines Dateinamens angegeben werden darf.

GENERATION = *YES / *NO
Nur relevant für die Datentypen filename und partial-filename.
Bestimmt, ob die Katalogkennung als Teil eines Dateinamens angegeben werden darf.
Bestimmt, ob die Generierungsbezeichnung als Teil eines Dateinamens angegeben werden darf.

WILDCARDS = *NO / *YES
Nur relevant für die Datentypen filename, partial-filename, name, alphanum-name und composed-name.
Bestimmt, ob der Operandenwert Wildcards bzw. Platzhalterzeichen enthalten darf.
Die Angabe *YES darf nicht mit PATTERN = string_ausdruck gekoppelt werden.

KEYSTAR = *NO / *YES
Nur relevant für die Datentypen keyword und keyword-number.
Bestimmt, ob der Operandenwert einen führenden Stern enthalten muss.

SEPARATORS = *YES / *NO
Nur relevant für den Datentyp text.
Bestimmt, ob Trennzeichen angegeben werden dürfen.

UNDERSCORE = *NO / *YES
Nur relevant für die Datentypen name und composed-name.
Bestimmt, ob der Operandenwert Unterstriche enthalten darf.

ODD = *YES / *NO
Nur relevant für den Datentyp x-text.
Bestimmt, ob eine ungerade Zahl von Zeichen akzeptiert wird.

CORRECTION-STATE = *YES / *NO / *ANY
Nur relevant für den Datentyp product-version.
Bestimmt, ob der Korrekturstand angegeben werden muss.
*ANY: Es wird nicht geprüft, ob der Korrekturstand angegeben ist.

USER-INTERFACE = *YES / *NO / *ANY
Nur relevant für den Datentyp product-version.
Bestimmt, ob der Freigabestand der Benutzerschnittstelle angegeben werden darf.
*ANY: Es wird nicht geprüft, ob der Freigabestand angegeben ist.

ALIAS = *YES / *NO
Nur relevant für den Datentyp device.
Bestimmt, ob Alias-Namen angegeben werden dürfen.

VOLUME-ONLY = *NO / *YES
Nur relevant für den Datentyp device.
Bestimmt, ob der Volumetyp akzeptiert wird.

WILDCARD-TYPE = *SELECTOR / *CONSTRUCTOR
Nur relevant für die Datentypen filename, name, alphanum-name und structured-name.
Bestimmt, ob der angegebene Operandenwert als Auswahlzeichenfolge oder als Konstruktionszeichenfolge interpretiert werden soll.

LOWER-CASE = *NO / *YES
Nur relevant für den Datentyp name.
Bestimmt, ob der Operandenwert Kleinbuchstaben enthalten darf.

QUOTES = *OPTIONAL / *MANDATORY
Nur relevant für die Datentypen posix-filename und posix-pathname.
Bestimmt, ob der Operandenwert Anführungszeichen enthalten darf.

TEMPORARY-FILE = *YES / *NO
Nur relevant für den Datentyp filename.
Bestimmt, ob als Operandenwert der Name einer temporären Datei erlaubt ist.

SCOPE = *ALL / *STD-DISK
Nur relevant für den Datentyp device.
Bestimmt, ob als Operandenwert der Name eines beliebigen Plattengerätes oder eines Standard-Plattengerätes angegeben werden darf.

DEVICE-CLASS = *DISK / *TAPE / *DISK-OR-TAPE
Nur relevant für den Datentyp device.
Bestimmt, welcher Geräteklasse (Platten- und/oder Bandgeräte) die angegebenen Geräte angehören dürfen.

EXCEPT-DISKS = *NONE / list-poss(50): string_ausdruck
Nur relevant für den Datentyp device.
Bestimmt, welche Plattengeräte aus der Liste der verfügbaren Geräte nicht angegeben werden dürfen.

EXCEPT-TAPES = *NONE / list-poss(50): string_ausdruck
Nur relevant für den Datentyp device.
Bestimmt, welche Bandgeräte aus der Liste der verfügbaren Geräte nicht angegeben werden dürfen.

Ergebnis

TRUE
Der angegebene Operandenwert erfüllt die Prüfkriterien.

FALSE
Der angegebene Operandenwert erfüllt die Prüfkriterien nicht.

Fehlermeldungen

```
SDP0099   KEIN VIRTUELLER SPEICHERPLATZ Z.ZT. VORHANDEN
SDP0454   UNGUELTIGER PARAMETER: '(&00)'
SDP0459   PARAMETER-FEHLER ODER UNGUELTIGE PARAMETER-KOMBINATION.WEITERE
          INFORMATION: '(&00)'
```

Beispiel

```
/A = CHECK-DATA-TYPE(':CAT:$USER.MYFILE', DATA-TYPE=*FILENAME)
/SHOW-VARIABLE A
A = TRUE

/A = CHECK-DATA-TYPE(':CAT:$USER.MYFILE', DATA-TYPE=*FILENAME, CAT-ID=*NO)
/SHOW-VARIABLE A
A = FALSE

/A = CHECK-DATA-TYPE('PAR', DATA-TYPE=*KEYWORD,VALUE='PARAMETERS')
/SHOW-VARIABLE A
A = TRUE

/A = CHECK-DATA-TYPE('PAR', DATA-TYPE=*KEYWORD, VALUE='PARAMETERS',-
/KEYSTAR=*YES)
/SHOW-VARIABLE A
A =FALSE
```

COUNTER( ) Funktionsaufrufe zählen

Anwendungsgebiet: **taskspezifischer Zähler**

Die Funktion COUNTER( ) ermittelt die Anzahl der Aufrufe von COUNTER( ) in der aktuellen Task. Nach jedem Aufruf von COUNTER( ) wird der Zähler um 1 erhöht.

Format

COUNTER( )

Ergebnistyp

INTEGER (<integer 1 .. 2147483647>)

Eingabeparameter

Keine

Ergebnis

Maximal zehnstellige Zahl

Fehlermeldung

```
SDP0304   UEBERLAUF, ZAHL AUSSERHALB DES MOEGLICHEN WERTEBEREICHS
```

Beispiel

```
/FOR I = *COUNTER(FROM = 2, TO = 28, INCREMENT = 2)
/   A = COUNTER()
/END-FOR

/SHOW-VARIABLE A
A = 14
```

CURRENT-TYPE( ) Variablentyp abfragen

Anwendungsgebiet: **Variablenzugriff** (Variablenname)

Die Funktion CURRENT-TYPE( ) gibt den aktuellen Typ des Werts einer einfachen Variable zurück (dieser darf nicht mit dem aktuellen Typ einer Variablendeklaration verwechselt werden, der wiederum mit VARIABLE-ATTRIBUTE( ) zurückgegeben wird). Wenn der Variablentyp noch nicht festgelegt ist (TYPE = *ANY) oder CURRENT-TYPE( ) auf eine zusammengesetzte Variable angewendet wird, wird als Ergebnis *NONE zurückgegeben.

Format

CURRENT-TYPE( ) CURR-TYPE( )
VARIABLE-NAME = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet die Variable, die abgefragt werden soll. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn er direkt (als Literal) angegeben werden soll (siehe dazu das Beispiel auf der folgenden Seite sowie das Beispiel bei IS-DECLARED( ) auf Seite 417).

Ergebnis

**BOOLEAN*
„string_ausdruck“ bezeichnet eine Variable, die den Wert BOOLEAN enthält (die Variable muss entweder als Typ *BOOLEAN oder *ANY deklariert sein).

**INTEGER*
„string_ausdruck“ bezeichnet eine Variable, die den Wert INTEGER enthält (die Variable muss entweder als Typ INTEGER oder ANY deklariert sein).

**NONE*
Die Variable „string_ausdruck“ hat noch keinen definierten Variablentyp, oder „string_ausdruck“ bezeichnet eine zusammengesetzte Variable.

*STRING
„string_ausdruck“ bezeichnet eine Variable, die den Wert STRING enthält (die Variable muss entweder als Typ STRING oder ANY deklariert sein).

Fehlermeldungen

```
SDP1007   NOCH KEINE VARIABLE ANGELEGT
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

```
/DECLARE-VARIABLE A (TYPE = *ANY)
/B = CURRENT-TYPE(VARIABLE-NAME = 'A')
/SHOW-VAR B
B = *NONE

/A = 123
/B = CURRENT-TYPE(VARIABLE-NAME = 'A')
/SHOW-VAR B
B = *INTEGER
```

DATE( ) Datum ausgeben

Anwendungsgebiet: **Umgebungsinformationen** (Kalender)

Die Funktion DATE( ) ermittelt das aktuelle Datum und gibt es im gewählten Format zurück.

Format

DATE( )
FORMAT = *ISO / *AMERICAN / *GERMAN ,MODE = *LOCAL-TIME / *UNIVERSAL-TIME

Ergebnistyp

STRING (<string 10..13>)

Eingabeparameter

FORMAT = *ISO / *AMERICAN / *GERMAN
Legt fest, in welchem Format das Datum ausgegeben wird.

MODE = *LOCAL-TIME / *UNIVERSAL-TIME
Bestimmt, ob das Datum in der lokalen Landeszeit (LOCAL-TIME) oder in der universellen Weltzeit (UNIVERSAL-TIME) ausgegeben wird.
Zu LOCAL-TIME (LT) und UNIVERSAL-TIME (UTC) siehe auch Makro GTIME im Handbuch „Makroaufrufe an den Ablaufteil“ [7].

Ergebnis

Eingabeparameter FORMAT =	Datumsformat <string 10..13>
*AMERICAN	mm/tt/jjiii
*ISO	jjjj-mm-ttiii
*GERMAN	tt.mm.jjjj

iii	Tag im laufenden Jahr (001 .. 366)
jj	zweistellige Jahreszahl
jjjj	vollständige Jahreszahl
mm	zweistellige Monatszahl (01 .. 12)
tt	Tag im laufenden Monat (01 .. 31)

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/G = DATE(FORMAT = *GERMAN)
/SHOW-VARIABLE G
G = 16.04.2007

/A = DATE(FORMAT = *AMERICAN)
/SHOW-VARIABLE A
A = 04/16/07106

/I = DATE( )
/SHOW-VARIABLE I
I = 2007-04-16106
```

Der 16. April ist der 106-ste Tag des Jahres 2007. Daher enthält die Darstellung des Datums im ISO-Format und im amerikanischen Format den Zusatz 106.

DATE-VALUE( ) Bestimmtes Datum ausgeben

Anwendungsgebiet: **Umgebungsinformationen** (Kalender)

Die Funktion DATE-VALUE( ) gibt ein bestimmtes Datum aus, das einer angegebenen Zahl von Tagen seit dem Basisdatum entspricht (Defaultwert ist der Beginn des 20. Jahrhunderts, d.h. das Datum 1900-01-01).

Format

DATE-VALUE( )
NUMBER-OF-DAYS = arithm_ausdruck ,BASE = *STD / *TODAY / string_ausdruck ,FORMAT = *ISO / *AMERICAN / *GERMAN

Ergebnistyp

STRING (<string 10..13>)

Eingabeparameter

NUMBER-OF-DAYS = arithm_ausdruck
Anzahl der Tage ab dem Basisdatum.

BASE =
Bezeichnet das Basisdatum.
Das Format des BASE-Wert hängt nicht vom Wert des FORMAT-Operanden ab.
Das Format kann *ISO, *GERMAN oder *AMERICAN sein, mit Tag und Monat in einer oder zwei Dezimalstellen (eine führende Null ist für die ersten 9 Tage bzw. Monate nicht erforderlich) und mit dem Jahr in zwei oder vier Dezimalstellen (ohne der Anzahl der Tage im gegenwärtigen Jahr). Bei einer zweistelligen Jahresangabe werden die ersten beiden Dezimalstellen wie bei dem SDF-Datentyp date ergänzt:

Eingabe-Jahr	Die ersten beiden Dezimalstellen
00..59	19
60..99	20

BASE = *STD
Basisdatum ist der Beginn des 20. Jahrhunderts (1900-01-01).

BASE = *TODAY
Basisdatum ist das aktuelle Datum.

BASE = string_ausdruck
Gibt das Basisdatum an.
Der Wert ist ein Datum ab 1582-10-15.

FORMAT = *ISO / *AMERICAN / *GERMAN
Legt fest, in welchem Format das Datum ausgegeben wird.

Ergebnis

Eingabeparameter FORMAT =	Datumsformat <string 10..13>
*AMERICAN	mm/tt/jjiii
*ISO	jjjj-mm-ttiii
*GERMAN	tt.mm.jjjj

iii	Tag im laufenden Jahr (001 .. 366)
jj	zweistellige Jahreszahl
jjjj	vollständige Jahreszahl
mm	zweistellige Monatszahl (01 .. 12)
tt	Tag im laufenden Monat (01 .. 31)

Fehlermeldung

```
SDP0452   UNGUELTIGES DATUM
```

Beispiel

```
/A = DATE-VALUE(NUMBER-OF-DAYS = 23008, FORMAT = *ISO)
/SHOW-VARIABLE A
A = 1962-12-30364

/A = DATE-VALUE(NUMBER-OF-DAYS = 23008, FORMAT = *AMERICAN)
/SHOW-VARIABLE A
A = 12/30/62364

/A = DATE-VALUE(NUMBER-OF-DAYS = 23008, FORMAT = *GERMAN)
/SHOW-VARIABLE A
A = 30.12.1962

/TOMORROW = DATE-VALUE(NUMBER-OF-DAYS = 1, BASE = *TODAY)
/TODAY = DATE( )
/SHOW-VARIABLE (TODAY,TOMORROW)
TODAY = 2001-08-09221
TOMORROW = 2001-08-10222
```

DAY( ) Wochentag ausgeben

Anwendungsgebiet: **Umgebungsinformationen** (Kalender)

Die Funktion DAY( ) liefert den Namen des aktuellen Wochentages in der angegebenen Sprache, allerdings nur in einer Kurzform.

Format

DAY( )
LANGUAGE = *ENGLISH / *GERMAN / *STD

Ergebnistyp

STRING (<string 2..3>)

Eingabeparameter

LANGUAGE = *ENGLISH / *GERMAN / *STD
Legt fest, in welcher Sprache der Name des aktuellen Wochentages zurückgegeben wird. Bei Angabe von *STD erfolgt die Ausgabe in der für die Task voreingestellten Sprache.

Ergebnis

Die Abkürzung des Wochentages ist abhängig von der angegebenen Sprache.

Eingabeparameter	Ergebnis
*ENGLISH	SUN / MON / TUE / WED / THU / FRI / SAT
*GERMAN	SO / MO / DI / MI / DO / FR / SA

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/G = DAY(LANGUAGE = *GERMAN)
/SHOW-VARIABLE G
G = MO
/E = DAY( )
/SHOW-VARIABLE E
E = MON
```

ELAPSED-DAYS( ) Differenztage ausgeben

Anwendungsgebiet: **Umgebungsinformationen** (Kalender)

Die Funktion ELAPSED-DAYS( ) gibt die Anzahl der Differenztage zwischen zwei Datumsangaben an.

Format

ELAPSED-DAYS( )
DATE = string_ausdruck ,BASE = *STD / *TODAY / string_ausdruck

Ergebnistyp

INTEGER (<integer -3074323 .. 3074323>)

Eingabeparameter

DATE = string_ausdruck
Bezeichnet das Endedatum.
Es sind nur die Formate *ISO, *AMERICAN und *GERMAN erlaubt. Der Wert muss größer oder gleich als 1582-10-15 sein

BASE =
Bezeichnet das Basisdatum.
Es sind nur die Formate *ISO, *AMERICAN und *GERMAN erlaubt.

BASE = *STD
Basisdatum ist der Beginn des 20. Jahrhunderts (1900-01-01).

BASE = *TODAY
Basisdatum ist das aktuelle Datum.

BASE = string_ausdruck
Gibt Basisdatum an.
Der Wert ist ein Datum ab 1582-10-15.

Hinweise

Für die Angabe von BASE und DATE müssen folgende Regeln beachtet werden:
– Das Format kann nur *ISO, *GERMAN oder *AMERICAN sein.
– Eine führende Null ist für die ersten 9 Tage bzw. Monate nicht erforderlich.
  Das Jahr kann auf zwei Dezimalstellen abgekürzt werden. In diesem Fall werden die ersten beiden Stellen auf dieselbe Weise bestimmt wie bei dem SDF-Datentyp <date with-compl>:

Eingabe-Jahr	Die ersten beiden Dezimalstellen
00..59	19
60..99	20

Ergebnis

Zahl vom Typ Integer.

Fehlermeldung

```
SDP0452   UNGUELTIGES DATUM
```

Beispiel

```
/A = ELAPSED-DAYS (DATE='1963-12-30')
/SHOW-VARIABLE A
A = 23373

/A = ELAPSED-DAYS (DATE='12/30/1963', BASE = '12/30/1900')
/SHOW-VARIABLE A
A = 23010

/A = ELAPSED-DAYS (DATE='30.12.1963')
/SHOW-VARIABLE A
A = 23373

/DIFF = ELAPSED-DAYS (DATE='2001-08-23',BASE='2001-04-01')
/SHOW-VARIABLE DIFF
DIFF = 144

/DIFF = ELAPSED-DAYS (DATE='2001-08-23',BASE='2001-10-31')
/SHOW-VARIABLE DIFF
DIFF = -69
```

EXPLICIT-CALL( ) Expliziten Prozeduraufruf ausgeben

Anwendungsgebiet: **Prozedurinformationen**

Die Funktion EXPILICIT-CALL( ) gibt die Art des Prozeduraufrufs aus: Dabei bedeutet TRUE einen expliziten Aufruf (d.h. mit CALL-PROCEDURE, INCLUDE-PROCEDURE) und FALSE einen impliziten Aufruf (z.B. wenn der Aufruf von einem Kommando einer Prozedur erfolgte; siehe dazu die SDF-A-Anweisung //ADD-CMD ... IMPLEMENTOR= *PROCEDURE im Handbuch „SDF-A“ [16]).

Format

EXPLICIT-CALL( )

Ergebnistyp

BOOLEAN

Eingabeparameter

Keine

Ergebnis

TRUE
Der angegebene Aufruf ist ein expliziter Aufruf, d.h. er wurde mit CALL-PROCEDURE, INCLUDE-PROCEDURE gegeben (oder einem Alias- bzw. redefinierten Namen dafür).

FALSE
Der angegebene Aufruf ist kein expliziter Aufruf.

Fehlermeldung

Keine

Beispiel

```
/SET-PROCEDURE-OPTIONS "Procedure MYPROC"
/    WRITE-TEXT 'Expliziter Aufruf: &(EXPLICIT-CALL)'
/EXIT-PROCEDURE

/CALL-PROCEDURE MYPROC
Expliziter Aufruf: TRUE

/INCLUDE-PROCEDURE MYPROC
Expliziter Aufruf: TRUE

/DO MYPROC
Expliziter Aufruf: TRUE

/MY-COMMAND MYPROC "Benutzerkommando mit Implementierung der Prozedur MYPROC"
Expliziter Aufruf: FALSE
```

EXTEND-SDF-LIST( ) Listenelement anfügen

Anwendungsgebiet: **String-Bearbeitung**

Die Funktion EXTEND-SDF-LIST( ) fügt bei einer SDF-Liste ein neues Element hinzu. Dieses neue Element kann wiederum eine SDF-Liste sein.

Format

EXTEND-SDF-LIST( )
LIST = string_ausdruck ,ELEMENT = string_ausdruck ,POSITION = *LAST / *FIRST / arithm_ausdruck

Ergebnistyp

STRING

Eingabeparameter

LIST = string_ausdruck
Bezeichnet eine SDF-Liste. Eine leere Liste muss mit '()' angegeben werden. Die Eingabe wird intern mit IS-SDF-LIST auf Gültigkeit überprüft.

ELEMENT = string_ausdruck
Bezeichnet das Element, das hinzugefügt werden soll.

POSITION =
Gibt an, wo das Element hinzugefügt werden soll.

POSITION = *LAST
Das Element wird nach der Liste hinzugefügt.

POSITION = *FIRST
Das Element wird vor der Liste hinzugefügt.

POSITION = arithm_ausdruck
Das Element wird an der angegebenen Position eingefügt.
Liegt die angegebene Position außerhalb des zulässigen Bereichs wird *LAST angenommen.

Ergebnis

Ausdruck als erweiterter String

Fehlermeldung

```
SDP0447   DER ANGEGEBENE STRING IST KEINE SDF-LISTE

SDP0481   'POSITION' MUSS GROESSER NULL SEIN
```

Beispiel

```
A=EXTEND-SDF-LIST(LIST='(val1,val2)',ELEMENT='val3',POSITION=*last)
/SHOW-VARIABLE A
A = (val1,val2,val3)

/A=EXTEND-SDF-LIST(LIST=A,ELEMENT='val0',POSITION=*first)
/SHOW-VARIABLE A
A = (val0,val1,val2,val3)

/A=EXTEND-SDF-LIST(LIST=A,ELEMENT='(val4,val5)',POSITION=*last)
/SHOW-VARIABLE A
A = (val0,val1,val2,val3,(val4,val5))
```

EXTRACT-FIELD( ) Feld abtrennen

Anwendungsgebiet: **String-Bearbeitung**

Die Funktion EXTRACT-FIELD( ) trennt von einem Eingabe-String ein Feld ab.

Format

EXTRACT-FIELD( )
STRING = string_ausdruck ,FIELD-NUMBER = arithm_ausdruck ,FIELD-SEPARATOR = *ANY-BLANKS / string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet einen Eingabe-String.

FIELD-NUMBER = arithm_ausdruck
Bezeichnet die Feldnummer.

FIELD-SEPARATOR =
Gibt den Separator an. Separatoren sind nicht Teil des abgetrennten Felds.

FIELD-SEPARATOR = ***ANY-BLANKS**
Der Defaultwert bezeichnet ein oder mehrere Leerzeichen.
(Die Angabe '␣␣*' wird kombatibel unterstützt).

FIELD-SEPARATOR = string_ausdruck
string_ausdruck ist der Separator.
string_ausdruck muss hier allerdings ein einfacher regulärer Ausdruck sein (siehe dazu Handbuch „POSIX Kommandos“ [18]).

Ergebnis

Abgetrenntes Feld als String

Fehlermeldung

```
SDP0472   DIE OPERANDEN STRING UND FIELD-SEPARATOR DUERFEN KEIN NULLBYTE
          (X'00') ENTHALTEN

SDP0474   SYNTAXFEHLER IN REGULAEREM AUSDRUCK FUER OPERAND FIELD-SEPARATOR

SDP0485   'FIELD-NUMBER' MUSS GROESSER NULL SEIN
```

Beispiele

Beispiel 1

```
/DECLARE-VARIABLE mylist(TYPE=*STRING),MULTIPLE-ELEMENT=*LIST
/mylist = 'Pencil 100',WRITE-MODE=*EXTEND
/mylist = 'Table 5',WRITE-MODE=*EXTEND
/mylist = 'Lamp 20',WRITE-MODE=*EXTEND
/mylist = 'Paper 75',WRITE-MODE=*EXTEND
/mylist = 'Diskette 1000',WRITE-MODE=*EXTEND
/mylist = 'Envelope 1500',WRITE-MODE=*EXTEND

/FOR x = *LIST(mylist)
/  article = EXTRACT-FIELD(STRING=x,FIELD-NUMBER=1)
/  quantity = EXTRACT-FIELD(STRING=x,FIELD-NUMBER=2)
/  WRITE-TEXT '&quantity of &article are available'
/END-FOR
```

Ausgabe:

```
100 of Pencil are available
5 of Table are available
20 of Lamp are available
75 of Paper are available
1000 of Diskette are available
1500 of Envelope are available
```

Beispiel 2

```
/A=EXTRACT-FIELD(STRING='field1,field3,field4',FIELD-NUMBER=3,FIELD-
SEPARATOR=',')
/SHOW-VARIABLE A
A = field4
```

FILL( ) String auffüllen

Anwendungsgebiet: **String-Bearbeitung**

Die Funktion FILL( ) füllt den String, der im Funktionsaufruf angegeben wurde, bis zur angegebenen Länge und in der angegebenen Richtung mit Füllzeichen auf. Diese Füllzeichen können im Funktionsaufruf definiert werden.

Format

FILL( )
STRING = string_ausdruck ,LENGTH = arithm_ausdruck ,SIDE = *RIGHT / *LEFT ,FILL-BYTE = C'␣' / zeichen

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den Eingabestring, der auf die Länge gebracht werden soll, die mit dem Parameter LENGTH = definiert wird.

LENGTH = arithm_ausdruck
Positive Zahl, die die Länge des Ergebnisstrings bestimmt.
Wenn der Wert für LENGTH größer ist als die aktuelle Länge des Eingabestrings (siehe Parameter STRING) und innerhalb des für Stringlängen gültigen Wertebereichs liegt, wird der Eingabestring mit dem vereinbarten Füllzeichen (Parameter FILL-BYTE) aufgefüllt. Wenn „arithm_ausdruck“ außerhalb des gültigen Wertebereichs liegt, wird eine Fehlermeldung ausgegeben. Ist LENGTH kleiner als die Länge von STRING, so wird STRING als Ergebnis unverändert zurückgegeben.

SIDE =
Bestimmt die Ausrichtung des Eingabestrings im Ergebnisstring, das heißt, die Richtung, in der die Füllzeichen angefügt werden. SIDE hat keine Wirkung, wenn der Eingabestring unverändert zurückgegeben wird.

SIDE = *RIGHT
Fügt die Füllzeichen rechts an, das heißt hinter dem letzten Zeichen des Eingabestrings.

SIDE = *LEFT
Fügt die Füllzeichen links an, das heißt vor dem ersten Zeichen des Eingabestrings.

FILL-BYTE =
Bestimmt, mit welchem Zeichen der Eingabestring aufgefüllt wird. FILL-BYTE hat keine Wirkung, wenn der Eingabestring nicht verlängert wird.

FILL-BYTE = C' '
Füllt den Eingabestring (rechts oder links) mit Leerzeichen auf.

FILL-BYTE = zeichen
Füllt den Eingabestring (rechts oder links) mit dem an dieser Stelle angegebenen Zeichen auf. „zeichen“ ist ein beliebiges Zeichen. Wird mehr als ein Zeichen angegeben, wird nur das erste Zeichen als Füllzeichen verwendet.
Wird an Stelle eines Zeichens ein Nullstring angegeben (C''), wird eine Fehlermeldung ausgegeben.

Ergebnis

String in der Länge von LENGTH = zahl

Fehlermeldungen

```
SDP0431   FEHLER '(&00)' IN BUILTIN-FUNKTION '(&01)'

SDP0436   ANGEGEBENE LAENGE LIEGT NICHT ZWISCHEN NULL UND MAXIMAL MOEGLICHER
          STRING-LAENGE

SDP0437   LAENGE DES PARAMETERS 'FILL-BYTE' GLEICH NULL
```

Beispiel

```
/A = 'ABCDE'
/SHOW-VARIABLE A
A = ABCDE

/A = FILL(STRING = A, LENGTH = 8, FILL-BYTE = C'.')
/SHOW-VARIABLE A
A = ABCDE...

/A = FILL(STRING = A, LENGTH = 12, SIDE = *LEFT, FILL-BYTE = C'.')
/SHOW-VARIABLE A
A = ....ABCDE...
```

FIRST-VARIABLE-NAME( ) Variablenelementnamen abfragen

Anwendungsgebiet: **Variablenzugriff (Variablenname)**

Mit der Funktion FIRST-VARIABLE-NAME( ) lässt sich der Aufbau von zusammengesetzten Variablen analysieren, vor allem in Kombination mit der Funktion NEXT-VARIABLE-NAME( ). Die Funktion FIRST-VARIABLE-NAME( ) kann auf alle Aggregate und Listen angewendet werden. Dabei geht FIRST-VARIABLE-NAME( ) vom angegebenen Variablen- oder Variablenelementnamen aus und liefert dann den Namen der ersten Variablen. Gibt es keine tiefere Ebene, liefert FIRST-VARIABLE-NAME( ) das Ergebnis *END.

Format

FIRST-VARIABLE-NAME( ) FIRST-VAR-NAME( )
VARIABLE-NAME = string_ausdruck

Ergebnistyp

STRING (<composed-name 1..255> oder '*END')

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet eine zusammengesetzte Variable oder ein Variablenelement.
Ist die zusammengesetzte Variable eine Liste, wird der Name des ersten Listenelements (liste#1) geliefert.
Ist das Variablenelement ein Listenelement, wird *END oder der Name des entsprechenden Elements ausgegeben. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe dazu das Beispiel auf der folgenden Seite).

Ergebnis

elementname, wenn „string_ausdruck“ eine zusammengesetzte Variable bezeichnet.

**END*
„string_ausdruck“ bezeichnet bereits ein Variablenelement der untersten Ebene, das selbst keine zusammengesetzte Variable mehr ist; das heißt, es gibt keine tiefere Ebene.

Fehlermeldung

```
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

Die Variable FSTAT wird als Array deklariert mit dynamischen Strukturen als Elementen.

```
/DECLARE-VARIABLE FSTAT(TYPE=*STRUCTURE(*DYNAMIC)),MULTIPLE-ELEMENTS=*ARRAY
```

Die Elemente der dynamischen Strukturen werden folgendermaßen initialisiert:

```
/FSTAT#1.F-NAME = 'DATEI.A'
/FSTAT#1.F-SIZE = 0000003
/FSTAT#2.F-NAME = 'DATEI.B'
/FSTAT#2.F-SIZE = 000006
```

Mit FIRST-VARIABLE-NAME kann die Variable analysiert werden:

```
/A = FIRST-VARIABLE-NAME('FSTAT')
/SHOW-VARIABLE A
A = FSTAT#1

/B = FIRST-VARIABLE-NAME(A)
/SHOW-VARIABLE B
B = FSTAT#1.F-NAME

/C = FIRST-VARIABLE-NAME(B)
/SHOW-VARIABLE C
C = *END
```

FROM-C-LITERAL( ) C-Literal konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion FROM-C-LITERAL( ) konvertiert ein C-Literal in den entsprechenden Stringwert. Dabei werden das vorangestellte C und die Hochkommas am Anfang und Ende des Literals gelöscht.

FROM-C-LITERAL( ) ist die inverse Funktion zu TO-C-LITERAL( ).

Format

FROM-C-LITERAL( ) FROM-C-LIT( )
STRING = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet ein C-Literal; das kennzeichnende C am Anfang des Literals wird entfernt, ebenso die Hochkommas am Anfang und Ende des Literals. Doppelte Hochkommas innerhalb des Strings werden zu einfachen Hochkommas reduziert.

Ergebnis

String

Fehlermeldung

```
SDP0433   EINGEGEBENER STRING KEIN C-LITERAL
```

Beispiel

```
/B = FROM-C-LITERAL(STRING = 'C''ABC''')
/SHOW-VARIABLE B
B = ABC

/A = 'ABC'
/B = FROM-C-LITERAL(STRING = A)

SDP0433 EINGEGEBENER STRING KEIN C-LITERAL
SDP0431 FEHLER 'SDP0433' IN BUILTIN-FUNKTION 'FROM-C-LITERAL'
SDP0239 FEHLER WAEHREND DER AUSWERTUNG DER RECHTEN SEITE DER ZUWEISUNG
```

ABC ist kein C-Literal und kann daher nicht konvertiert werden.

FROM-X-LITERAL( ) X-Literal konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion FROM-X-LITERAL( ) konvertiert ein X-Literal in den entsprechenden Stringwert.

FROM-X-LITERAL( ) ist die inverse Funktion zu TO-X-LITERAL( ).

Format

FROM-X-LITERAL( ) FROM-X-LIT( )
STRING = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet ein X-Literal.

Ergebnis

String

Fehlermeldung

```
SDP0434   EINGEGEBENER STRING KEIN X-LITERAL
```

Beispiel

```
/F = 'X''C1'''
/B = FROM-X-LITERAL(F)
/SHOW-VARIABLE B
B = A
```

HASH-STRING( ) Ausdruck als String verschlüsseln

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion HASH-STRING( ) konvertiert einen String-Ausdruck in einen String binären Inhalts mit frei zu wählender Länge (z.B. ein Passwort). Der verwendete Algorithmus liefert zu unterschiedlichen Strings mit hoher Wahrscheinlichkeit unterschiedliche Ausgaben.

Format

HASH-STRING( )
STRING = string_ausdruck ,LENGTH = 4 / arithm_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck_1..256
Bezeichnet den Eingabe-String.

LENGTH = 4 / arithm_ausdruck_1..256
Bezeichnet die Länge des Ausgabe-String.

Ergebnis

String

Fehlermeldung

```
SDP0455   EINGABESTRING MIT UNZULAESSIGER LAENGE (ERLAUBT: 1..256)
SDP0456   PARAMETER LENGTH AUSSERHALB DES GUELTIGEN WERTEBEREICHS
```

Beispiel

```
/A = HASH-STRING(STRING='AB',LENGTH=2)
/SHOW-VAR A, INFORMATION = *PARAMETERS(VALUE=*X-LITERAL)
A = X'8C5E'
```

HASH-VALUE( ) Ausdruck als Integer-Wert verschlüsseln

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion HASH-VALUE( ) konvertiert einen String-Ausdruck in einen Integer-Wert, wobei der verwendete Algorithmus zu unterschiedlichen Strings mit hoher Wahrscheinlichkeit unterschiedliche Ausgabewerte erzeugt.

Format

HASH-VALUE( )
STRING = string_ausdruck

Ergebnistyp

INTEGER (<integer $-2^{31}..2^{31}-1$>)

Eingabeparameter

STRING = string_ausdruck_1..256
Bezeichnet den Eingabe-String.

Ergebnis

Integer

Fehlermeldung

```
SDP0455   EINGABESTRING MIT UNZULAESSIGER LAENGE (ERLAUBT: 1..256)
```

Beispiel

```
/RANDOM = HASH-VALUE(TIME( )) MOD 60           "ZWISCHEN 0 UND 59"
/HOUR = INTEGER(SUBSTRING(TIME(),START=1,LENGTH=2))
/MINUTE = INTEGER(SUBSTRING(TIME( ),START=4,LENGTH=2))
/MINUTE = MINUTE + RANDOM
/IF (MINUTE > 59)
/      MINUTE = MINUTE - 60
/      HOUR = HOUR + 1
/       IF (HOUR > 23)
/             HOUR = 0
/       END-IF
/END-IF
/WRITE-TEXT 'In &RANDOM. Minute(n) ist es
    &HOUR.:&(FILL(STRING=STRING(MINUTE),LENGTH=2,SIDE=*LEFT,FILL-BYTE='0'))'

In 32 Minute(n) ist es    10:00
```

HOME-CAT-ID( ) Katalogkennung des Home-Pubsets abfragen

Anwendungsgebiet: **Umgebungsinformationen**

Die Funktion HOME-CAT-ID( ) liefert die Katalogkennung des Home-Pubsets.
Die Katalogkennung eines Pubsets wird von der Systembetreuung beim Einrichten des Pubsets vergeben und kann bis zu vier Zeichen lang sein. Der Home-Pubset des laufenden Systems ist der Pubset, von dem das System geladen wurde (siehe auch Handbuch „Einführung in die Systembetreuung“ [8]).

Format

HOME-CAT-ID( )

Ergebnistyp

STRING (<string 1..4>)

Eingabeparameter

Keine

Ergebnis

Katalogkennung des Home-Pubsets als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = HOME-CAT-ID()
/SHOW-VARIABLE A
A = 10SB
```

Der Home-Pubset hat demnach die Katalogkennung 10SB.

HOST( ) Rechnernamen abfragen

Anwendungsgebiet: **Umgebungsinformationen**

Die Funktion HOST( ) liefert den internen Namen des Rechners, auf dem die Funktion aufgerufen wird. Den internen Rechnernamen legt die Systembetreuung beim Starten des DCM fest.

Format

HOST( )

Ergebnistyp

STRING (<string 1..8>)

Eingabeparameter

Keine

Ergebnis

Name des Rechners als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = HOST()
/SHOW-VARIABLE A
A = D016ZE04
```

D016ZE04 ist der Rechnername.

INDEX( ) String suchen

Anwendungsgebiet: **String-Funktionen**

Die Funktion INDEX( ) zeigt die Position eines Suchstrings im Gesamtstring an. Der Gesamtstring kann von links nach rechts oder von rechts nach links durchsucht werden; der Ergebniswert bezieht sich immer auf den Anfang des Gesamtstrings.

Format

INDEX( )
STRING = string_ausdruck$_1$,PATTERN = string_ausdruck$_2$,DIRECTION = *FORWARD / *REVERSE ,BEGIN-COLUMN = 1 / arithm_ausdruck ,END-COLUMN = *LAST / arithm_ausdruck

Ergebnistyp

INTEGER

Eingabeparameter

STRING = string_ausdruck$_1$
Gesamtstring, der durchsucht wird.

PATTERN = string_ausdruck$_2$
Suchstring, der im Gesamtstring gesucht wird.

DIRECTION =
Suchrichtung

DIRECTION = *FORWARD
Der Gesamtstring wird vorwärts, das heißt von links nach rechts, durchsucht.

DIRECTION = *REVERSE
Der Gesamtstring wird rückwärts, das heißt von rechts nach links, durchsucht.

BEGIN-COLUMN =
Die Suche wird vom Beginn des Gesamtstrings gesehen auf einen bestimmten Spaltenbereich beschränkt. Angegeben wird das erste Zeichen im Gesamtstring an, ab dem die Suche nach dem Suchstring beginnt.
Der durchsuchte String ist leer, wenn der Gesamtstring weniger Zeichen enthält als bei BEGIN-COLUMN angegeben sind.

BEGIN-COLUMN = 1
Die Suche beginnt ab Spalte 1, d.h. der Gesamtstring wird von Anfang an durchsucht.

BEGIN-COLUMN = arithm_ausdruck
Der Gesamtstring wird ab der angegebenen Spalte bzw. ab diesem Zeichen nach dem Suchstring durchsucht.

END-COLUMN =
Die Suche wird vom Ende des Gesamtstrings gesehen auf einen bestimmten Spaltenbereich beschränkt. Angegeben wird das letzte Zeichen im Gesamtstring an, das in die Suche einbezogen wird. Alle nachfolgenden Zeichen werden ignoriert.

END-COLUMN = *LAST
Der Gesamtstring wird bis zum Ende durchsucht.

END-COLUMN = arithm_ausdruck
Der Gesamtstring wird bis zu der angegebenen Spalte bzw. bis zu einschließlich diesem Zeichen nach dem Suchstring durchsucht.

Ergebnis

Integer
Anfangsposition des Suchstrings im Gesamtstring.
Ist der Suchstring im Gesamtstring mehrfach enthalten, zeigt „zahl“ bei Suchrichtung von links nach rechts das erste Auftreten des Suchstrings an, bei Suchrichtung von rechts nach links das Letzte.

0
Der Suchstring ist länger als der Gesamtstring, oder der Suchstring ist nicht im Gesamtstring enthalten.

Fehlermeldungen

```
SDP0413   NICHT ZULAESSIGE LAENGENANGABE

SDP0493   BEGIN-INDEX, END-INDEX, BEGIN-COLUMN UND END-COLUMN MUESSEN
          GROESSER NULL SEIN

SDP0498   BEGIN-COLUMN DARF NICHT ALS END-COLUMN GROESSER SEIN
```

Beispiel 1

```
/A = INDEX(STRING = 'ABCDE', PATTERN = 'C')
/SHOW-VARIABLE A
A = 3

/B = INDEX(STRING = 'ABCDEABC', PATTERN = 'AB')
/SHOW-VARIABLE B
B = 1

/C = INDEX(STRING = 'ABCDEABC', PATTERN = 'AB', DIRECTION = *REVERSE)
/SHOW-VARIABLE C
C = 6
```

Beispiel 2

```
/STRING = '1080:0:0:0:8:800:200C:417A'
/
/WRITE-TEXT '- FROM LEFT TO RIGHT -'
/START = 1
/REPEAT
/   WRITE-TEXT '&(START) => &(SUBSTR( STRING, START ))'
/   START = INDEX( STRING, ':', *FORWARD, START, *LAST ) + 1
/UNTIL ( START == 1 )
/              &* AT THE LAST LOOP ITERATION
/              &* INDEX DOES NOT FIND THE ':'
/              &* AND RETURNED 0
/              &* 1 IS ADDED FROM THIS RETURNED VALUE.
/
/WRITE-TEXT '- FROM RIGHT TO LEFT -'
/END = LENGTH( STRING )
/REPEAT
/   WRITE-TEXT '&(END) => &(SUBSTR( STRING, 1, END ))'
/   END = INDEX( STRING, ':', *REVERSE, 1, END ) - 1
/UNTIL ( END <= 0 )
/              &* AT THE LAST LOOP ITERATION
/              &* INDEX DOES NOT FIND THE ':'
/              &* AND RETURNED 0.
/              &* 1 IS SUBTRACTED FROM THIS RETURNED VALUE.
/
/WRITE-TEXT '- SURROUNDING CUT -'
/START = 1
/END = LENGTH( STRING )
/REPEAT
/   TEXT = '&(START):&(END) => ' // -
/          SUBSTR( STRING, START, END - START + 1 )
/   WRITE-TEXT '&(TEXT)'
/   START = INDEX( STRING, ':', *FORWARD, START, END ) + 1
```

```
/   END = INDEX( STRING, ':', *REVERSE, START, END ) - 1
/UNTIL ( START > END )
```

Das Beispiel zeigt, wie ein String schrittweise nach Trennzeichen (hier Doppelpunkt) durchsucht und um die bereits durchsuchte Teilzeichenfolge reduziert werden kann. Die Suche und Reduzierung, die dabei in verschiedenen Richtungen (links nach rechts, rechts nach links und beidseitig) durchgeführt wird, liefert folgende Ausgabe:

```
- FROM LEFT TO RIGHT -
1 => 1080:0:0:8:800:200C:417A
6 => 0:0:0:8:800:200C:417A
8 => 0:0:8:800:200C:417A
10 => 0:8:800:200C:417A
12 => 8:800:200C:417A
14 => 800:200C:417A
18 => 200C:417A
23 => 417A
- FROM RIGHT TO LEFT -
26 => 1080:0:0:0:8:800:200C:417A
21 => 1080:0:0:0:8:800:200C
16 => 1080:0:0:0:8:800
12 => 1080:0:0:0:8
10 => 1080:0:0:0
8 => 1080:0:0
6 => 1080:0
4 => 1080
- SURROUNDING CUT -
1:26 => 1080:0:0:0:8:800:200C:417A
6:21 => 0:0:0:8:800:200C
8:16 => 0:0:8:800
10:12 => 0:8
```

INSTALLATION-PATH( ) Pfadnamen ausgeben

Anwendungsgebiet: **Umgebungsinformationen**

Die Funktion INSTALLATION-PATH( ) gibt für den logischen Namen einer Datei (Installation-Item), die zu einer bestimmten Produktversion gehört, den zugeordneten Pfadnamen aus dem SCI aus.

Die Zuordnung zwischen logischem Namen und dem Pfadnamen einer Datei ist nur verfügbar, wenn die Datei Bestandteil eines Produktes ist, das mit IMON installiert wurde. Die Zuordnung kann von der Systembetreuung auch mit dem Kommando SET-INSTALLATION-PATH in das SCI eingetragen werden. Nähere Einzelheiten siehe Handbuch „IMON“ [12].

Im Operanden DEFAULT-PATH-NAME muss ein Ausdruck angegeben werden, der ersatzweise als Ergebnis zurückgegeben wird, wenn kein zugeordneter Pfadname existiert (das Produkt ist nicht im SCI registriert oder zu dem angegebenen logischen Namen existiert keine Datei).

Format

INSTALLATION-PATH( )
LOGICAL-ID = string_ausdruck ,INSTALLATION-UNIT = string_ausdruck ,VERSION = *STD / string_ausdruck ,DEFAULT-PATH-NAME = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

LOGICAL-ID = string_ausdruck
Bezeichnet den logischen Namen der Datei (Installation-Item), deren Pfadname ausgegeben werden soll (z.B. SYSPRG).

INSTALLATION-UNIT = string_ausdruck
Bezeichnet den Produktnamen (Name der Installation-Unit).

VERSION = *STD / string_ausdruck
Bezeichnet die Produktversion (bis zu 8 Zeichen).
Explizit kann eine Version im Format *[V][m]m.naso* angegeben werden (siehe auch SDF-Datentyp composed-name). Ist die angegebene Version nicht im SCI registriert, wird der Funktionsaufruf ohne Rückgabe eines Ergebnisses (auch keine Ersatzzeichenfolge) abgebrochen.

DEFAULT-PATH-NAME = string_ausdruck
Legt die Ersatzzeichenfolge (z.B. einen Pfadnamen, der mit Sicherheit exisiert) fest, die ausgegeben wird, wenn kein zugeordneter Pfadname verfügbar ist (Produkt oder Installations-Item nicht im SCI registriert).

Ergebnis

String gemäß den Richtlinien für den SDF-Datentyp <filename 1..54>.

Fehlermeldung

```
SDP0469   DER ANGEGEBENE PARAMETER '(&00)' IST UNGUELTIG

SDP0470   INTERNER FEHLER BEIM GETINSP-/GETSINV-SCHNITTSTELLENAUFRUF.
          RETURNCODE '(&00)'

SDP0489   WARNUNG: INSTALLATIONS-UNIT '(&00)' NICHT GEFUNDEN IM IMON-
          SOFTWARE-INVENTORY. STANDARDWERT WIRD BENUTZT

SDP0490   INSTALLATIONS-UNIT '(&00)', VERSION '(&01)' NICHT GEFUNDEN

SDP0491   WARNUNG: LOGICAL-ID '(&00)' NICHT GEFUNDEN IM INSTALLATION-UNIT
          '(&01)' , VERSION '(&02)' . STANDARD-WERT WIRD BENUTZT
```

Beispiele

```
/A = INSTALLATION-PATH(LOGICAL-ID='SYSLNK',INSTALLATION-UNIT='EDT',
DEFAULT-PATH-NAME='*** kein Pfadname vorhanden! ***')
/SHOW-VARIABLE A
A = :2OSH:$TSOS.SYSLNK.EDT.166
```

Ausgegeben wird der Pfadname der Ladebibliothek des Produkts EDT.

```
/A = INSTALLATION-PATH(LOGICAL-ID='SYSRME.D',INSTALLATION-UNIT='EDT',
DEFAULT-PATH-NAME='*** keine Readme-Datei vorhanden! ***')
%  SDP0491 Warning: Logical-id 'SYSRME.D' not found in Installation-Unit
'EDT' version '*STD'. Default value assumed
/SHOW-VARIABLE A
A = *** keine Readme-Datei vorhanden! ***
```

Ausgegeben wird die in DEFAULT-PATH-NAME vereinbarte Ersatzzeichenfolge, da für die aktuelle EDT-Version keine Readme-Datei existiert.

```
/B = INSTALLATION-PATH(LOGICAL-ID='SYSLNK',INSTALLATION-UNIT='EDT',
VERSION='16.0',DEFAULT-PATH-NAME='*** kein Pfadname vorhanden! ***')
%  SDP0490 Installation-Unit 'EDT' version '16.0' not found
%  SDP0431 ERROR 'SDP0490' IN BUILTIN FUNCTION 'INSTALLATION-PATH'
%  SDP0239 ERROR DURING EVALUATION OF RIGHT SIDE OF ASSIGNMENT
/SHOW-VARIABLE B
%  SDP1008 VARIABLE/LAYOUT 'B' DOES NOT EXIST
%  SDP0234 OPERAND 'NAME' INVALID
```

Der Funktionsaufruf wird abgebrochen, da zwar das Produkt EDT installiert ist (s.o), aber die explizit angegebene Version V16.0 nicht existiert. Die Variable B wird nicht versorgt (bzw. implizit angelegt), da auch kein Wert zurückgeliefert wird.

```
/C = INSTALLATION-PATH(LOGICAL-ID='SYSPRC',INSTALLATION-UNIT='USER-TOOLS',
DEFAULT-PATH-NAME='$RZTOOLS.SYSPRC.USER-TOOLS.010')
/SHOW-VARIABLE C
C = $RZTOOLS.SYSPRC.USER-TOOLS.010
```

Ausgegeben wird die in DEFAULT-PATH-NAME vereinbarte Ersatzzeichenfolge (in diesem Fall ein Pfadname), da ein Produkt mit dem Namen USER-TOOLS nicht im SCI registriert ist.

INTEGER( ) Ausdruck in Integer-Wert konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion INTEGER( ) konvertiert einen beliebigen Ausdruck in den Datentyp INTEGER. Dabei werden Ausdrücke vom Typ STRING entsprechend den Regeln der impliziten Konvertierung umgesetzt. Für Ausdrücke vom Typ BOOLEAN gilt: TRUE wird zum Wert 1 konvertiert, FALSE zu 0.

Ob ein Ausdruck vom Typ STRING konvertierbar ist, kann vorab mit der Funktion IS-INTEGER( ) geprüft werden.

Format

INTEGER( ) INT( )
EXPRESSION = ausdruck

Ergebnistyp

INTEGER

Eingabeparameter

EXPRESSION = ausdruck
„ausdruck“ ist ein Ausdruck vom Typ STRING, INTEGER oder BOOLEAN.

Ergebnis

Zahl vom Typ INTEGER

Fehlermeldung

```
SDP0415   SYNTAX-FEHLER: ZAHL IM STRING ERWARTET. KONVERTIERUNG
          NICHT MOEGLICH
```

Beispiel

Die Variablen A, B, C und D werden initialisiert:

```
/A = '4'
/B = 5
/C = 30
/D = TRUE    "Typ: BOOLEAN"

/AINT = INTEGER(EXPRESSION = A)
/SHOW-VARIABLE AINT
AINT = 4

/BINT = INTEGER(EXPRESSION = B + C)
/SHOW-VARIABLE BINT
BINT = 35

/CINT = INTEGER(EXPRESSION = D)
/SHOW-VARIABLE CINT
CINT = 1
```

INTEGER-TO-CHARACTER( ) Zahl in Zeichen konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion INTEGER-TO-CHARACTER( ) konvertiert eine Zahl in ein Zeichen(C-String).

Die Zahl wird als Integer-Wert des EBCDI-Codes eines Zeichens interpretiert. Das so codierte Zeichen wird als Ergebnis zurückgegeben.

INTEGER-TO-CHARACTER( ) ist die Umkehrung der Funktion CHARACTER-TO-INTEGER( ).

Format

INTEGER-TO-CHARACTER( ) INT-TO-CHAR( )
INTEGER = arithm_ausdruck

Ergebnistyp

STRING (<string 1..1>)

Eingabeparameter

INTEGER = arithm_ausdruck
Bezeichnet die zu konvertierende Zahl, wobei gilt 0 ≤ zahl ≤ 255.

Ergebnis

Zeichen des EBCDI-Codes als String

Fehlermeldung

```
SDP0416   ZAHL AUSSERHALB ZULAESSIGEN WERTEBEREICHS
```

Beispiel

```
/B = INTEGER-TO-CHARACTER(INTEGER = 129)
/SHOW-VARIABLE B
B = a

/C = INTEGER-TO-CHARACTER(INTEGER = 193 + 16)
/SHOW-VARIABLE C
C = J
```

INTEGER-TO-X-LITERAL( ) Zahl in X-Literal konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion INTEGER-TO-X-LITERAL( ) konvertiert eine Dezimalzahl in ein X-Literal, das die 4 Byte lange Codierung der Zahl enthält.

INTEGER-TO-X-LITERAL( ) ist die inverse Funktion zu X-LITERAL-TO-INTEGER( ).

Format

INTEGER-TO-X-LITERAL( ) INT-TO-X-LIT( )
STRING = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den maximal 4 Byte langen String, der konvertiert wird.

Ergebnis

String, der ein X-Literal enthält

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/DECLARE-VARIABLE A( TYPE= *STRING )
/DECLARE-VARIABLE B( TYPE= *INTEGER )
/A = INT-TO-X-LIT( -235736076 )
/SHOW-VARIABLE A
A = X'F1F2F3F4'

/B = X-LIT-TO-INT(&(A)) &* die &-Ersetzung beachten
/SHOW-VARIABLE B
B = -235736076

/B = X-LIT-TO-INT('1234')
/SHOW-VARIABLE B
B = -235736076

/A = INT-TO-X-LIT( 0 )
/SHOW-VARIABLE A
A = X'00000000'
```

IS-C-LITERAL( ) C-Literal prüfen

Anwendungsgebiet: **Stringfunktionen/Prüffunktionen**

Die Funktion IS-C-LITERAL( ) prüft, ob der angegebene String ein C-Literal ist und in einen String konvertiert werden kann (mit der Funktion FROM-C-LITERAL).

Format

IS-C-LITERAL( ) IS-C-LIT( )
STRING = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den String-Ausdruck, dessen Inhalt geprüft wird.

Ergebnis

TRUE
„string_ausdruck“ enthält ein C-Literal und kann z. B. mit der Funktion FROM-C-LITERAL in einen String konvertiert werden.

FALSE
„string_ausdruck“ enthält kein C-Literal.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = 'C''abc'''
/D = IS-C-LITERAL(STRING = A)
/SHOW-VARIABLE D
D = TRUE

/B = C'abc'
/D = IS-C-LITERAL(STRING = B)
/SHOW-VARIABLE D
D = FALSE

/C = '''abc'''
/D = IS-C-LITERAL(STRING = C)
/SHOW-VARIABLE D
D = TRUE
```

IS-CATALOGED-FILE( ) Katalogeintrag prüfen

Anwendungsgebiet: **Datei-Informationen**

Die Funktion IS-CATALOGED-FILE( ) prüft, ob es einen Katalogeintrag mit dem angegebenen Dateinamen gibt. Die Reaktion im Fehlerfall (ungültiger Dateiname usw.) kann vereinbart werden.

Format

IS-CATALOGED-FILE( ) IS-CAT-FILE( )
FILE = string_ausdruck ,ERROR-REPORTING = *PROC-ERROR-MECHANISM / *RETURN-FALSE ,ERROR-VARIABLE = *NONE / string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

FILE = string_ausdruck
Bezeichnet eine Datei, muss also dem SDF-Datentyp <filename 1...54 without-gen-vers> entsprechen.

Enthält der String Katalog- und Benutzerkennung, wird der Katalogeintrag im Benutzerkatalog der angegebenen Benutzerkennung gesucht, und zwar auf dem Pubset mit der angegebenen Katalogkennung.

Enthält der String eine Benutzerkennung, jedoch keine Katalogkennung, wird der Katalogeintrag im Benutzerkatalog der angegebenen Benutzerkennung auf dem Pubset gesucht, der der Benutzerkennung als Default-Pubset zugewiesen ist.

Enthält der String keine Benutzerkennung, wird die Benutzerkennung des laufenden Auftrags, das heißt die Benutzerkennung des SET-LOGON-PARAMETERS-Kommandos, eingesetzt. Dann wird, abhängig davon, ob eine Katalogkennung angegeben wurde, der Benutzerkatalog auf dem Default-Pubset oder dem angegebenen Pubset durchsucht.

ERROR-REPORTING =
Es kann vereinbart werden, ob im Fehlerfall die Fehlerbehandlung ausgelöst oder ob der Meldungsschlüssel der Fehlermeldung in einer S-Variablen abgelegt wird.

ERROR-REPORTING = *PROC-ERROR-MECHANISM
Im Fehlerfall wird die Fehlerbehandlung ausgelöst, siehe Abschnitt „Fehlerbehandlung“ auf Seite 43.

ERROR-REPORTING = *RETURN-FALSE
Im Fehlerfall wird das Ergebnis *FALSE* ausgegeben. Eine Fehlermeldung wird nicht ausgegeben. In die bei ERROR-VARIABLE = ... angegebene Variable wird der Meldungsschlüssel der Fehlermeldung geschrieben.

ERROR-VARIABLE =
Es kann eine S-Variable für den Meldungsschlüssel vereinbart werden. Der Meldungsschlüssel wird nur dann in die Variable geschrieben, wenn im Funktionsaufruf ERROR-REPORTING = *RETURN-FALSE angegeben wurde.

ERROR-VARIABLE = *NONE
Es wird keine S-Variable vereinbart.

ERROR-VARIABLE = string_ausdruck
Name der S-Variablen, in die der Meldungschlüssel der Fehlermeldung geschrieben wird. Es ist Folgendes zu beachten:
– Wird der Variablenname direkt angegeben, muss er in Hochkommas eingeschlossen sein. Anderenfalls wird der Inhalt der Variablen als Variablenname interpretiert.
– In die S-Variable wird nur geschrieben, wenn ein Fehler auftritt (Ergebnis=FALSE) und ERROR-REPORTING=*RETURN-FALSE angegeben wurde. Beispiel für mögliche Meldungsschlüssel: SDP0439, SDP0440 oder DMSxxxx.
– Tritt beim Schreiben in die S-Variable ein Fehler auf, werden die entsprechenden Fehlermeldungen unabhängig von der Angabe in ERROR-REPORTING nach SYSOUT ausgegeben und die S-Variable enthält keinen Rückgabewert.

Ergebnis

TRUE
Die im Parameter FILE bezeichnete Datei ist katalogisiert.

FALSE
Die im Parameter FILE bezeichnete Datei ist nicht katalogisiert oder beim Aufruf mit ERROR-REPORTING=*RETURN-FALSE ist ein Fehler aufgetreten.

Fehlermeldung

```
SDP0439   LAENGE DES DATEI-NAMENS NULL ODER GROESSER ALS 54

SDP0440   DNAME '(&00)' KEIN DATEINAME ODER KEIN SPEZIFISCHER DATEINAME

SDP0441   DVS-FEHLER '(&00)' BEIM AUFRUF DES FSTAT-MAKROS.
          IM SYSTEM-MODUS: /HELP-MSG (&00)
```

Beispiel 1

Eine Banddatei BAND.A soll gelesen werden. Zuvor muss geprüft werden, ob BAND.A schon katalogisiert ist oder ob BAND.A vor dem Importieren katalogisiert werden muss.

```
/IF (NOT IS-CATALOGED-FILE(FILE = 'BAND.A'))
/IMPORT-FILE SUPPORT=*TAPE(FILE-NAME=BAND.A,DEVICE-TYPE=...,VOLUME=...)
/END-IF
```

Beispiel 2

```
/DECLARE-VARIABLE NAME=(A(TYPE=*BOOL),B(TYPE=*STRING),C(TYPE=*BOOL))
/A = IS-CATALOGED-FILE(FILE='A_A',ERROR-REPORTING=*RETURN-FALSE,-
/                                ERROR-VARIABLE='B')
/C = IS-CATALOGED-FILE(FILE='A_A')
/ . . .
/SET-JOB-STEP
/SHOW-VARIABLE SELECT=*BY-ATTRIBUTES(INITIALIZATION=*ANY)
```

Ablaufprotokoll

```
% 1  1 /DECLARE-VARIABLE NAME=(A(TYPE=*BOOL), B(TYPE=*STRING), C(TYPE=*BOOL))
% 2  1 /A = IS-CATALOGED-FILE(FILE='A_A',ERROR-REPORTING=*RETURN-FALSE,ERROR-
VARIABLE='B')
% 3  1 /C = IS-CATALOGED-FILE(FILE='A_A')
%  SDP0440 NAME 'A_A' NOT A FILE NAME OR NOT A SPECIFIC FILE NAME
%  SDP0431 ERROR 'SDP0440' IN BUILTIN FUNCTION 'IS-CATALOGED-FILE'
%  SDP0239 ERROR DURING EVALUATION OF RIGHT SIDE OF ASSIGNMENT
%  SDP0004 ERROR DETECTED AT COMMAND LINE: 3 IN PROCEDURE ':R:$CSLTOM.PROC'
% 5  1 /SET-JOB-STEP
% 6  1 /SHOW-VARIABLE SELECT=*BY-ATTR(INIT=*ANY)
A = FALSE
B = SDP0440
C = *NO-INIT
*END-OF-CMD
```

IS-CATALOGED-JV( ) Jobvariable abfragen

Anwendungsgebiet: **Jobvariablen**

Die Funktion IS-CATALOGED-JV( ) prüft, ob es einen Katalogeintrag für den angegebenen Jobvariablennamen gibt, das heißt, ob die angegebene Jobvariable existiert.

Die Funktion setzt voraus, dass das Subsystem JV geladen ist. Nähere Informationen über Jobvariablen enthält das Handbuch „Jobvariablen“ [5].

Format

IS-CATALOGED-JV( ) IS-CAT-JV( )
JV = string_ausdruck ,ERROR-REPORTING = *PROC-ERROR-MECHANISM / *RETURN-FALSE ,ERROR-VARIABLE = *NONE / string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

JV = string_ausdruck
Bezeichnet eine Jobvariable, muss also dem SDF-Datentyp <filename 1...54 without-gen-vers> entsprechen.

ERROR-REPORTING =
Es kann vereinbart werden, ob im Fehlerfall die Fehlerbehandlung ausgelöst oder ob der Meldungsschlüssel der Fehlermeldung in einer S-Variablen abgelegt wird.

ERROR-REPORTING = ***PROC-ERROR-MECHANISM**
Im Fehlerfall wird die Fehlerbehandlung ausgelöst, siehe Abschnitt „Fehlerbehandlung“ auf Seite 43.

ERROR-REPORTING = *RETURN-FALSE
Im Fehlerfall wird das Ergebnis *FALSE* ausgegeben. Eine Fehlermeldung wird nicht ausgegeben. In die bei ERROR-VARIABLE = ... angegebene Variable wird der Meldungsschlüssel der Fehlermeldung geschrieben.

ERROR-VARIABLE =
Es kann eine S-Variable für den Meldungsschlüssel vereinbart werden. Der Meldungsschlüssel wird nur dann in die Variable geschrieben, wenn im Funktionsaufruf ERROR-REPORTING = *RETURN-FALSE angegeben wurde.

ERROR-VARIABLE = *NONE
Es wird keine S-Variable vereinbart.

ERROR-VARIABLE = string_ausdruck
Name der S-Variablen, in die der Meldungschlüssel der Fehlermeldung geschrieben wird. Es ist Folgendes zu beachten:
– Wird der Variablenname direkt angeben, muss er in Hochkommas eingeschlossen sein. Anderenfalls wird der Inhalt der Variablen als Variablenname interpretiert.
– In die S-Variable wird nur geschrieben, wenn ein Fehler auftritt (Ergebnis=FALSE) und ERROR-REPORTING=*RETURN-FALSE angegeben wurde. Beispiel für mögliche Meldungsschlüssel: SDP0439, SDP0440 oder DMSxxxx.
– Tritt beim Schreiben in die S-Variable ein Fehler auf, werden die entsprechenden Fehlermeldungen unabhängig von der Angabe in ERROR-REPORTING nach SYSOUT ausgegeben und die S-Variable enthält keinen Rückgabewert.

Ergebnis

TRUE
Die im Parameter JV bezeichnete Jobvariable ist katalogisiert.

FALSE
Die im Parameter JV bezeichnete Jobvariable ist nicht katalogisiert oder beim Aufruf mit ERROR-REPORTING=*RETURN-FALSE ist ein Fehler aufgetreten.

Fehlermeldung

```
SDP0495   '(&00)' IST KEIN KORREKTER JV-NAME

SDP1054   JOBVARIABLEN-FEHLER: JVS-FEHLERCODE '(&00)' BEI ZUGRIFF AUF
          JOBVARIABLE '(&01)'. IN SYSTEM-MODUS: /HELP-MSG JVS(&00)
```

Beispiel

```
/IF (IS-CATALOGED-JV(JV='PS'))
/   WRITE-TEXT 'VORHANDEN'
/ELSE
/   WRITE-TEXT 'ERZEUGEN'
/END-IF
```

Ausgabe:
```
ERZEUGEN
```

IS-DECLARED( ) Variablendeklaration prüfen

Anwendungsgebiete: **Variablenzugriff/Prüffunktionen**

Die Funktion IS-DECLARED( ) prüft, ob die angegebene einfache oder zusammengesetzte Variable bereits deklariert ist.

Format

IS-DECLARED( )
VARIABLE-NAME = string_ausdruck ,SCOPE = *BY-HIERARCHY / *TASK / *CALLING-PROCEDURES

Ergebnistyp

BOOLEAN

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet eine Variable. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe dazu das Beispiel auf der folgenden Seite).

SCOPE =
Bezeichnet den Geltungsbereich, in dem die Variable gesucht wird.

SCOPE = *BY-HIERARCHY
Zuerst wird im INCLUDE-, dann im PROCEDURE- und schließlich im Task-Bereich gesucht. Task-Variable sind nur sichtbar, wenn sie importiert wurden.

SCOPE = *TASK
Es wird nur im Taskbereich gesucht.

SCOPE = *CALLING-PROCEDURES
Prüft, ob die angegebene Variable schon - und zwar mit IMPORT-ALLOWED = *YES - deklariert ist. Die Variable wird von der Aufrufer-Prozedur aufwärts zur Dialogebene (bei Vordergrund-Prozeduren) oder bis zur ersten Prozedur (in einer Hintergrund-Prozedur) gesucht. Wenn die gefundene Variable mit IMPORT-ALLOWED = *NO deklariert. ist, wird die Suche wieder aufgenommen, solange noch nicht der gesamte Bereich geprüft wurde. Wird bei dieser Prüfung irgendwann eine Variable mit dem angegebenen Namen gefunden, die mit IMPORT-ALLOWED = *YES deklariert wurde, wird TRUE zurückgegeben, ansonsten FALSE.

Ergebnis

TRUE
Die im Parameter VARIABLE-NAME bezeichnete Variable ist im angegebenen Geltungsbereich deklariert (bzw. in der gesuchten Weise deklariert).

FALSE
Die im Parameter VARIABLE-NAME bezeichnete Variable ist im angegebenen Geltungsbereich nicht deklariert (bzw. in der gesuchten Weise nicht deklariert).

Fehlermeldungen

```
SDP0010   TYP DES PARAMETERS '(&00)' FALSCH
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

In Prozedur „Proz1“ wird die Variable „A“ deklariert und die Prozedur „Proz2“ aufgerufen:

```
/DECLARE-VARIABLE VARIABLE-NAME=A(TYPE=*INTEGER,INITIAL-VALUE=12),-
/SCOPE=*PROCEDURE(IMPORT-ALLOWED=*YES)
/CALL-PROCEDURE Proz2
```

In Prozedur „Proz2“ ist Folgendes geschrieben:

```
/B=IS-DECLARED(VARIABLE-NAME='A',SCOPE=*CALLING-PROCEDURES)
/SHOW-VARIABLE VARIABLE-NAME=B
```

Ausgabe

```
B=TRUE
```

IS-EMPTY-FILE( ) Dateigröße prüfen

Anwendungsgebiet: **Umgebungsinformationen**

Die Funktion IS-EMPTY-FILE( ) prüft, ob die Datei leer ist (Last-Page-Pointer zeigt auf die Seite 0). Das ist der Fall, wenn in der Ausgabe des Kommandos SHOW-FILE-ATTRIBUTES das Ausgabefeld HIGH-US-PA den Wert 0 enthält.

Format

IS-EMPTY-FILE( )
FILE-NAME = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

FILE-NAME = string_ausdruck
Bezeichnet die zu überprüfende Datei.

Ergebnis

TRUE
Die Datei ist leer.

FALSE
Die Datei ist nicht leer.

Fehlermeldungen

```
SDP0093   FEHLER BEIM ZUGRIFF AUF DATEI/BIBLIOTHEK '(&00)', FEHLER '(&01)'.
          WEITERE INFORMATIONEN: /HELP-MSG (&01)

SDP0440   NAME '(&00)' KEIN DATEINAME ODER KEIN SPEZIFISCHER DATEINAME

SDP0453   PARAMETER (&00) IST LEER ODER LAENGER ALS (&01) ZEICHEN ODER
          ENTHAELT LEERZEICHEN
```

Beispiel

```
/CREATE-FILE newfile
/IF (IS-EMPTY-FILE ('newfile'))
/     WRITE-TEXT 'Diese Datei ist leer'
/ELSE
/     WRITE-TEXT 'Diese Datei ist nicht leer'
/END-IF
```

Ausgabe

```
Diese Datei ist leer
```

IS-INITIALIZED( ) Variableninitialisierung prüfen

Anwendungsgebiet: **Variablenzugriff/Prüffunktionen**

Die Funktion IS-INITIALIZED( ) prüft, ob die angegebene Variable initialisiert ist, das heißt, ob sie einen gültigen Inhalt hat. Auch der Leerstring ist ein gültiger Variableninhalt.

Es können nur einfache Variablen oder Listenvariablen geprüft werden.

Format

IS-INITIALIZED( )
VARIABLE-NAME = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet eine einfache Variable oder Listenvariable. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe dazu das nachfolgende Beispiel sowie das Beispiel bei IS-DECLARED( )).
Eine Listenvariable muss in der Form 'listenname#' bezeichnet werden. Einzelne Listenelemente können in der Form 'listenname#elementindex' angegeben werden.

Ergebnis

TRUE
Die mit dem Parameter VARIABLE-NAME bezeichnete Variable ist initialisiert.

FALSE
Die mit dem Parameter VARIABLE-NAME bezeichnete Variable ist nicht initialisiert.

Fehlermeldung

```
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

```
/DECLARE-VARIABLE X
/DECLARE-VARIABLE A
/A = 'ABC'

/B = IS-INITIALIZED(VARIABLE-NAME = 'A')
/SHOW-VARIABLE B
B = TRUE

/B = IS-INITIALIZED(VARIABLE-NAME = 'AA')
/SHOW-VARIABLE B
B = FALSE

/B = IS-INITIALIZED(VARIABLE-NAME = 'X')
/SHOW-VARIABLE B
B = FALSE

/FREE-VARIABLE NAME = A
/B = IS-INITIALIZED(VARIABLE-NAME = 'A')
/SHOW-VARIABLE B
B = FALSE
```

Die Variable A hat nach FREE-VARIABLE keinen Inhalt mehr, daher wird als Ergebnis FALSE geliefert.

IS-INTEGER( ) Ausdruck prüfen

Anwendungsgebiet: **Variablenzugriff / Prüffunktionen**

Die Funktion IS-INTEGER( ) prüft, ob der als String angegebene Ausdruck einen Integer-Wert darstellt:

– Der String darf nur aus den Ziffern 0 bis 9 bestehen sowie aus den Vorzeichen + und -.
– Das Vorzeichen + oder - muss direkt vor der Zahl stehen, das heißt, Vorzeichen und zugehörige Zahl dürfen nicht durch Leerzeichen getrennt werden.
– Der Wert der Zahl muss im gültigen Wertebereich von -2^{31} bis $2^{31}-1$ liegen.

Leerzeichen sind am Anfang und am Ende des Strings erlaubt, das heißt, der String darf links- und rechtsbündig mit Leerzeichen aufgefüllt sein. Wenn der geprüfte String einen Integer-Wert enthält, kann er z.B. anschließend mit der Funktion INTEGER( ) konvertiert werden.

Format

IS-INTEGER( )
STRING = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den String, der auf Integer-Inhalt überprüft wird.

Ergebnis

TRUE
Der String enthält einen Integer-Wert, das heißt, er kann in einen Integer-Wert konvertiert werden.

FALSE
Der String enthält keinen Integer-Wert.

Beispiel

```
/A = IS-INTEGER (STRING = '   -123')
/SHOW-VARIABLE A
A = TRUE

/B = IS-INTEGER(STRING = '+(123-3)')
/SHOW-VARIABLE B
B = FALSE
```

Im ersten Fall beinhaltet der String einen Integer-Wert. Die enthaltenen Leerzeichen sind erlaubt. Darum ist das Ergebnis: A = TRUE. Im zweiten Fall beinhaltet der String keinen Integer-Wert, sondern einen Ausdruck. Darum ist das Ergebnis: B = FALSE.

IS-LIBRARY( ) Bibliotheksname prüfen

Anwendungsgebiet: **Prüffunktionen**

Die Funktion IS-LIBRARY( ) prüft, ob die angegebene Datei im Katalog als PLAM-Bibliothek eingetragen ist.

Wenn die Datei im Katalog nicht als PLAM-Bibliothek eingetragen ist, liefert IS-LIBRARY( ) das Ergebnis FALSE.

Wenn die angegebene Datei nicht existiert, wird die Fehlerbehandlung angestoßen.

Nähere Informationen über das Arbeiten mit PLAM-Bibliotheken enthält das Handbuch „LMS (BS2000)“ [11].

Format

IS-LIBRARY( )
FILE = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

FILE = string_ausdruck
„string_ausdruck“ bezeichnet einen Dateinamen, entsprechend dem SDF-Datentyp <filename 1...54>.

Ergebnis

TRUE
Die mit dem Parameter FILE bezeichnete Datei ist als PLAM-Bibliothek im Katalog eingetragen.

FALSE
Die mit dem Parameter FILE bezeichnete Datei ist nicht als PLAM-Bibliothek im Katalog eingetragen.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = IS-LIBRARY('MY-LIBRARY')
/SHOW-VARIABLE A
/A = TRUE
/SHOW-FILE-ATTRIBUTES MY-LIBRARY,INF=ALL
```

Die Ausgabe des Kommandos SHOW-FILE-ATTRIBUTES zeigt den vollständigen Katalogeintrag für MY-LIBRARY. Das Ausgabefeld TYPE enthält den Wert PLAM-LIB:

```
%00000000012 :2OSG:$USER1.MY-LIBRARY
%  ------------------------------- HISTORY      ------------------------------
%  CRE-DATE   = 1996-06-03  ACC-DATE   = 2007-04-19  CHANG-DATE = 2006-06-04
%  CRE-TIME   =   13:41:01  ACC-TIME   =   10:51:10  CHANG-TIME =   12:34:10
%  ACC-COUNT  = 40          S-ALLO-NUM = 0
%  ------------------------------- SECURITY     ------------------------------
%  READ-PASS  = NONE        WRITE-PASS = NONE        EXEC-PASS  = NONE
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
%  OWNER      = R W X       GROUP      = R - X       OTHERS     = R - X
%  AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = 2009-10-06
%  DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME =   00:00:00
%  SP-REL-LOCK= NO          ENCRYPTION = *NONE
%  ------------------------------- BACKUP       ------------------------------
%  BACK-CLASS = A           SAVED-PAG  = COMPL-FILE  VERSION    = 2
%  MIGRATE    = ALLOWED
%  ------------------------------- ORGANIZATION ------------------------------
%  FILE-STRUC = PAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  TYPE       = PLAM-LIB
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *NONE       SO-MIGR    = *ALLOWED
%  ------------------------------- ALLOCATION   ------------------------------
%  SUPPORT    = PUB         S-ALLOC    = 24          HIGH-US-PA = 9
%  EXTENTS     VOLUME     DEVICE-TYPE      EXTENTS     VOLUME     DEVICE-TYPE
%      1       GVS2.1       D3435
%  NUM-OF-EXT = 1
%:2OSG: PUBLIC:      1 FILE  RES=        12 FRE=        3 REL=        3 PAGES
```

IS-LIBRARY-ELEMENT( ) Bibliothekselement prüfen

Anwendungsgebiet: **Prüffunktionen**

Die Funktion IS-LIBRARY-ELEMENT( ) prüft, ob das angegebene Bibliothekselement existiert oder nicht.

Format

IS-LIBRARY-ELEMENT( ) IS-LIB-ELEM( )
LIBRARY = string_ausdruck ,ELEMENT = string_ausdruck ,TYPE = string_ausdruck ,VERSION = *HIGHEST-EXISTING / string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

LIBRARY = string_ausdruck
„string_ausdruck“ bezeichnet einen Dateinamen, entsprechend dem SDF-Datentyp <filename 1...54>.

ELEMENT = string_ausdruck
„string_ausdruck“ bezeichnet ein Bibliothekselement, entsprechend dem SDF-Datentyp <composed-name 1...64>.

TYPE = string_ausdruck
„string_ausdruck“ bezeichnet einen Bibliothekselement-Typ, entsprechend dem SDF-Datentyp <alphanum-name 1...8>.

VERSION = *HIGHEST-EXISTING / string_ausdruck
Bezeichnet ein Bibliothekselement-Version, entsprechend dem SDF-Datentyp <composed-name 1...24>.

Ergebnis

TRUE
Das angegebene Bibliothekselement ist vorhanden.

FALSE
Das angegebene Bibliothekselement ist nicht vorhanden.

Fehlermeldung

```
SDP0093   FEHLER BEIM ZUGRIFF AUF DATEI/BIBLIOTHEK '(&00)', FEHLER '(&01)'.
          WEITERE INFORMATIONEN: /HELP-MSG (&01)

SDP0454   UNGUELTIGER PARAMETER: '(&00)'
```

IS-SDF-LIST( ) String auf Kriterien für SDF-Liste analysieren

Anwendungsgebiet: **Stringfunktionen / Prüffunktionen**

Die Funktion IS-SDF-LIST( ) analysiert, ob ein String-Ausdruck eine SDF-Liste der Form '(<element$_1$>,<element$_2$>,...,<element$_n$>)' ist . Dabei sind <element$_i$> Folgen von Zeichen, die kein Komma (außerhalb von Klammerpaaren) enthalten.

Format

IS-SDF-LIST( )
STRING = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

STRING = string_ausdruck
Name des Strings, der analysiert werden soll.

Ergebnis

TRUE
Der String-Ausdruck ist eine SDF-Liste.

FALSE
Der String-Ausdruck ist keine SDF-Liste.

Fehlermeldung

Keine

Beispiel

```
/A=IS-SDF-LIST('(val1,val2)')
SHOW-VAR A
A = TRUE
/A=IS-SDF-LIST('val')
/SHOW-VAR A
A = FALSE
```

IS-SDF-P( ) Prüfen, ob SDF-P geladen ist

Anwendungsgebiet: **Prüffunktionen**

Die Funktion IS-SDF-P( ) prüft, ob SDF-P im System geladen ist. Wenn es geladen ist, wird als Ergebnis TRUE zurückgegeben. Das Ergebnis FALSE wird in folgenden Fällen zurückgegeben:
– SDF-P ist nicht geladen.
– SDF-P ist geladen, aber in der Task wird gerade die SDF-P-BASYS-Funktionalität simuliert (Einstellung FUNCTIONALITY=*BASIC im Kommando MODIFY-PROCEDURE-TEST-OPTIONS, siehe Seite 707).

Format

IS-SDF-P( )

Ergebnistyp

BOOLEAN

Eingabeparameter

Keine

Ergebnis

TRUE
SDF-P ist im System geladen.

FALSE
SDF-P ist nicht im System geladen oder in der Task wird die SDF-P-BASYS-Funktionalität simuliert.

Fehlermeldung

Keine

Beispiel

```
/A=IS-SDF-P()
/SHOW-VARIABLE A
A = TRUE
```

Anwendungsfall: Eine Prozedur soll auch ablauffähig sein, wenn nur die SDF-P-BASYS-Funktionalität verfügbar ist (z.B. können mit SDF-P Prozedurparameter mit dem Kommando READ-VARIABLE eingelesen und weitere Überprüfungen bzw. Korrekturen der Eingabe durchgeführt werden; ohne SDF-P kann der Parameter nur durch Prompting eingegeben werden):

```
/SET-PROCEDURE-OPTIONS
/DECLARE-PARAMETER A(INIT-VALUE=*PROMPT)
/...
/IF (IS-SDF-P())
/
/ "LIES DIE VARIABLE MIT HILFE-TEXT UND UEBERPRUEFE DAS ERGEBNIS"
/  IF (TASK-MODE() == 'DIALOG')
/ "LIES-ERNEUT:"
/     READ-VARIABLE A,INPUT=*TERMINAL(PROMPT='BITTE GEBEN SIE DATEINAME EIN')
/      IF (NOT CHECK-DATA-TYPE (A,*FULL-FILENAME))
/            WRITE-TEXT 'FEHLER: &A IST KEIN FILENAME'
/            GOTO LIES-ERNEUT
/      END-IF
/  ELSE
/      IF (NOT CHECK-DATA-TYPE (A,*FULL-FILENAME))
/          WRITE-TEXT 'FEHLER: &A IST KEIN FILENAME'
/          EXIT-PROCEDURE
/      END-IF
/  END-IF
/  "HIER KOENNEN NOCH ANDERE UEBERPRUEFUNGEN GEMACHT WERDEN"
/  ....
/ELSE
/          "GRUNDLEGENDE VERARBEITUNGEN MIT SDF-P-BASYS"
/          WRITE-TEXT 'BITTE GEBEN SIE DATEINAME EIN:'
/          REMARK &A
/END-IF
/
START-LMS
//OPEN &A,MODE=*READ
//..
```

IS-SDF-STRUCTURE( ) String auf Kriterien für SDF-Struktur analysieren

Anwendungsgebiet: **Stringfunktionen / Prüffunktionen**

Die Funktion IS-SDF-STRUCTURE( ) analysiert, ob der angegebene String eine SDF-Struktur ist.

Die angegebene SDF-Struktur muss mit einem Wert eingeleitet werden. Dieser Wert kann nicht weggelassen werden, ansonsten wird der String als SDF-Liste und nicht als SDF-Struktur betrachtet.

Format

IS-SDF-STRUCTURE( )
STRING = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

STRING = string_ausdruck
Name des Strings, der analysiert werden soll.

Ergebnis

TRUE
Der angegebene String ist eine SDF-Struktur.

FALSE
Der angegebene String ist keine SDF-Struktur.

Fehlermeldung

Keine

Beispiel

```
/A=IS-SDF-STRUCTURE('*P(val1,val2)')
/SHOW-VAR A
A = TRUE

/A=IS-SDF-STRUCTURE('(val1,val2)')
/SHOW-VAR A
A = FALSE
```

In diesem Fall wird der String als SDF-Liste betrachtet.

```
/A=IS-SDF-STRUCTURE('val')
/SHOW-VAR A
A = FALSE
```

IS-VARIABLE-NAME( ) Variablennamen prüfen

Anwendungsgebiet: **Stringfunktionen/Prüffunktionen**

Die Funktion IS-VARIABLE-NAME( ) prüft, ob der angegebene String ein syntaktisch korrekter Variablenname ist. Diese Prüfung ist eine reine Syntaxprüfung. Es wird nicht geprüft, ob es eine Variable dieses Namens gibt.

Format

IS-VARIABLE-NAME( ) IS-VAR-NAME( )
STRING = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

NAME = string_ausdruck
Bezeichnet den zu prüfenden String.

Ergebnis

TRUE
„string_ausdruck“ ist ein gültiger Variablenname.

FALSE
„string_ausdruck“ erfüllt nicht die Syntaxregeln für Variablennamen, ist also kein gültiger Variablenname.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/DECLARE-VARIABLE A
/A = '1234'
/B = IS-VARIABLE-NAME(STRING = A)
/SHOW-VARIABLE B
B = FALSE
```

Die Variable A wird deklariert und ihr wird der Wert '1234' zugewiesen. IS-VARIABLE-NAME( ) prüft jetzt, ob die Variable A einen gültigen Variablennamen enthält: '1234' ist kein gültiger Variablenname, da Variablennamen nicht mit einer Ziffer beginnen dürfen.

```
/B = IS-VARIABLE-NAME(STRING = 'A')
/SHOW-VARIABLE B
B = TRUE
```

IS-VARIABLE-NAME( ) prüft den String 'A': A ist ein gültiger Variablenname.

IS-X-LITERAL( ) X-Literal prüfen

Anwendungsgebiet: **Stringfunktionen/Prüffunktionen**

Die Funktion IS-X-LITERAL( ) prüft, ob der angegebene String-Ausdruck ein X-Literal enthält und mit FROM-X-LITERAL( ) konvertiert werden kann.

Format

IS-X-LITERAL( ) IS-X-LIT( )
STRING = string_ausdruck

Ergebnistyp

BOOLEAN

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den zu prüfenden String-Ausdruck.

Ergebnis

TRUE
„string_ausdruck“ enthält ein X-Literal.

FALSE
„string_ausdruck“ enthält kein X-Literal.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = 'X''01FF'''
/B = X'01FF'
/C = IS-X-LITERAL(STRING = A)
/SHOW-VARIABLE C
C = TRUE
/C = IS-X-LITERAL(STRING = B)
/SHOW-VARIABLE C
C = FALSE
```

Im ersten Fall (Variable A) enthält der zu prüfende String ein X-Literal, im zweiten Fall (Variable B) nicht. Denn der Variablenwert ist intern 01FF und damit kein X-Literal.

JOB-CLASS( ) Jobklasse abfragen

Anwendungsgebiet: **Job-Informationen**

Die Funktion JOB-CLASS( ) fragt ab, zu welcher Jobklasse die aktuelle Task gehört.

Format

JOB-CLASS( )

Ergebnistyp

STRING

Eingabeparameter

Keine

Ergebnis

Jobklasse als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = JOB-CLASS( )
/SHOW-VARIABLE A
A = JCDSTD
```

Zum Vergleich die Jobklasse im Feld JCLASS (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:     29XX       TYPE:    3 DIALOG   NOW:     2007-04-26.110747
%JOBNAME:            PRI:     0 210
%USERID:  USER1      JCLASS:  JCDSTD     LOGON:   2007-04-26.1053
%ACCNB:   89001      CPU-MAX:   9999     CPU-USED:000000.6447
%STATION: $$$06580   PROC:    FIREBALL
%TID:     000101AB   UNP/Q#:    00/000
%CMD:     SHOW-JOB-STATUS
%MONJV:   *NONE
```

JOB-MONJV( ) MONJV abfragen

Anwendungsgebiet: **Jobvariablen-Funktionen**

Die Funktion JOB-MONJV( ) liefert den Namen der Jobvariablen, die den Auftrag überwacht.

Die Funktion setzt voraus, dass das Subsystem JV geladen ist. Nähere Informationen über Jobvariablen enthält das Handbuch „Jobvariablen“ [5].

Format

JOB-MONJV( )

Ergebnistyp

STRING

Eingabeparameter

Keine

Ergebnis

Name der Jobvariablen

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

JOB-NAME( ) Jobnamen abfragen

Anwendungsgebiet: **Job-Informationen**

Die Funktion JOB-NAME( ) liefert den Jobnamen der aktuellen Task, das heißt, den Namen, der im Kommando SET-LOGON-PARAMETERS angegeben wurde.

Format

JOB-NAME( )

Ergebnistyp

STRING (<string 1..8>)

Eingabeparameter

Keine

Ergebnis

Jobname, wie im Kommando SET-LOGON-PARAMETERS angegeben.

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/SET-LOGON-PARAMETERS USER1,ACC01,'PASSWORD',JOB-NAME=BERTA
/ ...
/B = JOB-NAME( )
/SHOW-VARIABLE B
B = BERTA
```

Zum Vergleich (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:      29XX        TYPE:     3 DIALOG   NOW:      2007-04-26.110747
%JOBNAME: BERTA        PRI:      0 210
%USERID:  USER1        JCLASS:   JCDSTD     LOGON:    2007-04-26.1053
%ACCNB:   ACC01        CPU-MAX:    9999     CPU-USED:000000.6447
%STATION: $$$06580     PROC:     FIREBALL
%TID:     000101AB     UNP/Q#:     00/000
%CMD:     SHOW-JOB-STATUS
%MONJV:   *NONE
```

Das Feld JOBNAME zeigt den Namen BERTA, der im Kommando SET-LOGON-PARAMETERS angegeben wurde.

JV( ) Jobvariable abfragen

Anwendungsgebiet: **Jobvariablen-Funktionen**

Die Funktion JV( ) liefert den Inhalt der angegebenen Jobvariablen.

Die Funktion setzt voraus, dass das Subsystem JV geladen ist. Nähere Informationen über Jobvariablen enthält das Handbuch „Jobvariablen“ [5].

Format

JV( )
JV-NAME = string_ausdruck ,START = 1 / arithm_ausdruck$_1$,LENGTH = *REST-LENGTH / arithm_ausdruck$_2$

Ergebnistyp

STRING

Eingabeparameter

JV-NAME = string_ausdruck
Bezeichnet eine Jobvariable, „string_ausdruck“ muss ein gültiger Jobvariablenname oder ein durch einen vorangestellten * gekennzeichneter JV-Kettungsname sein.

START= 1 / arithm_ausdruck1
Bezeichnet die Startposition für den zu extrahierenden JV-Inhalt. Im Standardfall ist dies das erste Zeichen. arithm_ausdruck$_1$ muss ein positiver Integerwert sein, der kleiner ist als die Länge der gesamten JV. Wenn für arithm_ausdruck$_1$ kein gültiger Wert angegeben wird, wird der Nullstring zurückgegeben.

LENGTH = *REST-LENGTH / arithm_ausdruck2
Bezeichnet die Länge des zu extrahierenden JV-Inhalts. Der Defaultwert *REST-LENGTH zeigt an, das der zu extrahierende JV-Inhalt mit der bei START angegebenen Position beginnt und bis zum Ende reicht. Wird mit arithm_ausdruck$_2$ eine davon abweichende Länge angegeben, die zu lang ist, wird implizit LENGTH = *REST-LENGTH angenommen.

Ergebnis

Inhalt der mit „string_ausdruck“ bezeichneten Jobvariablen bzw. des durch „arithm_ausdruck$_1$“ und „arithm_ausdruck$_2$“ bezeichneten Teilbereichs.

Fehlermeldungen

SDP0412 START-POSITION AUSSERHALB MOEGLICHEN BEREICHS

SDP0414 WARNUNG: *REST-LENGTH WERTE FUER OPERAND LENGTH VERWENDET

SDP1022 AUF DIE JOBVARIABLE '(&00)' KANN NICHT ZUGEGRIFFEN WERDEN

SDP1024 JOBVARIABLE '(&00)' EXISTIERT NICHT

SDP1027 WERT FUER JOBVARIABLE '(&00)' IST KEIN STRING

SDP1054 JOBVARIABLEN-FEHLER: JVS-FEHLERCODE '(&00)' BEI ZUGRIFF AUF JOBVARIABLE '(&01)'. IM SYSTEM-MODUS: /HELP-MSG JVS(&00)

Beispiel

```
/CREATE-JV JV-NAME=HUGO
/MODIFY-JV JV-CONTENTS=HUGO,SET-VALUE=c'schalter ein'
/A = JV('HUGO')
/SHOW-VARIABLE A
A = schalter ein

/B = JV('HUGO',4,3)
/SHOW-VARIABLE B
B = alt
```

LAYOUT-SCOPE( ) Layout-Geltungsbereich abfragen

Anwendungsgebiet: **Variablenzugriff (Variablenname)**

Die Funktion LAYOUT-SCOPE( ) gilt nur für Strukturlayouts und liefert deren Geltungsbereich (Scope). Der Geltungsbereich eines Strukturlayouts wird bei der Deklaration der Strukturelemente im Kommando BEGIN-STRUCTURE festgelegt.

Format

LAYOUT-SCOPE( )
LAYOUT-NAME = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

LAYOUT-NAME = string_ausdruck
Bezeichnet ein Strukturlayout.

Ergebnis

**TASK*
Das Layout ist mit dem Geltungsbereich TASK deklariert.

**PROCEDURE*
Das Layout ist mit dem Geltungsbereich PROCEDURE deklariert.

**INCLUDE*
Das Layout ist mit dem Geltungsbereich INCLUDE deklariert.

Fehlermeldungen

```
SDP0442   LAYOUT EXISTIERT NICHT
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

```
/BEGIN-STRUCTURE LAY1
/   DECLARE-ELEMENT ELEM1
/   DECLARE-ELEMENT ELEM2
/   DECLARE-ELEMENT ELEM3
/END-STRUCTURE
/A = LAYOUT-SCOPE(LAYOUT-NAME='LAY1')
/SHOW-VARIABLE A
A = *PROCEDURE
```

Im Strukturdeklarationsblock zwischen BEGIN-STRUCTURE und END-STRUCTURE wird das Strukturlayout LAY1 deklariert. Standardmäßig hat das Strukturlayout den Geltungsbereich PROCEDURE.

LENGTH( ) Stringlänge abfragen

Anwendungsgebiet: **String-Analyse**

Die Funktion LENGTH( ) liefert die Länge des angegebenen Strings.

Format

LENGTH( )
STRING = string_ausdruck

Ergebnistyp

INTEGER

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den String, dessen Länge abgefragt wird.

Ergebnis

Zahl vom Typ INTEGER

Fehlermeldung

```
SDP1010   VARIABLE '(&00)' HAT KEINEN WERT
```

Beispiel 1

```
/SET-VARIABLE A = 'ANNAMARIA'
/SET-VARIABLE B = LENGTH(A)
/SHOW-VARIABLE B
B = 9
```

Beispiel 2

```
/DECLARE-VARIABLE VARLIST,MULTIPLE-ELEMENTS = *LIST
/  VARLIST = *STRING-TO-VAR('(Terminal,Drucker,Tastatur,Prozessor)')
/SHOW-VARIABLE VARLIST
VARLIST(*LIST) = Terminal
VARLIST(*LIST) = Drucker
VARLIST(*LIST) = Tastatur
VARLIST(*LIST) = Prozessor
/FOR A = *LIST(VARLIST)
/     B = LENGTH(A)
/     SHOW-VARIABLE B
/END-FOR
```

Ausgabe:

```
B = 8
B = 7
B = 8
B = 9
```

Eine Listenvariable VARLIST wird mit Defaultwerten deklariert. Anschließend wird den ersten vier Elementen von VARLIST ein Wert, hier ein Wort, zugewiesen (hier werden die vier Werte in einer Zuweisung angegeben; siehe Kommando SET-VARIABLE, Seite 753). In der FOR-Schleife wird die Länge jedes Listenelements geprüft und ausgegeben.

LIMIT( ) Maximale Listengröße abfragen

Anwendungsgebiet: **Variablenzugriff (Variablenname)**

Die Funktion LIMIT( ) ist nur auf Listenvariable anwendbar. LIMIT( ) fragt ab, wie viele Elemente eine Listenvariable enthalten darf. Dieser Grenzwert wird bei der Deklaration der Listenvariablen mit dem Kommando DECLARE-VARIABLE im Operanden MULTIPLE-ELEMENTS = *LIST(LIMIT = zahl) festgelegt.

Format

LIMIT( )
LIST-NAME = string_ausdruck

Ergebnistyp

INTEGER (<integer 1 .. 2147483647>)

Eingabeparameter

LIST-NAME = string_ausdruck
Bezeichnet eine Liste.

Ergebnis

Anzahl zulässiger Elemente; wird als Integer-Wert zurückgegeben.

Fehlermeldungen

```
SDP0426   VARIABLE '(&00)' KEINE LISTE
SDP1007   NOCH KEINE VARIABLE ANGELEGT
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

```
/DECLARE-VARIABLE LIST3, MULT-ELEM = *LIST(LIMIT = 10)
/DECLARE-VARIABLE LIST4, MULT-ELEM = *LIST

/A = LIMIT('LIST3')
/SHOW-VAR A
A = 10

/A = LIMIT('LIST4')
/SHOW-VARIABLE A
A = 2147483647
```

Die Listenvariable LIST3 wurde mit der Angabe LIMIT=10 deklariert. Entsprechend liefert LIMIT( ) als Ergebnis den Wert 10.
Die Listenvariable LIST4 wurde mit Defaultwerten deklariert. LIMIT( ) liefert daher den Defaultwert für die maximale Listengröße (2^{31}-1).

LOGGING-MODE( ) Protokollierung prüfen

Anwendungsgebiet: **Prozedurinformationen**

Die Funktion LOGGING-MODE( ) zeigt an, ob beim Aufruf des Kommandos CALL-PROCEDURE oder INCLUDE-PROCEDURE Protokollieren eingeschaltet wurde.

Die Protokollierung des Prozedurablaufs wird beim Aufruf des Kommandos CALL- oder INCLUDE-PROCEDURE im Operanden LOGGING festgelegt. Dabei kann die Protokollierung für Kommandofolge und Datenfluss unabhängig voneinander eingestellt werden. Gilt für eines oder beide BY-PROC-TEST-OPTION, wird der aktuelle Protokollierungsstatus über das Kommando MODIFY-PROC-TEST-OPTIONS bestimmt.

Zu beachten ist, dass LOGGING-MODE( ) für Kommandos und Daten getrennt aufgerufen werden muss.

Format

LOGGING-MODE( ) LOG-MODE( )
STREAM = *CMD / *DATA

Ergebnistyp

STRING ('YES' / 'NO')

Eingabeparameter

STREAM =
Legt fest, welcher Protokolltyp abgefragt wird.

STREAM = *CMD
Fragt ab, ob die Kommandofolge protokolliert wird.

STREAM = *DATA
Fragt ab, ob der Datenfluss protokolliert wird.

Ergebnis

YES
Kommandos/Daten werden protokolliert.

NO
Kommandos/Daten werden nicht protokolliert.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/CALL-PROCEDURE PROC, LOGGING = *PAR(DATA = *BY-PROC-TEST-OPTION)
```

In der Prozedur PROC wird der Protokollstatus geprüft:

```
/IF (LOGGING-MODE (STREAM = *DATA) = 'NO')
/MODIFY-PROCEDURE-TEST-OPTIONS LOGGING = *YES
/END-IF
```

Wenn der Datenfluss noch nicht protokolliert wird, wird Protokollieren mit MODIFY-PROCEDURE-TEST-OPTIONS eingeschaltet.

LOWER-CASE( ) Großbuchstaben in Kleinbuchstaben umsetzen

Anwendungsgebiet: **String-Funktionen/Konvertierungsfunktionen**

Die Funktion LOWER-CASE( ) setzt alle Großbuchstaben im angegebenen String in Kleinbuchstaben um.

Die umzusetzenden Buchstaben müssen dem Standard-EBCDI-Code entsprechen. Es werden keine sprachspezifischen Varianten unterstützt.

Format

LOWER-CASE( )
STRING =string_ausdruck ,TRANSLATE = *ALL / *OUTSIDE-QUOTES-ONLY / *INSIDE-QUOTES-ONLY

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den umzusetzenden String.

TRANSLATE =
Bezeichnet, welche Zeichen umzusetzen sind.

TRANSLATE = *ALL
Bezeichnet, dass alle Zeichen umzusetzen sind.

TRANSLATE = *OUTSIDE-QUOTES-ONLY
Bezeichnet, dass nur die Zeichen außerhalb der Hochkommata umzusetzen sind.

TRANSLATE = *INSIDE-QUOTES-ONLY
Bezeichnet, dass nur die Zeichen innerhalb der Hochkommata umzusetzen sind.

Ergebnis

Nur aus Kleinbuchstaben, Ziffern und Sonderzeichen bestehender String

Fehlermeldung

```
SDP0486   UNGERADE ANZAHL VON APOSTROPHEN IM STRING-WERT
```

Beispiel

```
/A = 'AbCD123' // 'Ghi'
/B = LOWER-CASE(STRING = A)
/SHOW-VARIABLE B
B = abcd123ghi
```

```
/A = 'ABC''DEF''GHI'
/B = LOWER-CASE(STRING = A,TRANSLATE=*OUTSIDE-QUOTES-ONLY)
/SHOW-VARIABLE B
B = abc'DEF'ghi
```

```
/A = 'ABC''DEF''GHI'
/B = LOWER-CASE(STRING = A,TRANSLATE=*INSIDE-QUOTES-ONLY)
/SHOW-VARIABLE B
B = ABC'def'GHI
```

MAINCODE( ) Fehlerschlüssel abfragen

Anwendungsgebiet: **Kommando-Returncode**

Die Funktion MAINCODE( ) greift auf den Returncode des letzten Kommandos zu, das einen Fehler auslöste oder dem ein /SAVE-RETURNCODE folgte. Sie liefert den sieben Byte langen Fehlerschlüssel des Returncodes, der gleichzeitig der Meldungsschlüssel für eine Fehlermeldung ist (die übrigen Komponenten des Kommando-Returncodes werden mit den Funktionen SUBCODE1( ) und SUBCODE2( ) abgefragt).

Der Fehlerschlüssel, den die Funktion MAINCODE( ) liefert, besteht aus zwei Teilen: die ersten drei Byte bezeichnen die Meldungsklasse, die letzten vier Byte spezifizieren den Fehler. Der Fehlerschlüssel kann anschließend in der Funktion MSG( ) als Meldungsschlüssel eingesetzt werden, MSG( ) liefert dann - falls verfügbar den dazugehörenden Meldungstext.

Außerhalb von Dialogblöcken und Prozeduren kann MAINCODE( ) bzw. können allgemein Kommando-Returncodes nicht abgefragt werden.

Format

MAINCODE( ) MC( )

Ergebnistyp

STRING (<string 7..7>)

Eingabeparameter

Keine

Ergebnis

Fehlerschlüssel als String

Fehlermeldungen

```
SDP0428   KOMMANDO-RETURN-CODE KANN IM DIALOG NICHT ABGEFRAGT WERDEN
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Fehlerbehandlung mit MAINCODE( )

```
/BL1: BEGIN-BLOCK
/...
/  IF-BLOCK-ERROR
/    WRITE-TEXT '&(MSG(MAINCODE()))'
/  END-IF
/...
/END-BLOCK BLOCK = BL1
```

Tritt in dem entsprechenden Block (hier BL1) ein Fehler auf, wird der aktuelle Maincode ausgewertet und die entsprechende Meldung ausgegeben.

Es muss berücksichtigt werden, dass die Meldung `&(MSG(MAINCODE()))` ebenfalls Klammern und Hochkommas enthalten kann und dies bei der WRITE-TEXT-Angabe zu Problemen führt. Diese können durch Benutzung der Funktion TO-C-LITERAL() vermieden werden, z.B.:

```
/WRITE-TEXT &(TO-C-LITERAL('*** ' // MSG(MAINCODE()) // ' ***'))
```

MONTH( ) Monatsnamen ausgeben

Anwendungsgebiet: **Umgebungsinformationen** (Kalender)

Die Funktion MONTH( ) liefert den aktuellen Monatsnamen in der angegebenen Sprache, und zwar als drei Zeichen langer Kurzname. Zusammen mit anderen Kalenderfunktionen kann so eine vollständige Datumsangabe aufgebaut werden.

Format

MONTH( )
LANGUAGE = *ENGLISH / *GERMAN / *STD

Ergebnistyp

STRING (<string 1..3>)

Eingabeparameter

LANGUAGE = *ENGLISH / *GERMAN / *STD
Bestimmt die Sprache, in der der Monatsname ausgegeben wird.
*STD: Die Ausgabe erfolgt in der für die Task voreingestellten Sprache.

Ergebnis

Drei Zeichen langer Kurzname als String

Eingabeparameter	Ergebnis
LANGUAGE = *ENGLISH	JAN / FEB / MAR / APR / MAY / JUN / JUL / AUG / SEP / OCT / NOV / DEC
LANGUAGE = *GERMAN	JAN / FEB / MRZ / APR / MAI / JUN / JUL / AUG / SEP / OKT / NOV / DEZ

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = MONTH(LANGUAGE = *GERMAN)
/SHOW-VARIABLE A
A = FEB
```

MSG( ) Meldungstext ausgeben

Anwendungsgebiet: **Kommando-Returncodes** (Meldungen)

Die Funktion MSG( ) liefert den Meldungstext, der dem angegebenen Meldungsschlüssel zugeordnet ist, und zwar in der angegebenen Sprache und mit dem angegebenen Ausgabeformat.

Der Meldungsschlüssel kann z.B. für einige SDF-P-Kommandos vorher mit der Funktion MAINCODE( ) aus dem Kommando-Returncode abgefragt worden sein.

Format

MSG( )
MSG-IDENTIFICATION = string_ausdruck ,LANGUAGE = *STD / *ENGLISH / *GERMAN ,INSERT-00 = *NONE / string_ausdruck ,INSERT-01 = *NONE / string_ausdruck : : : ,INSERT-29 = *NONE / string_ausdruck ,MSG-STRUCTURE-OUTPUT = *NONE / *SYSMSG

Ergebnistyp

STRING (<string>)

Eingabeparameter

MSG-IDENTIFICATION = string_ausdruck
string_ausdruck enthält den 7 Byte langen Meldungsschlüssel mit folgendem Aufbau:

Byte 1-3: Buchstaben als Kennzeichen der Meldungsklasse
Byte 4-7: Ziffern 0-9, Buchstaben A-F als hexadezimale Darstellung der exakten Fehlernummer

LANGUAGE = *STD / ENGLISH / *GERMAN
Es wird der englische bzw. der deutsche Meldungstext ausgegeben.
Voreingestellt ist *STD, d.h. die Ausgabe erfolgt in der für die Task voreingestellten Sprache.
Die (früheren) Operandenwerte *E für *ENGLISH und *D für *GERMAN werden weiter kompatibel unterstützt.

INSERT-nn = *NONE / string_ausdruck
Bezeichnet den Zusatzinhalt einer Meldung.

MSG-STRUCTURE-OUTPUT =
Bezeichnet, ob Variablen für Ausgaben von Meldungen erzeugt und weitergeschickt werden müssen oder nicht.

MSG-STRUCTURE-OUTPUT = *NONE
Variablen für Ausgaben von Meldungen werden nicht über den S-Variablenstrom SYSMSG weitergeschickt. Der Meldungstext ist mit dem Kommando /HELP-MSG-INFORMATION MSG-ID=*LAST nicht verfügbar.

MSG-STRUCTURE-OUTPUT = *SYSMSG
Wenn die Meldungen garantiert sind, werden Variablen für Ausgaben von Meldungen über den S-Variablenstrom SYSMSG weitergeschickt. Der Meldungstext ist mit dem Kommando /HELP-MSG-INFORMATION MSG-ID=*LAST verfügbar.

Ergebnis

Meldungstext als String

Leerstring ('') bedeutet, dass dem Meldungsschlüssel kein Text zugeordnet ist.

Fehlermeldungen

```
SDP0413   NICHT ZULAESSIGE LAENGENANGABE
SDP0418   UNGUELTIGE MELDUNGSSCHLUESSEL
```

Beispiel

```
/DECLARE-VARIABLE MIP(TYPE=*STRUCTURE(*DYNAMIC)),MULTIPLE-ELEMENTS=*LIST
/ASSIGN-STREAM SYSMSG,TO=*VARIABLE(MIP)
/A=MSG(MSG-IDENTIFICATION='SDP1018',"Diese Meldung ist garantiert" -
/                  INSERT-00='MY-VARIABLE', -
/                  MSG-STRUCTURE-OUTPUT = *SYSMSG)
/B=MSG(MSG-IDENTIFICATION='SDP1010',"Diese Meldung ist NICHT garantiert" -
/                  INSERT-00='MY-SECOND-VARIABLE', -
/                  MSG-STRUCTURE-OUTPUT = *SYSMSG)
/SHOW-VARIABLE *ALL
A =  %  SDP1018 VARIABLE 'MY-VARIABLE' EXISTIERT BEREITS MIT ANDEREN
ATTRIBUTEN
B =  %  SDP1010 VARIABLE 'MY-SECOND-VARIABLE' HAT KEINEN WERT
MIP(*LIST).MSG-TEXT = %  SDP1018 VARIABLE 'MY-VARIABLE' EXISTIERT BEREITS MIT
ANDEREN ATTRIBUTEN
MIP(*LIST).MSG-ID = SDP1018
MIP(*LIST).IO = MY-VARIABLE
*END-OF-CMD
```

NEXT-VARIABLE-NAME( ) Variablenebene abfragen

Anwendungsgebiet: **Variablenzugriff** (Variablenname)

Mit der Funktion NEXT-VARIABLE-NAME( ) lässt sich der Aufbau von zusammengesetzten Variablen analysieren, vor allem in Verbindung mit der Funktion FIRST-VARIABLE-NAME( ).

Die Funktion NEXT-VARIABLE-NAME( ) liefert den Namen des in der gleichen Ebene folgenden Variablenelements. Wenn es kein Folgeelement auf dieser Ebene mehr gibt, liefert NEXT-VARIABLE-NAME( ) als Ergebnis *END.

Format

NEXT-VARIABLE-NAME( ) NEXT-VAR-NAME( )
VARIABLE-NAME = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet eine Variable.
Bezeichnet VARIABLE-NAME eine Liste, wird *END oder der entsprechende Elementname ausgegeben.
Bezeichnet VARIABLE-NAME ein Listenelement (liste#i), wird der Name des folgenden Elements ausgegeben (liste#i+1), sofern dieses existiert. Existiert kein Listenelement liste#i+1, wird *END ausgegeben. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe dazu das folgende Beispiel sowie das Beispiele bei IS-DECLARED( )).

Ergebnis

Name des Elements, das in der zusammengesetzten Variablen auf der gleichen Ebene auf „string_ausdruck“ folgt.

**END*
„string_ausdruck“ war das letzte Element der Ebene.

Fehlermeldung

```
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel 1

Eine zusammengesetzte Variable AR enthält folgende Elemente.

```
AR#1
AR#2
AR#3
```

```
/A = NEXT-VARIABLE-NAME(VARIABLE-NAME = 'AR#1')
/SHOW-VARIABLE A
A = AR#2
```

```
/A=NEXT-VARIABLE-NAME(VARIABLE-NAME = 'AR#3')
/SHOW-VARIABLE A
A = *END
```

Wenn NEXT-VARIABLE-NAME( ) auf AR#1 angewendet wird, wird als Ergebnis AR#2 geliefert, der Name des auf AR#1 folgenden Elements.
AR#3 ist das letzte Element des Arrays, NEXT-VARIABLE-NAME( ) liefert also *END.

Beispiel 2

Eine zusammengesetzte Variable ARR enthält folgende Elemente:

```
ARR#1
ARR#22
ARR#30
```

```
/A = NEXT-VAR-NAME('ARR#1')
/SHOW-VARIABLE A
A = ARR#22
```

PROC-LEVEL( ) Schachtelungstiefe abfragen

Anwendungsgebiet: **Prozedurinformationen**

Die Funktion PROC-LEVEL( ) liefert die aktuelle Schachtelungstiefe der S-Prozedur.

Format

PROC-LEVEL( )

Ergebnistyp

INTEGER

Eingabeparameter

Keine

Ergebnis

Zahl vom Typ INTEGER

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Im Dialog:

```
/A = PROC-LEVEL( )
/SHOW-VARIABLE A
A = 0
```

In geschachtelt aufgerufenen Prozeduren:

Die drei Prozeduren C.PROC1, C.PROC2 und C.PROC3 werden geschachtelt aufgerufen. In jeder Prozedur wird die Schachtelungstiefe abgefragt.

C.PROC1

```
/A = PROC-LEVEL( )
/SHOW-VARIABLE A
/CALL-PROCEDURE C.PROC2
```

C.PROC2

```
/B = PROC-LEVEL( )
/SHOW-VARIABLE B
/CALL-PROCEDURE C.PROC3
```

C.PROC3

```
/C = PROC-LEVEL( )
/SHOW-VARIABLE C
```

Beim Ablauf werden folgende Zeilen ausgegeben:

```
A = 1
B = 2
C = 3
```

PROCESSOR( ) Prozessornamen abfragen

Anwendungsgebiet: **Umgebungsinformationen** (TIAM)

Die Funktion PROCESSOR( ) liefert den Prozessornamen der TIAM-Station, das heißt der Datenstation, an der der aktuelle Auftrag gestartet wurde.

Format

PROCESSOR( )

Ergebnistyp

STRING (<string 1..8>)

Eingabeparameter

Keine

Ergebnis

TIAM-Prozessorname als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = PROCESSOR( )
/SHOW-VARIABLE A
A = FIREBALL
```

Zum Vergleich Name der TIAM-Station im Feld STATION (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:      29XX       TYPE:    3 DIALOG   NOW:      2007-04-26.110747
%JOBNAME: BERTA       PRI:     0 210
%USERID:  USER1       JCLASS:  JCDSTD     LOGON:    2007-04-26.1053
%ACCNB:   ACC01       CPU-MAX:   9999     CPU-USED:000000.6447
%STATION: $$$06580    PROC:    FIREBALL
...
```

PROG-MONJV( ) Programm-MONJV abfragen

Anwendungsgebiet: **Umgebungsinformationen /Jobvariablen-Funktionen**

Die Funktion PROG-MONJV( ) liefert den Namen der Jobvariablen, die das geladene Programm überwacht.

Diese Funktion kann nur dann genutzt werden, wenn im System das Softwareprodukt „Jobvariablen“ geladen ist. Nähere Informationen über Jobvariablen enthält das Handbuch „Jobvariablen“ [5].

Format

PROG-MONJV( )

Ergebnistyp

STRING (<string 1..255>)

Eingabeparameter

Keine

Ergebnis

Jobvariablenname als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

PROG-NAME( ) Programmnamen abfragen

Anwendungsgebiet: **Programm-Information**

Die Funktion PROG-NAME( ) liefert den auf acht Zeichen abgeschnittenen internen Namen des aktuell geladenen Programms. Wenn der Name eines Bindemoduls aus der Bindemodulbibliothek (OML) geliefert werden soll, werden 0 Zeichen ausgegeben.

Format

PROG-NAME( )

Ergebnistyp

STRING

Eingabeparameter

Keine

Ergebnis

Programmname als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

In einer Prozedur wird das Dienstprogramm LMS gestartet:

```
/START-LMS
...
```

Der Programmlauf wird unterbrochen und der Programmname geprüft:

```
/IF (PROG-NAME( ) = 'LMSSDF')
/   RESUME-PROGRAM
/ELSE
/   ...
/END-IF
```

RENAME( ) Neue Namen mit Wildcards bilden

Anwendungsgebiet: **Stringfunktionen**

Die Funktion RENAME( ) liefert einen neuen Namen. Der neue Name wird auf der Basis des Eingabenamens gebildet mithilfe von Wildcards.

Format

RENAME( )
INPUT-NAME = string_ausdruck ,WILDCARD-PATTERN = string_ausdruck ,CONSTRUCTION-WILDCARD = string_ausdruck ,NO-MATCH = *WARNING / *IGNORE / *ERROR ,WILDCARD-MODE = *BS2000 / *POSIX

Ergebnistyp

STRING

Eingabeparameter

INPUT-NAME = string_ausdruck
Bezeichnet den String, der ersetzt werden soll.

WILDCARD-PATTERN = string_ausdruck
Bezeichnet das gesuchte Muster.

CONSTRUCTION-WILDCARD = string_ausdruck
Bezeichnet die Regeln für die Bildung des neuen Namens. (Siehe dazu Handbuch „LMS“ [11].)

NO-MATCH =
Bezeichnet, was gemacht wird, wenn das gesuchte Muster nicht gefunden wird.

NO-MATCH = *WARNING
Es wird eine Warnung ausgegeben.

NO-MATCH = *IGNORE
Keine Aktion.

NO-MATCH = *ERROR
Es wird eine Fehlermeldung ausgegeben.

WILDCARD-MODE = *BS2000 / *POSIX
Gibt an, ob Wildcards bei der Ersetzung in der BS2000-Wildcard-Syntax oder in der POSIX-Wildcard-Syntax interpretiert werden.

Ergebnis

Neuer Name als String

Fehlermeldung

```
SDP0467   KEINEN NAMEN GEFUNDEN: VERARBEITUNG WIRD FORTGESETZT
SDP0468   KEINEN NAMEN GEFUNDEN
SDP0482   EIN EINGABE-STRING IST ZU LANG (1..255)
SDP0483   FALSCHER CONSTRUCTION-WILDCARD-WERT
SDP0484   ZU GROSSER AUSGABE-STRING (1..255)
```

Beispiel

```
/A = RENAME('A.B','A.*','NEWA.*')
/SHOW-VARIABLE A
A = NEWA.B

/A = RENAME('B.A','A.*','NEWA.*')
SDP0467 KEIN NAME GEFUNDEN; VERARBEITUNG WIRD FORTGESETZT
/SHOW-VARIABLE A
A = B.A

/A = RENAME('B.A','A.*','NEWA.*',NO-MATCH=*IGNORE)
/SHOW-VARIABLE A
A = B.A

/A = RENAME('B.A','A.*','NEWA.*',NO-MATCH=*ERROR)
SDP0468 KEINEN NAMEN GEFUNDEN
SDP0431 FEHLER 'SDP0468' IN BUILTIN-FUNKTION 'RENAME'
SDP0239 FEHLER WAEHREND DER AUSWERTUNG DER RECHTEN SEITE DER ZUWEISUNG

/A = RENAME('A.B.C','A///C','NEWA///NEWC')
/SHOW-VARIABLE A
A = NEWA.B.NEWC

/A = RENAME('A.B','*.*','**')
/SHOW-VARIABLE A
A = AB

/A = RENAME('A.B','*.*','<1><1>')
/SHOW-VARIABLE A
A = AA

/A = RENAME('A.B','/./','XYZ<2>')
/SHOW-VARIABLE A
A = XYZB
```

REPLACE( ) Teilstring überschreiben oder ersetzen

Anwendungsgebiet: **String-Bearbeitung**

Die Funktion REPLACE( ) überschreibt oder ersetzt einen Teilstring innerhalb eines Strings durch einen anderen String. Dabei kann der ursprüngliche String länger werden.

Format

REPLACE( )
STRING = string_ausdruck$_1$,START = 1 / zahl ,REPLACE = string_ausdruck$_2$,SUPPRESSED-LENGTH = *REPLACE-LENGTH / zahl

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck$_1$
Bezeichnet den String, in dem ein Teilstring ersetzt werden soll.

START = 1 / zahl
Bezeichnet die Position, ab der der (Eingabe-)String überschrieben wird; „zahl“ ist ein positiver Integer-Wert bzw. ein arithmetischer Ausdruck, der zu einem positiven Integer-Wert ausgewertet wird.

REPLACE = string_ausdruck$_2$
Bezeichnet den String, der ab der Startposition eingesetzt wird.

SUPPRESSED-LENGTH =
Gibt an, ob string_ausdruck$_2$ Teile des (Eingabe-)Strings überschreibt oder ersetzt.

SUPPRESSED-LENGTH = *REPLACE-LENGTH
Ab der bei START=.. angegebenen Position werden die folgenden Zeichen durch string_ausdruck$_2$ überschrieben (in der Länge von string_ausdruck$_2$).

SUPPRESSED-LENGTH = zahl
Bezeichnet die Anzahl von Zeichen, die ab der bei START=.. angegebenen Position unterdrückt werden und für die string_ausdruck$_2$ eingesetzt wird.

Ergebnis

Geänderter String

Fehlermeldung

```
SDP0412   START-POSITION AUSSERHALB MOEGLICHEN BEREICHS
SDP0413   NICHT ZULAESSIGE LAENGENANGABE
```

Beispiele

```
/A = 'ABCDEFGHIJ'
/B = REPLACE(STRING = A, REPLACE = '**')
/SHOW-VARIABLE A
A = ABCDEFGHIJ
/SHOW-VARIABLE B
B = **CDEFGHIJ

/C = 10
/B = REPLACE(STRING = A, START = C, REPLACE = 'KLMN')
/SHOW-VARIABLE B
B = ABCDEFGHIKLMN

/B = REPLACE(STRING = A, START = 0, REPLACE = '**')
SDP0412 START-POSITION AUSSERHALB MOEGLICHEN BEREICHS
SDP0431 FEHLER 'SDP0412' IN BUILTIN-FUNCTION 'REPLACE'
SDP0239 FEHLER WAEHREND DER AUSWERTUNG DER RECHTEN SEITE DER ZUWEISUNG
/A = REPLACE(STRING = A, REPLACE = '****')
/SHOW-VARIABLE A
A = ****EFGHIJ
```

Die letzte Zuweisung an die Variable B führt zum Fehler SDP0412, da bei START kein korrekter Wert angegeben wurde.

```
/WHILE (INDEX(TESTSTRING,X'00') > 0)
/    TESTSTRING = REPLACE(TESTSTRING,INDEX(TESTSTRING,X'00'),X'40')
/END-WHILE
```

Innerhalb der Variablen TESTSTRING werden alle X’00’ durch Leerzeichen (X’40’) ersetzt.

Beispiele mit dem Operanden SUPPRESSED-LENGTH=..

```
/A = 'I am the king of the replace()'
/B1= 'developer'              "REPLACE"
/B2 = REPLACE (A,10,B1,4)     " 10th position is 'k' "
/SHOW-VAR B2
B2 = I am the developer of the replace()

/C1 = 'not '                  "INSERT"
/C2 = REPLACE (A,6,C1,0)      " 6th position is 't' "
/SHOW-VAR C2
C2 = I am not the king of the replace()

/D1 = 'replacement'           "OVERWRITE (like before)"
/D2 = REPLACE (A,22,D1)       " 22th position is 'r' "
/SHOW-VAR D2
D2 = I am the king of the replacement
```

RUN-PRIORITY( ) Laufzeitpriorität abfragen

Anwendungsgebiet: **Job-Informationen**

Die Funktion RUN-PRIORITY( ) liefert die Prioritätsstufe des aktuellen Jobs.

Der gelieferte Wert kann anschließend geprüft und dann - wenn nötig - die Priorität der Task verändert werden.

Format

RUN-PRIORITY( ) RUN-PRIO( )

Ergebnistyp

INTEGER (<integer 0..255>)

Eingabeparameter

Keine

Ergebnis

Zahl vom Typ INTEGER, 0 ≤ zahl ≤ 255

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = RUN-PRIORITY( )
/SHOW-VARIABLE A
A = 210
```

Zum Vergleich die Laufzeitpriorität im Feld PRI (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:     29XX       TYPE:    3 DIALOG   NOW:     2007-04-26.110747
%JOBNAME: BERTA      PRI:     0 210
%USERID:  USER1      JCLASS:  JCDSTD     LOGON:   2007-04-26.1053
......
```

SDF-P-VERSION( ) SDF-P-Version abfragen

Anwendungsgebiet: **Prozedurinformationen**

Die Funktion SDF-P-VERSION( ) informiert über die installierte Version des (kostenpflichtigen) Subsystems SDF-P bzw. über die aktuelle Version des im Grundausbau enthaltenen Subsystems SDF-P-BASYS.

Format

SDF-P-VERSION( )
FUNCTION-RANGE = *STD / *BASIC

Ergebnistyp

STRING

Eingabeparameter

FUNCTION-RANGE = *STD / *BASIC
Operandenwert *STD: aktuelle Version des (kostenpflichtigen) Subsystems SDF-P.
Operandenwert *BASIC: aktuelle Version des im Grundausbau enthaltenen Subsystems SDF-P-BASYS

Ergebnis

Versionsangabe als String

Fehlermeldung

Keine Fehlermeldung

Beispiel

```
/A = SDF-P-VERSION
/B = SDF-P-VERSION(FUNCTION-RANGE=*BASIC)

/SHOW-VARIABLE A
A = V02.4A10

/SHOW-VARIABLE B
B = V02.4A10
```

SDF-STRUCTURE-VALUE( ) Wert einer Struktur ausgeben

Anwendungsgebiet: **Stringfunktionen / Prüffunktionen**

Die Funktion SDF-STRUCTURE-VALUE( ) liefert den Inhalt einer SDF-Struktur teilweise oder ganz.

Format

SDF-STRUCTURE-VALUE( )
STRING = string_ausdruck ,OPERAND-NAME = *ROOT / string_ausdruck / arithm_ausdruck ,ATTACHED-STRUCTURE = *YES / *NO

Ergebnistyp

STRING (<string>)

Eingabeparameter

STRING = string_ausdruck
Name des Strings, der analysiert werden soll. Er wird intern durch IS-SDF-STRUCTURE( ) geprüft. Siehe deshalb die Beschreibung dieser Funktion.

OPERAND-NAME = *ROOT / string_ausdruck / arithm_ausdruck
Name des Operanden bzw. Position, wo der Wert zu finden ist.
Es werden nur die direkt angesprochenen Operanden untersucht bzw. die Operanden der direkt angesprochenen Ebene. Andere Operanden bzw. Operanden anderer Ebenen müssen extra angesprochen werden.

ATTACHED-STRUCTURE= *YES / *NO
Gibt an, ob die betroffene Struktur angegeben werden soll oder nicht.

Ergebnis

Gewünschter Ausdruck als String

Fehlermeldungen

```
SDP0460   DER EINGEGEBENE STRING IST KEINE STRUKTUR
SDP0461   NUMERISCHER OPERANDENWERT MUSS GROESSER NULL SEIN
SDP0462   '(&00)' IST KEIN STRUKTURNAME
SDP0463   DER ANGEGEBENE OPERAND '(&00)' IST UNBEKANNT
SDP0464   OPERANDENANGABE ENTHAELT ZU VIELE MEHRDEUTIGKEITEN
SDP0465   OPERAND VOM TYP BOOLEAN HIER UNZULAESSIG
```

Beispiel 1

```
/START-SDF-A
//OPEN syssdf.myuser,type=user,mode=create
//ADD-CMD my-cmd-1,IMPL=*PROC('myproc')
//ADD-OPERAND op
//ADD-VALUE *NAME
//ADD-VALUE *KEYWORD(STAR=*MANDATORY),STRUCTURE=*YES,VALUE='PARAMETERS'
//ADD-OPERAND op1,RESULT-OPERAND-LEVEL=2
//ADD-VALUE *NAME
//ADD-OPERAND op2,RESULT-OPERAND-LEVEL=2
//ADD-VALUE *FILENAME
//CLOSE-STRUCTURE
//END
```

Zunächst wird eine SDF-Syntaxdatei mit dem Namen SYSSDF.MYUSER angelegt, in der anschließend das Kommando MY-CMD-1 definiert wird. Dieses Kommando ist durch die Prozedur MYPROC implementiert (siehe SDF-A-Anweisung //ADD-CMD).

Inhalt der Prozedur MYPROC:

```
/SET-PROCEDURE-OPTIONS "Procedure: myproc"
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER op
/END-PARAMETER-DECLARATION

/value=SDF-STRUCTURE-VALUE(op,*ROOT,*NO)
/WRITE-TEXT 'root value : &value'

/value=SDF-STRUCTURE-VALUE(op,'OP1')
/WRITE-TEXT 'operand op1 value : &value'

/value=SDF-STRUCTURE-VALUE(op,'OP2')
/WRITE-TEXT 'operand op1 value : &value'

/EXIT-PROCEDURE
```

Um das Kommando MY-CMD-1 aufzurufen, muss die Syntaxdatei aktiviert werden:

```
/MODIFY-SDF-OPTIONS *ADD(ADD-NANE=syssdf.myuser)
```

Aufruf des Kommandos MY-CMD-1:

```
/MY-CMD-1 OP=*PARAMETERS(OP1=VALUE1,OP2=VALUE2)
```

Ausgabe

```
root value: *PARAMETERS
operand op1 value: VALUE1
operand op2 value: VALUE2
```

Das Kommando MY-CMD-1 ruft die Prozedur MYPROC auf und liefert die Inhalte der angegebenen Struktur OP zurück.

Beispiel 2

```
/A='*PARAMETERS(OP1=val1(OP11=val11,OP12=val12),OP2=val2(val21,val22))'

/B=SDF-STRUCTURE-VALUE(A,'OP1',*YES)
/SHOW-VAR B
B=val1(OP11=val11,OP12=val12)

/C=SDF-STRUCTURE-VALUE(B,'OP11')
/SHOW-VAR C
C=val11

/D=SDF-STRUCTURE-VALUE(B,2)
/SHOW-VAR D
D=val12
```

SEARCH-LIST-INDEX( ) String in einer Liste suchen

Anwendungsgebiet: **Stringfunktionen**

Die Funktion SEARCH-LIST-INDEX( ) sucht in einer Listenvariablen einen String oder einen nach POSIX-Regeln gebildeten regulären Ausdruck. Der Returnwert gibt die Nummer des ersten Listenelements an, das diesen Ausdruck bzw. diesen Suchstring enthält.

Normalerweise - wenn man eine solche Operation durch eine Schleife mit der Funktion INDEX ausführen lässt - benötigt man dafür sehr viel Zeit. Durch die Einführung dieser vordefinierten Funktion wird die Ausführungszeit wesentlich reduziert, da die Suche in einem Schritt ausgeführt wird.

Format

SEARCH-LIST-INDEX( )
LIST-VARIABLE-NAME = string_ausdruck$_1$,PATTERN = string_ausdruck$_2$,BEGIN-INDEX = 1 / arithm_ausdruck ,END-INDEX = *LAST / arithm_ausdruck ,BEGIN-COLUMN = 1 / arithm_ausdruck ,END-COLUMN = *LAST / arithm_ausdruck ,PATTERN-TYPE = *STRING / *REGULAR-EXPRESSION ,DIRECTION = *FORWARD / *REVERSE

Ergebnistyp

INTEGER

Eingabeparameter

LIST-VARIABLE-NAME= string_ausdruck$_1$
Bezeichnet die Variable, die aus einer Liste von Strings besteht.

PATTERN = string_ausdruck$_2$
Bezeichnet den Suchstring oder regulären Ausdruck, nach dem sequenziell innerhalb der Listenvariablen gesucht wird, die in LIST-VARIABLE-NAME angegeben ist.

BEGIN-INDEX =
Bezeichnet das erste Listenelement bzw. den Index, bei dem die Suche gestartet wird. Listenelemente mit einem kleineren Index werden nicht geprüft. Ist der angegebene Index größer als die Anzahl der Elemente, wird immer „0“ zurückgegeben; d.h. es wird „nichts gefunden“ zurückgemeldet.

BEGIN-INDEX = 1
Die Suche beginnt beim Listenelement 1, d.h. die Liste wird von Beginn an durchsucht.

BEGIN-INDEX = arithm_ausdruck
Ab dem angegebenen Listenelement beginnt die Suche.

END-INDEX =
Bezeichnet das letzte Listenelement bzw. den Index, bis zu dem gesucht wird.
Ist einschließlich diesem Index nichts gefunden worden, wird immer „0“ zurückgegeben.

END-INDEX = *LAST
Die Suche endet beim letzten Listenelement.

END-INDEX = arithm_ausdruck
Die Suche endet mit dem angegebenen Listenelement.

BEGIN-COLUMN =
Die Suche wird vom Beginn auf einen bestimmten Spaltenbereich beschränkt.
BEGIN-COLUMN zeigt das erste Zeichen im String an, das als Suchstart im jeweiligen Element betrachtet wird, um den in PATTERN angegeben Suchstring zu finden.
Wenn PATTERN-TYPE = *REGULAR-EXPRESSION ist, gehört das Zeichen „^“ am Anfang des gesuchten Strings nicht mehr zum eigentlichen Suchstring, sondern trifft genau die Position vor dem in BEGIN-COLUMN angegebenen Wert.
Der durchsuchte String ist leer, wenn das Listenelement weniger Zeichen enthält als bei BEGIN-COLUMN angegeben sind.

BEGIN-COLUMN = 1
Die Suche beginnt ab Spalte 1, d.h. es wird das gesamte Listenelement durchsucht.

BEGIN-COLUMN = arithm_ausdruck
Ab der angegebenen Spalte bzw. ab diesem Zeichen wird in einem Listenelement nach dem Suchstring gesucht.

END-COLUMN =
Die Suche wird vom Ende gesehen auf einen bestimmten Spaltenbereich beschränkt.
END-COLUMN zeigt dasjenige Zeichen im String an, das als Suchende im jeweiligen Element betrachtet wird, um den in PATTERN angegeben Suchstring zu finden. Die Zeichen des Listenelements nach der in END-COLUMN bezeichneten Position werden bei der Suche nicht berücksichtigt.

Wenn PATTERN-TYPE = *REGULAR-EXPRESSION ist, trifft das Zeichen „$“ am Ende des gesuchten Strings genau die Position nach dem in END-COLUMN angegebenen Wert (das bedeutet: dieses Zeichen gehört nicht mehr zum eigentlichen Suchstring), oder es trifft das Ende des Listenelements, wenn das Listenelement weniger Zeichen enthält, als bei END-COLUMN angegeben sind.

END-COLUMN = *LAST
Die Suche endet mit dem Ende des Strings bzw. mit dem Ende des Listenelements.

END-COLUMN = arithm_ausdruck
Ab der angegebenen Spalte bzw. ab diesem Zeichen wird in einem Listenelement nach dem Suchstring nicht mehr gesucht.

PATTERN-TYPE =
Bezeichnet den Datentyp des Suchstrings oder gesuchten regulären Ausdrucks.

PATTERN-TYPE = *STRING
Der Datentyp ist STRING.

PATTERN-TYPE = *REGULAR-EXPRESSION
Der Datentyp entspricht dem eines nach POSIX-Regeln erstellten regulären Ausdrucks. Dieser ersetzt eine beliebige, auch leere Zeichenfolge.

DIRECTION =
Bestimmt die Richtung, in der die Listenvariable durchsucht werden soll.

DIRECTION = *FORWARD
Die Suche beginnt bei dem in BEGIN-INDEX bezeichneten Listenelement und endet bei dem in END-INDEX bezeichneten Listenelement.

DIRECTION = *REVERSE
Die Suche verläuft in umgekehrter Richtung, d.h. sie beginnt bei dem in END-INDEX bezeichneten Listenelement und endet bei dem in BEGIN-INDEX bezeichneten Listenelement.

Ergebnis

Positiver Integer-Wert, der das erste Listenelement angibt, welches den in PATTERN angegebenen Suchstring enthält. Dieser Wert ist größer gleich dem in BEGIN-INDEX angegebenen Wert und kleiner gleich der Anzahl der Listenelemente der angegebenen Variablen.

0
Der Suchstring wurde nicht gefunden.

Fehlermeldungen

```
SDP0492   DER OPERAND PATTERN UND DIE LISTENELEMENTE DUERFEN KEIN NULLBYTE
          (X'00') ENTHALTEN

SDP0493   BEGIN-INDEX, END-INDEX, BEGIN-COLUMN UND END-COLUMN MUESSEN
          GROESSER NULL SEIN

SDP0494   SYNTAXFEHLER IN REGULAEREM AUSDRUCK FUER OPERAND PATTERN

SDP1008   VARIABLE BZW. LAYOUT '(&00)' EXISTIERT NICHT

SDP1096   VARIABLE '(&00)' MUSS EINE LISTE VOM TYP STRING ODER ANY SEIN UND
          NUR STRING-WERTE ENTHALTEN
```

Beispiel 1

```
/DECLARE-VARIABLE DATEI-NEU(TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(DATEI-NEU), INPUT=*SYSDTA
Erste Zeile von der Datei
Zweite
Dritte Zeile von der Datei
4.
5. ........
6. von der Datei
7. ... von der ...
*END-OF-CMD
/MATCH = 0
/REPEAT
/   MATCH=SEARCH-LIST-INDEX('DATEI-NEU',PATTERN='Zeile', BEGIN-INDEX=MATCH+1)
/   SHOW-VARIABLE MATCH
/UNTIL (MATCH == 0)
```

Ausgabe

```
MATCH = 1
MATCH = 3
MATCH = 0
```

Dieses Beispiel zeigt, wie in der Listenvariablen DATEI-NEU mithilfe von SEARCH-LIST-INDEX( ) nach einem String (hier „Zeile“) ab einem bestimmten Zeichen gesucht wird. Durch die REPEAT-Schleife ist dabei gewährleistet, dass der Wert bei BEGIN-INDEX jeweils um den Wert 1 hochgesetzt wird, bis die gesamte Listenvariable durchlaufen ist. Wird innerhalb der REPEAT-Schleife folgender Ausdruck geschrieben, ändert sich die Ausgabe:

Beispiel 2

```
/MATCH=SEARCH-LIST-INDEX('DATEI-NEU', -
/                        PATTERN =' von ', -
/                        BEGIN-COLUMN=7,-
/                        END-COLUMN=11)
/SHOW-VARIABLE MATCH
```

Ausgabe

```
MATCH = 7
```

In diesem Beispiel wird die Listenvariable DATEI-NEU (siehe Beispiel 1) nach dem String „ von “ durchsucht, wobei die Suche auf den Spaltenbereich Spalte 7 bis (inklusive) Spalte 11 beschränkt wird: Nur das Listenelement 7 erfüllt die Suchbedingung, nicht jedoch die Listenelemente 1, 3 und 6.

Beispiel 3

```
/DECLARE-VARIABLE RECORD-LIST(TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(RECORD-LIST), INPUT=*SYSDTA
WIEDEMANN BERNHARD 64528
BACHMANN MICHAEL 37214
ARTMANN HELMUT 74634
HEUBACH HUGO 97884
BACH ANDREAS 12012
KIRSCHNER ANITA 76325
*END-OF-CMD
/NAME = 'BACH'
/MATCH=SEARCH-LIST-INDEX('RECORD-LIST',
/                        PATTERN ='^&NAME. ', -
/                        PATTERN-TYPE=*REGULAR-EXPRESSION -
/                      )
/NUMBER = INTEGER(EXTRACT-FIELD(RECORD-LIST#MATCH,3))
/WRITE-TEXT '&NAME. HAS NUMBER &NUMBER.'
```

Ausgabe

```
BACH HAS NUMBER 12012
```

Der Name „BACH“ wird in der Listenvariablen RECORD-LIST gesucht. Es wird nur ein Treffer gemeldet, denn durch die Angabe von PATTERN-TYPE = *REGULAR-EXPRESSION erfüllen „HEUBACH“ und „BACHMANN“ die Kriterien für den gesuchten String nicht: Der eine beginnt nicht mit „B“ und der andere endet nicht mit einem Leerzeichen nach „BACH“.

Beispiel 4

```
/DECLARE-VARIABLE A-LIST (TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/                 A-LIST#1  = 'ACTIVE  '
/                 A-LIST#2  = 'WAITING '
/                 A-LIST#3  = 'INACTIVE'
/                 A-LIST#4  = 'ABORTED '
/                 A-LIST#5  = 'ACTIVE  '
/                 A-LIST#6  = 'LOCKED  '
/                 A-LIST#7  = 'WAITING '
/                 A-LIST#8  = 'ACTIVE  '
/                 A-LIST#9  = 'ACTIVE  '
/                 A-LIST#10 = 'INACTIVE'
/WAITING-IDX=SEARCH-LIST-INDEX('A-LIST','WAITING',DIRECTION=*FORWARD)
/SHOW-VARIABLE WAITING-IDX
WAITING-IDX = 2
/WAITING-IDX=SEARCH-LIST-INDEX('A-LIST','WAITING',DIRECTION=*REVERSE)
/SHOW-VARIABLE WAITING-IDX
WAITING-IDX = 7
```

In der Listenvariable A-LIST wird der String „WAITING“ gesucht. Die Vorwärtssuche meldet als Treffer das Listenelement 2. Die Rückwärtssuche meldet das Listenelement 7 als Treffer.

SESSION-NUMBER( ) Systemlaufnummer abfragen

Anwendungsgebiet: **Systeminformationen**

Die Funktion SESSION-NUMBER( ) liefert die Systemlaufnummer des aktuell laufenden Systems (die Systemlaufnummer ist z.B. Bestandteil des CONSLOG-Dateinamens).

Format

SESSION-NUMBER( )

Ergebnistyp

STRING (<string 3..3>)

Eingabeparameter

Keine

Ergebnis

Systemlaufnummer als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = SESSION-NUMBER( )
/SHOW-VARIABLE A
A = 012
```

SIZE( ) Größe zusammengesetzter Variablen abfragen

Anwendungsgebiet: **Variablenzugriff** (Variableneigenschaften)

Die Funktion SIZE( ) fragt ab, aus wie vielen Elementen die angegebene Variable besteht. SIZE( ) kann auf Arrays, Listen und Strukturen angewendet werden.

In Verbindung mit der Funktion NEXT-VARIABLE-NAME( ) kann das Ergebnis von SIZE( ) zum Beispiel als Schleifenzähler verwendet werden, wenn der Aufbau von zusammengesetzten Variablen analysiert wird.

Format

SIZE( )
VARIABLE-NAME = string_ausdruck

Ergebnistyp

INTEGER

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet eine Variable (Array, Liste oder Struktur). Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe dazu das folgende Beispiel sowie das Beispiel bei IS-DECLARED( )).

Ergebnis

Zahl der Elemente, aus denen die Variable „string_ausdruck“ besteht.

0
Der Wert „0“ wird in folgenden Fällen zurückgegeben:
– „string_ausdruck“ enthält kein Element.
– „string_ausdruck“ bezeichnet keine zusammengesetzte Variable, oder es gibt keine zusammengesetzte Variable mit diesem Namen.

Fehlermeldung

```
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel

In der aktuellen Task wurde bereits taskglobal eine Listenvariable VARLIST angelegt und initialisiert. In der aktuellen Prozedur darf VARLIST genau 10 Elemente haben.

```
/IF SIZE('VARLIST') > 10
/  WRITE-TEXT 'zu viele Elemente'
/  FOR I = *LIST(VARLIST)
/    SHOW-VARIABLE I
/  END-FOR
/  GOTO ZUVIEL
/ELSE
/  ZAEHL = SIZE('VARLIST')
/  WHILE ZAEHL < 10
/    VARLIST = ZAEHL+1, WRITE-MODE = *EXTEND
/    ZAEHL = ZAEHL+1
/  END-WHILE
/END-IF
...
/ZUVIEL: "Fehlerbehandlung > 10 Listenelemente"
...
```

In der ersten Zeile wird die Größe der Listenvariablen VARLIST geprüft. Wenn diese mehr als 10 Elemente enthält, wird der Inhalt aller Elemente über eine FOR-Schleife ausgegeben und der Prozedurlauf mit der Fehlerbehandlung ZUVIEL fortgesetzt.
Wenn VARLIST nicht mehr als 10 Elemente enthält, wird der ELSE-Zweig des IF-Blocks abgearbeitet, der in einer WHILE-Schleife Elemente an VARLIST anhängt, bis VARLIST genau 10 Elemente enthält. Der Prozedurlauf wird dann mit dem Kommando fortgesetzt, das auf das Kommando END-IF folgt.

STATION( ) TIAM-Stationsnamen abfragen

Anwendungsgebiet: **Umgebungsinformationen** (TIAM)

Die Funktion STATION( ) liefert den Stationsnamen der TIAM-Station, von der die Prozedur angestoßen wurde.

Format

STATION( )

Ergebnistyp

STRING(<string 1..8>)

Eingabeparameter

Keine

Ergebnis

Stationsname als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = STATION( )
/SHOW-VARIABLE A
A = $$$06580
```

Zum Vergleich Name der TIAM-Station im Feld STATION (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:     29XX       TYPE:    3 DIALOG   NOW:     2007-04-26.110747
%JOBNAME: BERTA      PRI:     0 210
%USERID:  USER1      JCLASS:  JCDSTD     LOGON:   2007-04-26.1053
%ACCNB:   ACC01      CPU-MAX:   9999     CPU-USED:000000.6447
%STATION: $$$06580   PROC:    FIREBALL
...
```

STATION-TYPE( ) TIAM-Gerätetyp abfragen

Anwendungsgebiet: **Umgebungsinformationen**

Die Funktion STATION-TYPE( ) liefert den generierten Gerätetyp der TIAM-Station, von der die Prozedur aufgerufen wurde.

Format

STATION-TYPE( )

Ergebnistyp

STRING(<string 1..8>)

Eingabeparameter

Keine

Ergebnis

Gerätetyp der TIAM-Station als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = STATION-TYPE( )
/SHOW-VARIABLE A
A = DSS-9763
```

STD-CAT-ID( ) Katalogkennung abfragen

Anwendungsgebiet: **Benutzer-Informationen**

Die Funktion STD-CAT-ID( ) liefert die Kennung des Pubsets, der der aktuellen Benutzerkennung als Default-Pubset zugeteilt wurde.

Der Default-Pubset ist der Pubset, auf dem die Daten eines Benutzers angelegt, das heißt, katalogisiert werden, wenn er beim Erstellen des Katalogeintrags keine Katalogkennung angibt.

Format

STD-CAT-ID( )

Ergebnistyp

STRING (<string 1..4>)

Eingabeparameter

Keine

Ergebnis

Bis zu vier Zeichen lange Katalogkennung als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = STD-CAT-ID( )
/SHOW-VARIABLE A
A = 1OSN
```

STMT-SPINOFF( ) Statement-Spinoff abfragen

Anwendungsgebiet: **Prozedurinformationen**

Die Funktion STMT-SPINOFF( ) zeigt an, ob für das geladene Programm ein Statement-Spinoff eingeschaltet ist.

Ein Statement-Spinoff wird ausgelöst, wenn in einem Programm SDF-Anweisungen von der Systemdatei SYSSTMT gelesen werden (Read Statement mit dem Makroaufruf CMDRST bzw. RDSTMT, siehe Handbuch „SDF-A“ [16]) und dabei Fehler auftreten. Auf Anweisungsebene kann das Statement-Spinoff durch die Anweisung //STEP abgefangen werden. Auf Kommando-Ebene ist die Funktion STMT-SPINOFF( ) die einzige Möglichkeit, ein Statement-Spinoff abzufragen.

Hinweis

Innerhalb von Kommandoblöcken, in denen Returncodes von Programmanweisungen wie Kommando-Returncodes interpretiert werden (siehe Kommando BEGIN-BLOCK, Operand PROGRAM-INPUT=*MIXED-WITH-CMD(PROPAGATE-STMT-RC=*TO-CMD-RC)), ist die Verwendung dieser Funktion nutzlos: Es wird niemals der Wert YES zurückgegeben, weil bei der Returncode-Verarbeitung nicht zwischen Anweisungen und Kommandos unterschieden wird.

Format

STMT-SPINOFF( )

Ergebnistyp

STRING (YES / NO / UNDEFINED)

Eingabeparameter

Keine

Ergebnis

YES
Für das geladene Programm ist ein Statement-Spinoff ausgelöst worden.

NO
Für das geladene Programm ist kein Statement-Spinoff ausgelöst worden.

UNDEFINED
Es ist kein Programm geladen.

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Die folgende Prozedur startet das Programm SDF-A und öffnet eine Syntaxdatei unbekannten Typs zum Lesen.

```
/DECLARE-PARAMETER NAME=SYNTAX-FILE(INITIAL-VALUE=*PROMPT)
/BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
/   START-SDF-A
//  OPEN-SYNTAX-FILE &(SYNTAX-FILE),TYPE=SYSTEM,MODE=READ
/   IF (STMT-SPINOFF='YES')
//     STEP
//     OPEN-SYNTAX-FILE &(SYNTAX-FILE),TYPE=GROUP(*NO),MODE=READ
/      IF (STMT-SPINOFF='YES')
//        STEP
//        OPEN-SYNTAX-FILE &(SYNTAX-FILE),TYPE=USER(*NO, *NO),MODE=READ
/      END-IF
/   END-IF
//  SHOW-STATUS
/   EXIT-PROCEDURE RESUME-PROGRAM=*YES
/END-BLOCK
```

STRING( ) Ausdruck in String konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen/Stringfunktionen**

Die Funktion STRING( ) konvertiert einen Ausdruck vom Typ INTEGER, BOOLEAN oder STRING in den Typ STRING. Es gelten die Regeln für implizites Konvertieren.

Format

STRING( ) STR( )
EXPRESSION = ausdruck

Ergebnistyp

STRING

Eingabeparameter

EXPRESSION = ausdruck
Bezeichnet den zu konvertierenden Ausdruck.

Ergebnis

Konvertierter Ausdruck als String

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/DECLARE-VARIABLE A,TYPE=*INTEGER     "Datentyp: INTEGER"
/A = 30
/C = STRING(A)
/SHOW-VARIABLE C
C = 30

/D = CURRENT-TYPE('C')
/SHOW-VARIABLE D
D = *STRING     "Datentyp ist nicht mehr Integer, sondern String"
```

SUBCODE1( ) Subcode1 abfragen

Anwendungsgebiet: **Kommando-Returncode**

Die Funktion SUBCODE1( ) liefert die Fehlerklasse des aktuellen Kommando-Returncodes, das heißt des Returncodes des Kommandos, das einen Fehler auslöste oder dem das Kommando SAVE-RETURNCODE folgte.

Kommando-Returncodes bestehen aus drei Komponenten: Subcode1 und Subcode2, die die Fehlerklasse und die Fehlergewichtung anzeigen, und Maincode, der den siebenstelligen Fehlerschlüssel enthält.
Diese Komponenten werden mit den Funktionen SUBCODE1( ), SUBCODE2( ) und MAINCODE( ) ausgewertet. Zu dem Maincode (Fehlerschlüssel) kann mit der Funktion MSG( ) die dazugehörende Meldung ausgegeben werden.

Außerhalb von Prozeduren und Dialogblöcken ist SUBCODE1( ) nicht verfügbar.

Format

SUBCODE1( ) SC1( )

Ergebnistyp

INTEGER (<integer 0..255>)

Eingabeparameter

Keine

Ergebnis

Bezeichnung der Fehlerklasse als Integer-Wert <integer 0..255>

0
In der aktuellen Prozedur ist noch kein Fehler aufgetreten oder der Returncode wurde nach einem fehlerfreien Kommando mit /SAVE-RETURNCODE gesichert.

Fehlermeldungen

```
SDP0428   KOMMANDO-RETURNCODE KANN IM DIALOG NICHT ABGEFRAGT WERDEN
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Fehlerbehandlung mit SUBCODE1( )

```
/DECLARE-VARIABLE MYVAR
/...
/DECLARE-VARIABLE MYVAR    "schon deklariert"
/SAVE-RETURNCODE
/IF ((SUBCODE1=0) AND (SUBCODE2=1))
/   WRITE-TEXT 'Variable ist schon deklariert'
/END-IF
```

SUBCODE2( ) Subcode2 abfragen

Anwendungsgebiet: **Kommando-Returncode**

Die Funktion SUBCODE2( ) liefert die Gewichtung des aktuellen Kommando-Returncodes, das heißt des Returncodes des letzten Kommandos, das einen Fehler auslöste oder dem das Kommando SAVE-RETURNCODE folgte.

Kommando-Returncodes bestehen aus drei Komponenten: Subcode1 und Subcode2, die die Fehlerklasse und die Fehlergewichtung anzeigen, und Maincode, der den siebenstelligen Fehlerschlüssel enthält.
Diese Komponenten werden mit den Funktionen SUBCODE1( ), SUBCODE2( ) und MAINCODE( ) ausgewertet. Zu dem Maincode (Fehlerschlüssel) kann mit der Funktion MSG( ) die dazugehörende Meldung ausgegeben werden.

Außerhalb von Prozeduren und Blöcken ist SUBCODE2( ) nicht verfügbar.

Format

SUBCODE2( ) SC2( )

Ergebnistyp

INTEGER (<integer 0..255>)

Eingabeparameter

Keine

Ergebnis

Wert von Subcode2 als Integer-Wert (<integer 0..255>)

Fehlermeldungen

```
SDP0428   KOMMANDO-RETURN-CODE KANN IM DIALOG NICHT ABGEFRAGT WERDEN
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Fehlerbehandlung mit SUBCODE2( )

```
/DECLARE-VARIABLE MYVAR
/...
/DECLARE-VARIABLE MYVAR     "schon deklariert"
/SAVE-RETURNCODE
/IF ((SUBCODE1=0) AND (SUBCODE2=1))
/   WRITE-TEXT 'Variable ist schon deklariert'
/END-IF
```

SUBLIST( ) Element aus einer SDF-Liste wählen

Anwendungsgebiet: **String-Funktionen**

Die Funktion SUBLIST( ) liefert den Inhalt des gewählten Elements einer SDF-Liste. Eine SDF-Liste ist ein String, der nach den syntaktischen Regeln für Operandenlisten in Kommandos interpretiert wird. Die Auswertung des Strings ist nur dann mit dieser Funktion sinnvoll durchzuführen, wenn er die Form '(<element$_1$>,<element$_2$>,...,<element$_n$>)' aufweist. Dabei sind <element$_i$> Folgen von Zeichen, die kein Komma (außerhalb von Klammerpaaren) enthalten.
Ob eine SDF-Liste vorliegt, kann mit der Funktion IS-SDF-LIST() geprüft werden.

Format

SUBLIST( )
LIST = string_ausdruck ,POSITION = arithm_ausdruck

Ergebnistyp

STRING

Eingabeparameter

LIST = string_ausdruck
Bezeichnet eine SDF-Liste.

POSITION =
Bezeichnet das Element der SDF-Liste, dessen Inhalt ausgegeben werden soll.

POSITION = arithm_ausdruck
Bezeichnet ein Element einer SDF-Liste über seine Elementnummer.
„arithm_ausdruck“ muss eine gültige Elementnummer sein.

Ergebnis

Inhalt des Elements als String

Fehlermeldungen

SDP0411 STRING LEER

SDP0447 DER ANGEGEBENE STRING IST KEINE SDF-LISTE

SDP0448 DER PARAMETER 'NUMBER' AUSSERHALB DES MOEGLICHEN WERTEBEREICHS

Beispiel

```
/A = SUBLIST('(abc,def,jkl,uvw)',3)
/SHOW-VARIABLE A
A = jkl
```

SUBLIST-NUMBER( ) Elementanzahl einer SDF-Liste abfragen

Anwendungsgebiet: **String-Funktionen**

Die Funktion SUBLIST-NUMBER( ) gibt aus, wie viele Elemente eine SDF-Liste enthält. Eine SDF-Liste ist ein String, der nach den syntaktischen Regeln für Operandenlisten in Kommandos interpretiert wird. Die Auswertung des Strings ist nur dann mit dieser Funktion sinnvoll durchzuführen, wenn er die Form '(<element$_1$>,<element$_2$>,...,<element$_n$>)' aufweist. Dabei sind <element$_i$> Folgen von Zeichen, die kein Komma (außerhalb von Klammerpaaren) enthalten.
Ob eine SDF-Liste vorliegt, kann mit der Funktion IS-SDF-LIST( ) geprüft werden.

Format

SUBLIST-NUMBER( )
LIST = string_ausdruck

Ergebnistyp

INTEGER

Eingabeparameter

LIST = string_ausdruck
Bezeichnet eine SDF-Liste.

Ergebnis

Anzahl der Elemente der SDF-Liste als Integer-Wert

Fehlermeldungen

```
SDP0411   STRING LEER
SDP0447   DER ANGEGEBENE STRING IST KEINE SDF-LISTE
```

Beispiel

```
/A = SUBLIST-NUMBER('(abc,def,jkl,uvw)')
/SHOW-VARIABLE A
A = 4
```

SUBSTRING( ) Teilstring ausgeben

Anwendungsgebiet: **String-Funktionen**

Die Funktion SUBSTRING( ) extrahiert einen Teilstring aus dem angegebenen String. Über die Eingabeparameter werden die Anfangsposition des Teilstrings und seine Länge bestimmt.

Format

SUBSTRING( ) SUBSTR( )
STRING = string_ausdruck ,START = 1 / arithm_ausdruck$_1$,LENGTH = *REST-LENGTH / arithm_ausdruck$_2$

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den Eingabestring, aus dem ein Teilstring extrahiert werden soll.

START = 1 / arithm_ausdruck$_1$
Bezeichnet die Anfangsposition des Teilstrings, das heißt das erste Zeichen des Teilstrings; „arithm_ausdruck$_1$“ ist eine positive Zahl vom Typ INTEGER und muss kleiner sein als die Länge des Eingabestrings. Voreinstellung für „arithm_ausdruck$_1$“ ist 1. Bezeichnet „arithm_ausdruck$_1$“ keine gültige Anfangsposition im Eingabestring, wird der Leerstring zurückgegeben.

LENGTH =
Länge des Teilstrings

LENGTH = *REST-LENGTH
Bezeichnet implizit die Länge des Teilstrings als Restlänge ab der mit START angegebenen Zeichenposition (Stringlänge - „arithm_ausdruck$_1$“ + 1 = Restlänge).

LENGTH = arithm_ausdruck$_2$
Bezeichnet explizit die Länge des Teilstrings; ist die Längenangabe zu groß, gilt implizit LENGTH = *REST-LENGTH.

Ergebnis

Teilstring des Eingabestrings in der Länge LENGTH

Leerstring ('') bedeutet:
START = „arithm_ausdruck$_1$“ war keine gültige Anfangsposition, oder es galt implizit oder explizit LENGTH = 0.

Fehlermeldungen

```
SDP0412   START-POSITION AUSSERHALB MOEGLICHEN BEREICHS
SDP0414   WARNUNG: *REST-LENGTH WERTE FUER OPERAND LENGTH VERWENDET
```

Beispiel

```
/A = 'ABCDEFGH'
/B = SUBSTRING(STRING = A, START = 2, LENGTH = 4)
/SHOW-VARIABLE B
B = BCDE

/C = 2
/B = SUBSTRING(STRING = A, START = C)
/SHOW-VARIABLE B
B = BCDEFGH

/D = 4
/B = SUBSTRING(STRING = A, LENGTH = D)
/SHOW-VARIABLE B
B = ABCD

/B = SUBSTRING(STRING = A, START = 9)
SDP0412 START-POSITION AUSSERHALB MOEGLICHEN BEREICHS
/SHOW-VARIABLE B
B =
```

Weiteres Beispiel: siehe Funktion VERIFY( ), Seite 539

SYSCMD( ) SYSCMD-Zuweisung abfragen

Anwendungsgebiet: **SYSFILE-Informationen**

Die Funktion SYSCMD( ) liefert den Namen der Datei (alternativ ist auch ein Bibliothekselement oder eine Listenvariable möglich), die der Systemdatei SYSCMD zugewiesen ist. Es kann zwischen der SYSFILE-Umgebung der Prozedur und der SYSFILE-Umgebung der Task gewählt werden.

Format

SYSCMD( )
SYSTEM-FILE-CONTEXT = *OWN / *CALLER

Ergebnistyp

STRING

Eingabeparameter

SYSTEM-FILE-CONTEXT =
Bezeichnet die SYSFILE-Umgebung, siehe auch Seite 82.

SYSTEM-FILE-CONTEXT = ***OWN**
SYSFILE-Umgebung ist die der Prozedur.

SYSTEM-FILE-CONTEXT = *CALLER
SYSFILE-Umgebung ist die der Task des Aufrufers.

Ergebnis

Das Format der Ausgabe entspricht der Ausgabe des Kommandos /SHOW-SYSTEM-FILE-ASSIGNMENT (siehe Handbuch „Kommandos, Bd. 1-5“ [3]). Wird SYSCMD aus einer Prozedur gelesen (d.h. SYSCMD ist einer Datei, einem Bibliothekselement oder einer Listenvariablen zugewiesen), wird zusätzlich die Art des Prozeduraufrufs angezeigt (bei /INCLUDE-PROCEDURE mit *INCLUDE*, bei /CALL-PROCEDURE mit *PROCEDURE*).

datei (aufrufart)
Name der Datei, der SYSCMD zugewiesen ist.

**LIB-ELEM(bibliothek,element(version),typ) (aufrufart)*
Bibliothekselement (bezeichnet durch den Namen der Bibliothek, des Elements mit Version und den Elementtyp), dem SYSCMD zugewiesen ist.

**VAR(variable) (aufrufart)*
Listenvariable, der SYSCMD zugewiesen ist.

**PRIMARY*
Für SYSCMD gilt die Primärzuweisung (Datenstation im Dialog bzw. SPOOLIN-Datei im Stapelbetrieb).

**PRIMARY (DIALOG-BLOCK)*
Für SYSCMD gilt die Primärzuweisung (wie **PRIMARY*, aber die Abfrage erfolgte in einem Dialogblock).

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiele

Im Dialog:

```
/C = SYSCMD()
/SHOW-VARIABLE C
C = *PRIMARY
```

Im Dialogblock:

```
/begin-block
%BEGIN-BLOCK/a=syscmd()
%BEGIN-BLOCK/show-variable a
%BEGIN-BLOCK/end-block
A = *PRIMARY (DIALOG-BLOCK)
```

In Prozeduren:

Die Datei C.PROC und die Listenvariable PROC-1 enthälten jeweils folgende Kommandos:

```
/A = SYSCMD()
/SHOW-VARIABLE A
```

Aufrufe:

```
/CALL-PROCEDURE C.PROC
```

```
/INCLUDE-PROCEDURE *VAR(PROC-1)
```

Ausgaben:

```
A = :2OSG:$USER1.C.PROC (PROCEDURE)
```

```
A = *VAR(PROC-1) (INCLUDE)
```

SYSDTA( ) SYSDTA-Zuweisung abfragen

Anwendungsgebiet: **SYSFILE-Informationen**

Die Funktion SYSDTA( ) liefert den Namen der Datei (alternativ ist auch ein Bibliothekselement oder eine Listenvariable möglich), die der Systemdatei SYSDTA zugewiesen ist.

Format

SYSDTA( )

Ergebnistyp

STRING

Eingabeparameter

Keine

Ergebnis

Das Format der Ausgabe entspricht der Ausgabe des Kommandos /SHOW-SYSTEM-FILE-ASSIGNMENT (siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

datei
Name der Datei, der SYSDTA zugewiesen ist.

**LIB-ELEM(bibliothek,element(version),typ)*
Bibliothekselement (bezeichnet durch den Namen der Bibliothek, des Elements mit Version und den Elementtyp), dem SYSDTA zugewiesen ist.

**VAR(variable)*
Listenvariable, der SYSDTA zugewiesen ist.

**PRIMARY*
Für SYSDTA gilt die Primärzuweisung (Datenstation im Dialog bzw. SPOOLIN-Datei im Stapelbetrieb).

**SYSCMD*
Wenn SYSDTA explizit der Systemdatei SYSCMD zugewiesen wurde.

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiele

Im Dialog:

```
/A = SYSDTA()
/SHOW-VARIABLE A
A = *PRIMARY
```

In der Prozedur:

Die Prozedur C.PROC enthält folgende Kommandos:

```
/A = SYSDTA()
/SHOW-VARIABLE A
```

Beim Prozedurablauf wird folgende Zeile ausgegeben:

```
A = *SYSCMD
```

In S-Prozeduren ist dies der Defaultwert für SYSDTA.

Ausgabe bei verschiedenen Zuweisungen:

– SYSDTA wird einer Datei zugewiesen

```
/ASSIGN-SYSDTA TO=TEST.EINGABE-DATEN.1
/A = SYSDTA()
/SHOW-VARIABLE A
A = :2OSG:$USER1.TEST.EINGABE-DATEN.1
```

– SYSDTA wird einem Bibliothekselement zugewiesen

```
/ASSIGN-SYSDTA TO=*LIB-ELEM(LIB=ASS.PLAMLIB,ELEM=TEST.DTA.1,TYPE=S)
/A = SYSDTA()
/SHOW-VARIABLE A
A = *LIB-ELEM(:2OSG:$USER1.ASS.PLAMLIB,TEST.DTA.1(*UPPER-LIMIT),S)
```

– SYSDTA wird einer Listenvariablen zugewiesen

```
/ASSIGN-SYSDTA TO=*VARIABLE(DATEN-1)
/A = SYSDTA()
/SHOW-VARIABLE A
A = *VAR(DATEN-1)
```

SYS-ID( ) Systemkennzeichen abfragen

Anwendungsgebiet: **System-Informationen**

Die Funktion SYS-ID( ) liefert das Systemkennzeichen, das heißt die System-ID des laufenden Systems.

Format

SYS-ID( )

Ergebnistyp

STRING (<string 1..4>)

Eingabeparameter

Keine

Ergebnis

Systemkennzeichen als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = SYS-ID()
/SHOW-VARIABLE A
A = 160
```

SYSLST( ) SYSLST-Zuweisung abfragen

Anwendungsgebiet: **SYSFILE-Informationen**

Die Funktion SYSLST( ) liefert den Namen der Datei (alternativ ist auch ein Bibliothekselement oder eine Listenvariable möglich), die der Systemdatei SYSLST zugewiesen ist.

Format

SYSLST( )

Ergebnistyp

STRING

Eingabeparameter

Keine

Ergebnis

Das Format der Ausgabe entspricht der Ausgabe des Kommandos /SHOW-SYSTEM-FILE-ASSIGNMENT (siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

datei
Name der Datei, der SYSLST zugewiesen ist.

**DUMMY*
SYSLST ist einer Pseudodatei zugewiesen.

**LIB-ELEM(bibliothek,element(version),typ)*
Bibliothekselement (bezeichnet durch den Namen der Bibliothek, des Elements mit Version und den Elementtyp), dem SYSLST zugewiesen ist.

**VAR(variable)*
Listenvariable, der SYSLST zugewiesen ist.

**PRIMARY*
Für SYSLST gilt die Primärzuweisung (temporäre SPOOLOUT-Datei (EAM-Datei)).

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Ausgabe bei verschiedenen Zuweisungen:

– SYSLST wird einer Datei zugewiesen

```
/ASSIGN-SYSLST TO=PROTOKOLL.1
/A = SYSLST()
/SHOW-VARIABLE A
A = :2OSG:$USER1.PROTOKOLL.1
```

– SYSLST wird der Pseudodatei zugewiesen

```
/ASSIGN-SYSLST TO=*DUMMY
/A = SYSLST()
/SHOW-VARIABLE A
A = *DUMMY
```

– SYSLST wird einem Bibliothekselement zugewiesen

```
/ASSIGN-SYSLST TO=*LIB-ELEM(LIB=ASS.PLAMLIB,ELEM=PROTOKOLL.1)
/A = SYSLST()
/SHOW-VARIABLE A
A = *LIB-ELEM(:2OSG:$USER1.ASS.PLAMLIB,PROTOKOLL.1(*UPPER-LIMIT),P)
```

– SYSLST wird einer Listenvariablen zugewiesen

```
/ASSIGN-SYSLST TO=*VARIABLE(PROT-1)
/A = SYSLST()
/SHOW-VARIABLE A
A = *VAR(PROT-1)
```

SYSOUT( ) SYSOUT-Zuweisung abfragen

Anwendungsgebiet: **SYSFILE-Informationen**

Die Funktion SYSOUT( ) liefert den Namen der Datei (alternativ ist auch ein Bibliothekselement oder eine Listenvariable möglich), die der Systemdatei SYSOUT zugewiesen ist.

Format

SYSOUT( )

Ergebnistyp

STRING

Eingabeparameter

Keine

Ergebnis

Das Format der Ausgabe entspricht der Ausgabe des Kommandos /SHOW-SYSTEM-FILE-ASSIGNMENT (siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

datei
Name der Datei, der SYSOUT zugewiesen ist.

**DUMMY*
SYSOUT ist einer Pseudodatei zugewiesen.

**LIB-ELEM(bibliothek,element(version),typ)*
Bibliothekselement (bezeichnet durch den Namen der Bibliothek, des Elements mit Version und den Elementtyp), dem SYSOUT zugewiesen ist.

**VAR(variable)*
Listenvariable, der SYSOUT zugewiesen ist.

**PRIMARY*
Für SYSOUT gilt die Primärzuweisung (Datenstation im Dialog bzw. SPOOLOUT-Datei (S.OUT-Datei) im Stapelbetrieb).

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

Ausgabe bei verschiedenen Zuweisungen:

– SYSOUT wird einer Datei zugewiesen

```
/ASSIGN-SYSOUT TO=OUT.LOG.1
/A = SYSOUT()
/SHOW-VARIABLE A
A = :2OSG:$USER1.OUT.LOG.1
```

– SYSOUT wird der Pseudodatei zugewiesen

```
/ASSIGN-SYSOUT TO=*DUMMY
/A = SYSOUT()
/SHOW-VARIABLE A
A = *DUMMY
```

– SYSOUT wird einem Bibliothekselement zugewiesen

```
/ASSIGN-SYSOUT TO=*LIB-ELEM(LIB=ASS.PLAMLIB,ELEM=OUT.LOG.1)
/A = SYSOUT()
/SHOW-VARIABLE A
A = *LIB-ELEM(:2OSG:$USER1.ASS.PLAMLIB,OUT.LOG.1(*UPPER-LIMIT),P)
```

– SYSOUT wird einer Listenvariablen zugewiesen

```
/ASSIGN-SYSOUT TO=*VARIABLE(LOG-1)
/A = SYSOUT()
/SHOW-VARIABLE A
A = *VAR(LOG-1)
```

SYSTEM-CALL( ) Kommandoquelle ausgeben

Anwendungsgebiet: **Prozedurinformationen**

Die Funktion SYSTEM-CALL( ) gibt innerhalb einer Prozedur an, die als Implementor eines Kommandos aufgerufen wurde, ob das Kommando aus der System- oder Gruppensyntaxdatei stammt (siehe dazu auch Handbuch „SDF-A“ [16]).

Format

SYSTEM-CALL( )

Ergebnistyp

BOOLEAN

Eingabeparameter

Keine

Ergebnis

TRUE
Die Prozedur wurde vom System aufgerufen, d.h. der Prozeduraufruf ist als Kommando in einer System- bzw. Gruppensyntaxdatei implementiert (mit der SDF-A-Anweisung //ADD-CMD und IMPLEMENTOR=*PROCEDURE, siehe dazu Handbuch „SDF-A“ [16]).

FALSE
Der Prozeduraufruf erfolgte explizit über das Kommando CALL- bzw. INCLUDE-PROCEDURE oder über einen der Prozeduraufrufe, der als Kommando in der Benutzer-Syntaxdatei implementiert ist.

Hinweis

Die Angabe /IF (SYSTEM-CALL( ) AND NOT EXPLICIT-CALL( )) ist überflüssig, da nur die erste Überprüfung mit SYSTEM-CALL( ) notwendig ist.

Fehlermeldung

Keine

Beispiel

```
/SET-PROCEDURE-OPTIONS "Procedure MYPROC"
/
/WRITE-TEXT 'Systemaufruf: &(SYSTEM-CALL)'
/EXIT-PROCEDURE

/CALL-PROCEDURE MYPROC
Systemaufruf: TRUE

/MY-COMMAND MYPROC   "Kommando aus Benutzer-Syntaxdatei mit Implementierung"
/                    "der Prozedur MYPROC"
Systemaufruf: FALSE

/A-GROUP-COMMAND MYPROC   "Kommando aus Gruppen-Syntaxdatei mit "
/                         "Implementierung der Prozedur MYPROC"
Systemaufruf: TRUE
```

SYSTEM-INFORMATION( ) Systemwerte abfragen

Anwendungsgebiet: **Umgebungsinformationen**

Mit der Funktion SYSTEM-INFORMATION( ) können Systeminformationen und Systemparameter abgefragt werden. Pro Aufruf kann ein Wert abgefragt werden.

Einschränkung

Die Funktion SYSTEM-INFORMATION( ) entspricht auf Programmebene dem Makroaufruf SINF (siehe Handbuch „Makroaufrufe an den Ablaufteil“ [7]), der nur noch kompatibel unterstützt wird und durch die Makroaufrufe NSIINF und NSIOPT ersetzt wurde. Deshalb könne Systeminformationen bzw. Systemparameter, die erst nach Ersetzung dieses Makroaufrufes eingeführt wurden, mit der Funktion SYSTEM-INFORMATION( ) **nicht** abgefragt werden. Die abfragbaren Werte sind in der „Übersicht der möglichen Parameterwerte“ auf Seite 512 aufgelistet.

Die aktuell vorhandenen Systeminformationen und Systemparameter können auf Kommandoebene mit den Kommandos SHOW-SYSTEM-INFORMATION und SHOW-SYSTEM-PARAMETERS abgefragt werden. Diese Kommandos unterstützen auch die strukturierte Ausgabe in S-Variable (siehe Handbuch „Kommandos, Band 6“ [4]).

Format

SYSTEM-INFORMATION( ) SYS-INF( )
INFORMATION = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

INFORMATION = string_ausdruck
Bezeichnet den Namen einer Systeminformation oder eines Systemparameters. Es können nur Namen angegeben werden, die auch von dem Makroaufruf SINF unterstützt werden (mögliche Werte siehe Übersicht auf Seite 512).

Ergebnis

Der Wert der Systeminformation oder des Systemparameters wird als String zurückgegeben.

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFÜGBAR
```

Übersicht der möglichen Parameterwerte

Die möglichen Parameterwerte entsprechen den Werten, die im Operanden INFO des Makros SINF angegeben werden können. Die abfragbaren Systemparameter und Systeminformationen werden in den folgenden zwei Tabellen in gekürzter Form aufgelistet.
Die nicht-privilegierten Systemparameter sind beim Kommando SHOW-SYSTEM-PARAMETERS im Handbuch „Kommandos, Band 5“ [3] beschrieben; eine vollständige Beschreibung aller Systemparameter enthält das Handbuch „Systembetreuung“ [8]. Im Handbuch „Makroaufrufe an den Ablaufteil“ [7] werden die Systeminformationen beschrieben.

Systemparameter

System-parameter	**Bedeutung**
BLKCTRL	Defaultwert für das Dateiattribut BLKCTRL
BLSCOPYN	Defaultwert für den Operanden COPYRIGHT bei dem Dienstprogramm BINDER.
BLSCOPYR	Defaultwert für den Operanden COPYRIGHT bei dem Dienstprogramm TSOSLNK.
DEFLUID	Systemstandardkennung
DMCMAXP	Maximalzahl der Einträge im MRS-Katalog des Home-Pubsets.
DUMPCL5P	Informiert darüber, ob in den von CDUMP ausgegebenen User- oder Areadumps der privilegierte Klasse-5-Speicher enthalten ist.
DUMPSEPA	Informiert über die Ausgabe von Secret Pages in User- oder Systemdumps.
ENCRYPT	Informiert, ob Kennworte (Schutzworte) systemintern verschlüsselt werden.
SECSTART	Informiert darüber, ob der sichere Systemstart ein- oder ausgeschaltet ist.
SECSTENF	Informiert darüber, ob die Systemeinleitung abgebrochen wird, wenn REPs nicht vollständig protokolliert werden können.
SHUTARCH	Informiert, ob bei Einleitung des SHUTDOWN geprüft wird, ob das Programm ARCHIVE noch benützt wird.
SSMLGOF1	Informiert, wie der Spoolout der Systemdateien SYSLST, SYSOPT, SYSOUT bei Auftragsbeendigung abgewickelt wird.
SSMLGOF2	Informiert, ob bei Spoolout von Systemdateien Meldungen ausgegeben werden.
SVC79	Informiert über Einschränkungen für die Anwendung des SVC79 (Wechsel vom nichtprivilegierten (TU) in den privilegierten Systemzustand (TPR)).
TEMPFILE	Kennzeichen für temporäre Dateien bzw. Jobvariablen (#,/,@ oder NO)

Systeminformation

System-information	Bedeutung
CONFNAME	Anlagentyp (Modellreihe) z.B.: S150-40 (im alten Format).
CONFNAMX	Anlagentyp (Modellreihe) im neuen erweiterten Format, z.B.: 7.500- S150-40
CPUID	Identifikationen der CPUs. Die Ausgabe gliedert sich in 8 Elemente von jeweils 8 Byte Länge.
CPUSER	Seriennummer (jeweils 6 Zeichen) der ersten, zweiten, dritten und vierten CPU. Ist eine CPU nicht vorhanden, wird in das entsprechende Feld der Wert '000000' eingetragen. Die Angaben haben keine Beziehung zu den CPU-Adressen.
HSIBASE	Der HSI-Basistyp wird ausgegeben.
HSILINE	Zusätzliche Informationen über das HSI-Typ CFCS3 werden angezeigt.
HSITYPE	Eigenschaften des aktuellen HSI-Typs werden angezeigt.
HSIVM	Informiert darüber, ob eine reelle oder eine virtuelle Maschine vorliegt. Mögliche Werte: V2 Das Betriebssystem läuft auf einer virtuellen Maschine unter VM2000. NV Das Betriebssystem läuft auf einer realen Maschine.
MEMSIZE	Größe des für die Software nutzbaren (physikalischen) Hauptspeichers in Byte.
OSAMODE	Informiert über den Adressierungsmodus des Betriebssystems.
OSID	Byte 0-7: Programmname des Betriebssystems; z.B. 'P16BXS'. Byte 8-11: Version, z.B. 'V160'.
SYSBASE	Anfangsadresse des Betriebssystems im virtuellen Adressraum.

Beispiel 1

```
/A = SYSTEM-INFORMATION(INFORMATION='MEMSIZE')
/SHOW-VARIABLE A
A = 1073741824
```

In die Variable A wird die Größe des physikalischen Hauptspeichers, der für die Software nutzbar ist, ausgegeben: 1.073.741.824 Byte

Beispiel 2

Prozedur zur Abfrage mehrerer Systeminformationen bzw. Systemparameter:

```
/DECLARE-VARIABLE VALUE-LIST,TYPE=*STRING,-
/                 INITIAL-VALUE='(-
/BLKCTRL,DEFLUID,OSID,CONFNAMX,CPUID,HSIVM,SSMLGOF1,SSMLGOF2,TEMPFILE)'
/DECLARE-VARIABLE ACT-VALUE,TYPE=*STRING
/DECLARE-VARIABLE I,TYPE=*INTEGER
/FOR  I = *COUNTER(FROM=1,TO=SUBLIST-NUMBER(VALUE-LIST))
/  ACT-VALUE = SUBLIST(VALUE-LIST,I)
/  SHV ACT-VALUE,VALUE=C-LIT
/  WRITE-TEXT '&ACT-VALUE: &(SYSTEM-INFORMATION(ACT-VALUE))'
/  IF-BLOCK-ERROR
/     WRITE-TEXT 'CURRENTLY NO INFORMATION AVAILABLE FOR &ACT-VALUE'
/  END-IF
/END-FOR
```

Ausgabe nach Aufruf der Prozedur:

```
ACT-VALUE = 'BLKCTRL'
BLKCTRL: PAMKEY
ACT-VALUE = 'DEFLUID'
DEFLUID: $TSOS
ACT-VALUE = 'OSID'
OSID: M12BXS  V140
ACT-VALUE = 'CONFNAMX'
CONFNAMX: 7.500- S150-40
ACT-VALUE = 'CPUID'
CPUID:
300200018800000030120001880000003022000188000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
ACT-VALUE = 'HSIVM'
HSIVM: V2
ACT-VALUE = 'SSMLGOF1'
SSMLGOF1: REQ-SPOOL
ACT-VALUE = 'SSMLGOF2'
SSMLGOF2: YES
ACT-VALUE = 'TEMPFILE'
TEMPFILE: #
```

TASK-MODE( ) Task-Modus abfragen

Anwendungsgebiet: **taskspezifische Umgebungsinformationen**

Liefert den Modus der aktuellen Task.

Format

TASK-MODE( )

Ergebnistyp

STRING (<string 2..6>)

Eingabeparameter

Keine

Ergebnis

BATCH
Die aktuelle Task läuft im Batch-Modus, das heißt als Stapelauftrag asynchron im Hintergrund.

DIALOG
Die aktuelle Task ist eine Dialog-Task.

SYSTEM
Die aktuelle Task ist eine System-Task.

TP
Die aktuelle Task läuft im Modus TP.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = TASK-MODE( )
/SHOW-VARIABLE A
A = DIALOG
```

TIME( ) Uhrzeit abfragen

Anwendungsgebiet: **Umgebungsinformationen** (Kalender/ Zeit)

Die Funktion TIME( ) liefert die aktuelle Uhrzeit; das Trennzeichen zwischen den Angaben von Stunden, Minuten oder Sekunden ist frei wählbar.

Format

TIME( )
SEPARATOR = ':' / zeichen ,MODE = *LOCAL-TIME / *UNIVERSAL-TIME

Ergebnistyp

STRING (<string 8..8>)

Eingabeparameter

SEPARATOR = ':' / zeichen
Bestimmt das Trennzeichen zwischen den einzelnen Zeitangaben; voreingestellt ist der Doppelpunkt (:).
„zeichen“ ist ein beliebiges Zeichen als C-Literal.

MODE = *LOCAL-TIME / *UNIVERSAL-TIME
Bestimmt, ob die Uhrzeit in der lokalen Landeszeit (LOCAL-TIME) oder in der universellen Weltzeit (UNIVERSAL-TIME) ausgegeben wird.
Zu LOCAL-TIME (LT) und UNIVERSAL-TIME (UTC) siehe auch Makro GTIME im Handbuch „Makroaufrufe an den Ablaufteil“ [7].

Ergebnis

hh:mm:ss
Aktuelle Uhrzeit, wobei „hh“ die Stunden, „mm“ die Minuten und „ss“ die Sekunden anzeigen (der Doppelpunkt als Trennzeichen kann durch ein beliebiges Zeichen ersetzt sein).

Fehlermeldung

Keine Fehlermeldungen

Beispiel

Ausgabe der lokalen Zeit und der UTC-Zeit:

```
/A = TIME()
/B = TIME(MODE=*UNIVERSAL-TIME)

/SHOW-VARIABLE (A,B)
A = 10:46:51
B = 09:46:51
```

TO-C-LITERAL( ) String in C-Literal konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion TO-C-LITERAL( ) konvertiert den angegebenen String in ein C-Literal. Dabei wird jeweils an Anfang und Ende des Strings ein Hochkomma gesetzt. Hochkommas innerhalb des Strings werden verdoppelt.
FROM-C-LITERAL( ) ist die inverse Funktion zu TO-C-LITERAL( ).

Format

TO-C-LITERAL( ) TO-C-LIT( )
STRING = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den zu konvertierenden Ausdruck.

Ergebnis

String-Literal als String

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = 'AB''C'
/SHOW-VARIABLE A
A = AB'C
/B = TO-C-LITERAL(STRING = A)
/SHOW-VARIABLE B
B = 'AB''C'

/ADD-PASSWORD &(TO-C-LITERAL(A))
```

Das Kommando /ADD-PASSWORD 'AB"C' wird abgesetzt.

TO-X-LITERAL( ) String in X-Literal konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion TO-X-LITERAL( ) konvertiert den Hexadezimalwert des angegebenen Strings in die externe Darstellung: sie setzt an den Beginn und das Ende des Hexadezimalwerts jeweils ein Hochkomma und stellt außerdem das Zeichen X voran.

FROM-X-LITERAL( ) ist die inverse Funktion zu TO-X-LITERAL( ).

Format

TO-X-LITERAL( ) TO-X-LIT( )
STRING = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den String, der konvertiert werden soll.

Ergebnis

X-Literal als String

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/T = 'HALLO'
/B = TO-X-LITERAL(T)
/SHOW-VARIABLE B
B = X'C8C1D3D3D6'
```

TRANSLATE( ) Eingangswerten beliebige Ergebniswerte zuordnen

Anwendungsgebiet: **Konvertierungsfunktionen**

Mit der Funktion TRANSLATE( ) lassen sich beliebigen Eingangswerten beliebige Ergebniswerte zuordnen. Es dürfen bis zu zehn Umsetzungen angegeben werden, wobei WHEN- und THEN-Klauseln paarig auftreten müssen.

Ablauf: Die in den WHEN-Klauseln angegebenen Vergleichsstrings werden nacheinander auf Gleichheit mit dem Eingangsstring geprüft. Bei Gleichheit wird als Ergebnis der in der zugehörigen THEN-Klausel angegebene Ergebnisstring geliefert. Stimmt keiner der in den WHEN-Klauseln genannten Vergleichsstrings mit dem Eingangsstring überein, wird der im ELSE-Zweig angegebene String als Ergebnis geliefert.

Format

TRANSLATE( )

STRING = string_ausdruck$_0$
,WHEN1 = *NONE / string_ausdruck$_1$, THEN1 = *NONE / *SAME / string_ausdruck$_{11}$
,WHEN2 = *NONE / string_ausdruck$_2$, THEN2 = *NONE / *SAME / string_ausdruck$_{12}$
,WHEN3 = *NONE / string_ausdruck$_3$, THEN3 = *NONE / *SAME / string_ausdruck$_{13}$
,WHEN4 = *NONE / string_ausdruck$_4$, THEN4 = *NONE / *SAME / string_ausdruck$_{14}$
,WHEN5 = *NONE / string_ausdruck$_5$, THEN5 = *NONE / *SAME / string_ausdruck$_{15}$
,WHEN6 = *NONE / string_ausdruck$_6$, THEN6 = *NONE / *SAME / string_ausdruck$_{16}$
,WHEN7 = *NONE / string_ausdruck$_7$, THEN7 = *NONE / *SAME / string_ausdruck$_{17}$
,WHEN8 = *NONE / string_ausdruck$_8$, THEN8 = *NONE / *SAME / string_ausdruck$_{18}$
,WHEN9 = *NONE / string_ausdruck$_9$, THEN9 = *NONE / *SAME / string_ausdruck$_{19}$
,WHEN10 = *NONE / string_ausdruck$_{10}$, THEN10 = *NONE / *SAME / string_ausdruck$_{20}$
,ELSE = *NONE / *SAME / string_ausdruck$_{21}$

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck$_0$
Bezeichnet den Eingabestring, mit dem die Vergleichsstrings in den WHEN-Klauseln verglichen werden.

WHENi = *NONE
$1 \le i \le 10$; bezeichnet den Leerstring.

WHENi = string_ausdruck$_i$
$1 \le i \le 10$; bezeichnet die Vergleichsstrings und bestimmt die Umsetzungsbedingungen: Wenn ein „string_ausdruck$_i$“ mit dem Eingabestring übereinstimmt, wird der entsprechende THENi-Zweig ausgeführt, das heißt, der dort bezeichnete Ergebnisstring geliefert.

THENi = *NONE
$1 \le i \le 10$; bezeichnet den Leerstring.

THENi = *SAME
$1 \le i \le 10$; bestimmt, dass bei Gleichheit von Eingabestring und Vergleichsstring als Ergebnis der Eingabestring ausgegeben wird.

THENi = string_ausdruck$_j$
$1 \le i \le 10$; $11 \le j \le 20$; bezeichnet den Ergebnisstring, der bei Gleichheit von Eingabestring (STRING =) und Vergleichsstring (WHENi =) als Ergebnis ausgegeben wird.

ELSE = *NONE
Wenn keiner der Vergleichsstrings in den WHENi-Klauseln mit dem Eingabestring übereinstimmt, wird als Ergebnis der Leerstring ausgegeben.

ELSE = *SAME
Wenn keiner der Vergleichsstrings in den WHENi-Klauseln mit dem Eingabestring übereinstimmt, wird als Ergebnis der Eingabestring ausgegeben.

ELSE = string_ausdruck$_{21}$
Bezeichnet den Ergebnisstring, der ausgegeben wird, wenn keiner der Vergleichsstrings in den WHENi-Klauseln mit dem Eingabestring übereinstimmt.

Ergebnis

Ergebnisstring aus THENi oder ELSE-Klausel

Fehlermeldung

```
SDP0410   INKONSISTENZ ZWISCHEN 'WHEN' UND 'THEN' PARAMETERN
```

Beispiel

```
/A = 'ABC'
/C = TRANSLATE(STRING = A,-
/,WHEN1 = 'AB', THEN1 = '12'-
/,WHEN2 = 'BC', THEN2 = '23'-
/,WHEN3 = 'ABC', THEN3 = '123'-
/,ELSE = 'NIX_PASST')
/SHOW-VARIABLE C
C = 123
```

```
/ B = 'grasgruen'
/ D = TRANSLATE(STRING = B,-
/,WHEN1 = 'gruen', THEN1 = 'green'-
/,WHEN2 = 'rot', THEN2 = 'red'-
/,WHEN3 = 'blau', THEN3 = 'blue'-
/,ELSE = 'NIX_PASST')
/SHOW-VARIABLE D
D = NIX_PASST
```

TRANSLATE-BOOLEAN( ) Booleschen Ausdruck prüfen

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion TRANSLATE-BOOLEAN( ) prüft, ob der Eingabe-Ausdruck wahr oder falsch ist. Ist er wahr, wird der in der THEN-Klausel angegebene Wert als Ergebnis geliefert. Ist der Eingabe-Ausdruck nicht wahr (=falsch), wird der in der ELSE-Klausel angegebene Wert als Ergebnis geliefert.

Format

TRANSLATE-BOOLEAN( )
IF = ausdruck$_1$,THEN = ausdruck$_2$,ELSE = ausdruck$_3$

Ergebnistyp

BOOLEAN / INTEGER / STRING

Eingabeparameter

IF = ausdruck$_1$
Bezeichnet einen Ausdruck vom Typ BOOLEAN.

THEN = ausdruck$_2$
Bezeichnet einen Ausdruck vom Typ BOOLEAN, INTEGER oder STRING.

ELSE = ausdruck$_3$
Bezeichnet einen Ausdruck vom Typ BOOLEAN, INTEGER oder STRING.

Ergebnis

- String, wenn der Ausdruck in THEN-/ELSE-Klausel ein String-Ausdruck ist.
- Integer-Wert, wenn der Ausdruck in THEN-/ELSE-Klausel ein arithmetischer Ausdruck ist.
- TRUE / FALSE, wenn der Ausdruck in THEN-/ELSE-Klausel ein boolescher Ausdruck ist.

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = 6
/B = 5
/C = TRANSLATE-BOOLEAN(IF=(A > B), THEN='A groesser', ELSE='B groesser')
/SHOW-VARIABLE C
C = A groesser
```

```
/A = 5
/B = 6
/C = TRANSLATE-BOOLEAN(IF=(A > B), THEN='A groesser', ELSE='B groesser')
/SHOW-VARIABLE C
C = B groesser
```

TRIM( ) Gleiche Zeichen am Anfang oder Ende entfernen

Anwendungsgebiet: **String-Funktionen**

Die Funktion TRIM( ) entfernt gleiche Zeichen am Anfang oder Ende oder an beiden Seiten eines Strings.

Format

TRIM( )
STRING = string_ausdruck ,SIDE = *BOTH / *LEFT / *RIGHT ,TRIM-BYTE = C'␣' / zeichen

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet einen Ausdruck vom Typ STRING, der bearbeitet werden soll.

SIDE = *BOTH / *LEFT / *RIGHT
Das im Parameter TRIM-BYTE angegebene Zeichen wird am Anfang des Strings (*LEFT), an seinem Ende (*RIGHT) oder an beiden Seiten (*BOTH) so oft entfernt, bis ein anderes Zeichen auftritt.

TRIM-BYTE = C'␣' / zeichen
Bezeichnet das Zeichen (als C-Literal), das entfernt werden soll. Voreinstellung: Blank (Leerzeichen). Blank wird auch angenommen, wenn ein Nullstring (C'') eingegeben wird.

Ergebnis

String in gekürzter Form

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = '  ABC   '
/B = '  ABCOOO'

/A = TRIM(STRING=A)
/B = TRIM(STRING=B,SIDE=*LEFT,TRIM-BYTE=' ')
/B = TRIM(STRING=B,SIDE=*RIGHT,TRIM-BYTE='O')

/SHOW-VARIABLE
A = ABC
B = ABC
```

TSN( ) TSN abfragen

Anwendungsgebiet: **Job-Informationen**

Die Funktion TSN( ) liefert die Auftragsnummer (= Task Sequence Number) des aktuellen Auftrags.

Format

TSN( )

Ergebnistyp

STRING (<string 4..4>)

Eingabeparameter

Keine

Ergebnis

Auftragsnummer als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = TSN()
/SHOW-VARIABLE A
A = 29XX
```

Zum Vergleich die Auftragsnummer im Feld TSN (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:     29XX       TYPE:    3 DIALOG   NOW:     2007-04-26.110747
%JOBNAME: BERTA      PRI:     0 210
%USERID:  USER1      JCLASS:  JCDSTD     LOGON:   2007-04-26.1053
%ACCNB:   ACC01      CPU-MAX:   9999     CPU-USED:000000.6447
...
```

UPPER-CASE( ) Kleinbuchstaben in Großbuchstaben umsetzen

Anwendungsgebiet: **String-Funktionen/Konvertierungsfunktionen**

Die Funktion UPPER-CASE( ) setzt alle Kleinbuchstaben im angegebenen String in Großbuchstaben um.

Die umzusetzenden Buchstaben müssen dem Standard-EBCDI-Code entsprechen. Es werden keine sprachspezifischen Varianten unterstützt.

Format

UPPER-CASE( )
STRING =string_ausdruck ,TRANSLATE = *ALL / *OUTSIDE-QUOTES-ONLY / *INSIDE-QUOTES-ONLY

Ergebnistyp

STRING

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den umzusetzenden String.

TRANSLATE =
Bezeichnet, welche Zeichen umzusetzen sind.

TRANSLATE = *ALL
Bezeichnet, dass alle Zeichen umzusetzen sind.

TRANSLATE = *OUTSIDE-QUOTES-ONLY
Bezeichnet, dass nur die Zeichen außerhalb der Hochkommas umzusetzen sind.

TRANSLATE = *INSIDE-QUOTES-ONLY
Bezeichnet, dass nur die Zeichen innerhalb der Hochkommas umzusetzen sind.

Ergebnis

Nur aus Großbuchstaben, Ziffern und Sonderzeichen bestehender String

Fehlermeldung

Keine Fehlermeldungen

Beispiel

```
/A = 'abcd123' // 'gHI'
/B = UPPER-CASE(STRING = A)
/SHOW-VARIABLE B
B = ABCD123GHI
```

USER-IDENTIFICATION( ) Benutzerkennung abfragen

Anwendungsgebiet: **Job-Informationen**

Die Funktion USER-IDENTIFICATION( ) liefert die Benutzerkennung des aktuellen Auftrags, das heißt die Benutzerkennung aus dem SET-LOGON-PARAMETERS-Kommando.

Format

USER-IDENTIFICATION( ) USER-ID( )

Ergebnistyp

STRING (<string 1..8>)

Eingabeparameter

Keine

Ergebnis

Benutzerkennung als String

Fehlermeldung

```
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR
```

Beispiel

```
/A = USER-IDENTIFICATION( )
/SHOW-VARIABLE A
A = USER1
```

Zum Vergleich die Kennung im Feld USERID (Ausgabeformat BS2000/OSD-BC V7.0):

```
/show-job-status
%TSN:     29XX       TYPE:    3 DIALOG   NOW:     2007-04-26.110747
%JOBNAME: BERTA      PRI:     0 210
%USERID:  USER1      JCLASS:  JCDSTD     LOGON:   2007-04-26.1053
%ACCNB:   ACC01      CPU-MAX:   9999     CPU-USED:000000.6447
...
```

USER-SWITCH( ) Benutzerschalter auswerten

Anwendungsgebiet: **Job-Informationen**

Die Funktion USER-SWITCH( ) prüft den Wert des angegebenen Benutzerschalters.

Benutzerschalter werden zum Beispiel zur Synchronisation von Stapelaufträgen eingesetzt, das heißt in Hintergrund-Prozeduren. Jeder Benutzerkennung stehen 32 Benutzerschalter zur Verfügung, die für alle unter der Benutzerkennung laufenden Aufträge gelten. Das heißt, Benutzerschalter, die in einem Auftrag gesetzt werden, können in einem anderen Auftrag, der unter der gleichen Benutzerkennung läuft, ausgewertet werden. Gesetzt werden Benutzerschalter mit dem Kommando MODIFY-USER-SWITCHES.

Format

USER-SWITCH( )
NUMBER = zahl ,USER-ID = *OWN / <string 1..8>

Ergebnistyp

BOOLEAN

Eingabeparameter

NUMBER = zahl
0 ≤ zahl ≤ 31; bezeichnet den Benutzerschalter, der ausgewertet wird.

USER-ID = *OWN
Bezeichnet die eigene Benutzerkennung.

USER-ID = <string 1..8>
Bezeichnet die Benutzerkennung, zu der der abzufragende Benutzerschalter gehört.

Ergebnis

TRUE
Der angegebene Benutzerschalter hat den Wert 'ON', ist also „eingeschaltet“.

FALSE
Der angegebene Benutzerschalter hat den Wert 'OFF', ist also „ausgeschaltet“.

Fehlermeldung

```
SDP0304   UEBERLAUF, ZAHL AUSSERHALB DES MOEGLICHEN WERTEBEREICHS
```

Beispiel

Im Dialog:

```
/MODIFY-USER-SWITCHES ON = 1
/B = USER-SWITCH(1, USER-ID = 'USER1')
/SHOW-VARIABLE B
B = TRUE
```

Innerhalb einer Prozedur werden die Benutzerschalter 1, 3, 5 und 8 eingeschaltet und die Benutzerschalter 2 und 4 ausgeschaltet:

```
/MODIFY-USER-SWITCHES ON = (1,3,5,8), OFF = (2,4)
```

Später werden Benutzerschalter abgefragt:

```
/A = USER-IDENTIFICATION()
/IF (USER-SWITCH(2, USER-ID = A))
/   CALL-PROCEDURE C.PROC1.2
/END-IF
/...
/IF (USER-SWITCH(1, USER-ID = A))
/   CALL-PROCEDURE C.PROC1.1
/ELSE-IF USER-SWITCH(4, USER-ID = A)
/   CALL-PROCEDURE C.PROC1.4
/ELSE
/   CALL-PROCEDURE C.PROC2
/END-IF
```

Benutzerschalter 2 ist nicht gesetzt (OFF), die Bedingung im ersten IF-Kommando also nicht erfüllt. Daher wird PROC1.2 nicht aufgerufen, sondern gleich das auf END-IF folgende Kommando ausgeführt.

Im zweiten IF-Kommando wird der Benutzerschalter 1 geprüft; dieser ist gesetzt (ON) und daher ist die Bedingung erfüllt. Die Prozedur C.PROC1.1 wird aufgerufen.

VARIABLE-ATTRIBUTE( ) Variableneigenschaften abfragen

Anwendungsgebiet: **Variablenzugriff (Variableneigenschaften)**

Die Funktion VARIABLE-ATTRIBUTE( ) liefert für die angegebene Variable den Wert des angegebenen Attributs. Attribute sind die Variableneigenschaften, die mit dem SDF-P-Kommando DECLARE-VARIABLE definiert werden. Schlüsselwörter für die Abfrage sind dabei in der Regel die Operandennamen des Kommandos.

Detailinformationen über Strukturen müssen über ATTRIBUTE = *STRUCTURE-INFO (nicht über ATTRIBUTE = *TYPE) abgefragt werden.

Format

VARIABLE-ATTRIBUTE( ) VAR-ATTR( )
VARIABLE-NAME = string_ausdruck ,ATTRIBUTE = *TYPE / *CONTAINER / *CONTAINER-NAME / *CONTAINER-SCOPE / *MULTIPLE-ELEMENTS / *SCOPE / *STRUCTURE-INFO

Ergebnistyp

STRING

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Bezeichnet eine Variable. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe dazu das folgende Beispiel sowie das letzte Beispiel bei IS-DECLARED( )).

ATTRIBUTE =
Bezeichnet ein Variablenattribut.

ATTRIBUTE = *TYPE
Liefert den Variablentyp.
Wenn die mit „string_ausdruck“ bezeichnete Variable eine Struktur ist, wird der Wert „*STRUCTURE“ zurückgeliefert.

ATTRIBUTE = *CONTAINER
Liefert den Typ des Variablenbehälters.

ATTRIBUTE = *CONTAINER-NAME
Liefert den Namen des Variablenbehälters.

ATTRIBUTE = *CONTAINER-SCOPE
Liefert den Geltungsbereich des Variablenbehälters.

ATTRIBUTE = *MULTIPLE-ELEMENTS
Liefert den Typ der zusammengesetzten Variablen.

ATTRIBUTE = *SCOPE
Liefert den Geltungsbereich der Variablen.

ATTRIBUTE = *STRUCTURE-INFO
Liefert die Eigenschaften der mit „string_ausdruck“ bezeichneten zusammengesetzten Variablen vom Typ Struktur. Diese Eigenschaften werden im Kommando DECLARE-VARIABLE mit dem Operanden TYPE = *STRUCTURE(DEFINITION = ...) definiert.

Ergebnis

Wert des Attributs als String

Eingabeparameter ATTRIBUTE =	**Ergebnis**
*TYPE	'*ANY' / '*BOOLEAN' / '*INTEGER' / '*STRING' Variablentyp '*STRUCTURE' „string_ausdruck“ bezeichnet eine Struktur.
*CONTAINER	'*STD' / '*VARIABLE' / '*JV' / 'composed-name' Typ des Variablenbehälters
*CONTAINER-NAME	'name' / '' Name der Behältervariable oder Jobvariable wenn die Variable keinen Container hat: Fehlermeldung
*CONTAINER-SCOPE	'*INCLUDE' / '*PROCEDURE' / '*TASK' Geltungsbereich des Variablenbehälters wenn die Variable keinen Container hat: Fehlermeldung
*MULTIPLE-ELEMENTS	'*ARRAY' / '*LIST' Typ der zusammengesetzten Variablen '*NO' „string_ausdruck“ ist weder ein Array noch eine Liste

Eingabeparameter ATTRIBUTE =	**Ergebnis**
*SCOPE	'*INCLUDE' Geltungsbereich der mit „string_ausdruck“ bezeichneten Variablen
*STRUCTURE-INFO	'*BY-SYSCMD' „string_ausdruck“ bezeichnet eine statische Struktur. '*DYNAMIC' „string_ausdruck“ bezeichnet eine dynamische Struktur. 'name' Name des Strukturlayouts

Fehlermeldung

```
SDP0424   VARIABLER '(&00)' KEIN BEHAELTER ZUGEWIESEN
SDP0425   BUILTIN-FUNKTION VAR-ATTRIBUTES: VARIABLE IST KEINE STRUKTUR
SDP1007   NOCH KEINE VARIABLE ANGELEGT
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN
```

Beispiel 1

```
/BEGIN-STRUCTURE PERSON
...
/END-STRUCTURE
/DECLARE-VARIABLE A (TYPE = *STRUCTURE(PERSON))
...
/B = VARIABLE-ATTRIBUTE(VARIABLE-NAME = 'A', ATTRIBUTE = *TYPE)
/SHOW-VARIABLE B
B = *STRUCTURE

/B = VARIABLE-ATTRIBUTE(VARIABLE-NAME = 'A', ATTRIBUTE = *STRUCTURE-INFO)
/SHOW-VARIABLE B
B = PERSON
```

Beispiel 2

```
/OPEN-VARIABLE-CONTAINER mycontainer,*LIB(mylibrary)
/DECLARE-VARIABLE myvar, CONTAINER= mycontainer
/A = VARIABLE-ATTRIBUTE('myvar',*CONTAINER)
/SHOW-VARIABLE A
A = MYCONTAINER

/A = VARIABLE-ATTRIBUTE('myvar',*CONTAINER-NAME)
/SHOW-VARIABLE A
A = "leerer String"
```

VARIABLE-TO-STRING( ) Variable konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion VARIABLE-TO-STRING( ) konvertiert eine S-Variable vom Typ Struktur in einen String (siehe dazu Abschnitt „SDF-Kommando-Strings zu S-Variablen konvertieren und umgekehrt“ auf Seite 186).

Format

VARIABLE-TO-STRING( ) VAR-TO-STR( )
VARIABLE-NAME = string_ausdruck

Ergebnistyp

STRING

Eingabeparameter

VARIABLE-NAME = string_ausdruck
Name der S-Variablen vom Typ Struktur, die in einen String konvertiert werden soll. Der Variablenname muss in Hochkommata eingeschlossen werden, wenn man ihn direkt (als Literal) angeben will (siehe folgendes Beispiel sowie das Beispiel bei IS-DECLARED( ), Seite 417).

Ergebnis

Konvertierter Ausdruck als String

Fehlermeldung

```
SDP0475   VARIABLE MUSS STRUKTUR, LISTE ODER ARRAY SEIN
SDP0476   ERGEBNIS-STRING ZU LANG
SDP0477   FEHLERHAFTE SDF-STRUKTUR
SDP1007   NOCH KEINE VARIABLE ANGELEGT
```

Beispiel

```
/DECLARE-VARIABLE MYSTRUCT(TYPE=*STRUCTURE(DEF=*DYNAMIC))
/MYSTRUCT.OPERAND1.SYSSTRUC = 'VALUE1'
/MYSTRUCT.OPERAND1.OPERAND2 = 'VALUE2'
/MYSTRUCT.OPERAND1.OPERAND3#1 = 'VALUE3'
/WRITE-TEXT &(TO-C-LITERAL(VARIABLE-TO-STRING('MYSTRUCT')))
OPERAND1 = VALUE1(OPERAND2 =VALUE2, OPERAND3 = (VALUE3))
```

VERIFY( ) Strings prüfen

Anwendungsgebiet: **Stringfunktionen**

Die Funktion VERIFY( ) prüft zwei Strings und liefert als Ergebnis die Position des ersten Zeichens im String (STRING), das nicht im Suchstring (PATTERN) enthalten ist. Die Anzahl und Reihenfolge der Zeichen in den einzelnen Strings bleibt dabei unberücksichtigt. Die Suchrichtung ist frei wählbar, die Positionsangabe gilt immer ab Anfang des ersten Strings.

Format

VERIFY( )
STRING = string_ausdruck$_1$,PATTERN = string_ausdruck$_2$,DIRECTION = *FORWARD / *REVERSE

Ergebnistyp

INTEGER

Eingabeparameter

STRING = string_ausdruck$_1$
Bezeichnet den Vergleichsstring, der nach dem bei PATTERN angegebenen Suchstring durchsucht wird.

PATTERN = string_ausdruck$_2$
Bezeichnet den Suchstring.

DIRECTION =
Bezeichnet die Suchrichtung.

DIRECTION = *FORWARD
Die Suche beginnt am Anfang von string_ausdruck$_1$, das heißt, string_ausdruck$_1$ wird von links nach rechts durchsucht.

DIRECTION = *REVERSE
Die Suche beginnt am Ende von string_ausdruck$_1$, das heißt, string_ausdruck$_1$ wird von rechts nach links durchsucht.

Ergebnis

Positiver Integer-Wert, der die Position des ersten Zeichens im Vergleichsstring angibt, das nicht im Suchstring enthalten ist.

0
Alle Zeichen des Vergleichsstrings sind im Suchstring enthalten.

Fehlermeldung

```
SDP0413   NICHT ZULAESSIGE LAENGENANGABE
```

Beispiel 1

```
/A = '314!59'
/B = '0123456789'
/C = VERIFY(STRING = A, PATTERN = B)
/SHOW-VARIABLE C
C = 4
```

Beispiel 2

Das erste Zeichen, das in einer Zeichenkette nicht das Leerzeichen ist, wird gesucht.

```
/A = 'xyz'
/B = VERIFY(STRING = A, PATTERN = '␣')
/SHOW-VARIABLE B
B = 1
```

WILDCARD( ) Muster suchen

Anwendungsgebiet: **Stringfunktionen/Prüffunktionen**

Die Funktion WILDCARD( ) vergleicht einen String mit einem Musterstring. Für den Musterstring gelten die allgemeinen Regeln für Muster im BS2000 (siehe „BS2000-Platzhalter“ auf Seite 557, beim Datentypzusatz „with-wild“) bzw. die allgemeinen Regeln für Muster in POSIX (siehe „POSIX-Platzhalter“ auf Seite 558).

Format

WILDCARD( )
STRING = string_ausdruck1 ,PATTERN = string_ausdruck2 ,WILDCARD-MODE = *BS2000 / *POSIX

Ergebnistyp

BOOLEAN

Eingabeparameter

STRING = string_ausdruck$_1$
Bezeichnet den zu prüfenden String.

PATTERN = string_ausdruck$_2$
Bezeichnet das Muster.

WILDCARD-MODE = *BS2000 / *POSIX
Gibt an, ob Wildcards bei der Ersetzung in der BS2000-Wildcard-Syntax oder in der POSIX-Wildcard-Syntax interpretiert werden.

Ergebnis

TRUE
Der geprüfte String entspricht dem bei PATTERN genannten Muster.

FALSE
Der geprüfte String entspricht nicht dem bei PATTERN genannten Muster.

Fehlermeldung

```
SDP0443   SYNTAX DES MUSTERS IST KEINE KORREKTE WILDCARD-SYNTAX
```

Beispiel

```
/A = 'dies ist der zu pruefende Text. Gesucht werden Umlaute.'
/B = 'ue'
/C = 'ae'
/D = WILDCARD(STRING = A, PATTERN = B)
/SHOW-VARIABLE D
D = FALSE

/B = '*ue*'
/D = WILDCARD(STRING = A, PATTERN = B)
/SHOW-VARIABLE D
D = TRUE
/C = '*ae*'
/D = WILDCARD(STRING = A, PATTERN = C)
/SHOW-VARIABLE D
D = FALSE
```

X-LITERAL-TO-INTEGER( ) String in Zahl konvertieren

Anwendungsgebiet: **Konvertierungsfunktionen**

Die Funktion X-LITERAL-TO-INTEGER( ) konvertiert einen maximal 4 Byte langen String in eine Dezimalzahl. Der Eingabestring kann als X-String oder C-String angegeben werden. Ein Leerstring erhält den Wert 0.

Besteht der Eingabestring aus weniger als 4 Zeichen, wird er von links nach rechts mit X'00' aufgefüllt.

X-LITERAL-TO-INTEGER( ) ist die inverse Funktion zu INTEGER-TO-X-LITERAL( ).

Format

X-LITERAL-TO-INTEGER( ) X-LIT-TO-INT( )
STRING = string_ausdruck

Ergebnistyp

INTEGER

Eingabeparameter

STRING = string_ausdruck
Bezeichnet den maximal 4 Byte langen String, der konvertiert wird.

Ergebnis

Integer-Wert

0
Der String enthält den Leerstring.

Fehlermeldung

```
SDP0403   SPEZIFIZIERTE ZEICHENKETTE ZU LANG (MEHR ALS 4 BUCHSTABEN)
```

Beispiel

```
/DECLARE-VARIABLE A( TYPE= *INTEGER )
/A = X-LIT-TO-INT( X'F1F2F3F4' )
/SHOW-VARIABLE A
A = -235736076
/A = X-LIT-TO-INT( C'/ $*' )
/SHOW-VARIABLE A
A = 1631607644

/A = X-LIT-TO-INT( C'' )
/SHOW-VARIABLE A
A = 0
/A = X-LIT-TO-INT( X'' )
/SHOW-VARIABLE A
A = 0
/A = X-LIT-TO-INT( X'00' )
/SHOW-VARIABLE A
A = 0
```

15 SDF-P-Kommandos

Dieses Kapitel enthält - alphabetisch sortiert - die Beschreibungen der SDF-P-Kommandos. Zusätzlich sind die Kommandos CALL- und ENTER-PROCEDURE, die zum Grundausbau von BS2000/OSD gehören, auf dem Stand von BS2000/OSD-BC V7.0 enthalten.

Vor die Kommandobeschreibungen sind drei Abschnitte eingefügt:

– die Beschreibung der SDF-Syntax

– eine allgemeine Beschreibung der Kommando-Returncodes

– eine kurze Beschreibung der Privilegien

15.1 SDF-Syntaxdarstellung

Das folgende Beispiel zeigt die Syntaxdarstellung eines Kommandos in einem Handbuch. Das Kommandoformat besteht aus einem Feld mit dem Kommandonamen. Anschließend werden alle Operanden mit den zulässigen Operandenwerten aufgelistet. Struktureinleitende Operandenwerte und die von ihnen abhängigen Operanden werden zusätzlich aufgelistet.

HELP-SDF Kurzname: **HPSDF**

GUIDANCE**-MODE** = ***NO** / ***Y**ES
,**SDF-COM**MANDS = ***NO** / ***Y**ES
,**ABBR**EVIATION**-RULES** = ***NO** / ***Y**ES
,**GUID**ED**-DIA**LOG = ***Y**ES (...)
***Y**ES(...)
SCREEN-STEPS = ***NO** / ***Y**ES
,**SPEC**IAL**-FUNC**TIONS = ***NO** / ***Y**ES
,**FUNC**TION**-KEYS** = ***NO** / ***Y**ES
,**NEXT-FIELD** = ***NO** / ***Y**ES
,**UNGUID**ED**-DIA**LOG = ***Y**ES (...) / ***NO**
***Y**ES(...)
SPECIAL**-FUNC**TIONS = ***NO** / ***Y**ES
,**FUNC**TION**-KEYS** = ***NO** / ***Y**ES

Diese Syntaxbeschreibung basiert auf der SDF-Version 4.6A. Die Syntax der SDF-Kommando-/Anweisungssprache wird im Folgenden in drei Tabellen erklärt.

Zu Tabelle 1: Metasyntax

In den Kommando-/Anweisungsformaten werden bestimmte Zeichen und Darstellungsformen verwendet, deren Bedeutung in Tabelle 1 erläutert wird.

Zu Tabelle 2: Datentypen

Variable Operandenwerte werden in SDF durch Datentypen dargestellt. Jeder Datentyp repräsentiert einen bestimmten Wertevorrat. Die Anzahl der Datentypen ist beschränkt auf die in Tabelle 2 beschriebenen Datentypen.

Die Beschreibung der Datentypen gilt für alle Kommandos und Anweisungen. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle 2 erläutert.

Zu Tabelle 3: Zusätze zu Datentypen

Zusätze zu Datentypen kennzeichnen weitere Eingabevorschriften für Datentypen. Die Zusätze enthalten eine Längen- bzw. Intervallangabe, schränken den Wertevorrat ein (Zusatz beginnt mit *without*), erweitern ihn (Zusatz beginnt mit *with*) oder erklären eine bestimmte Angabe zur Pflichtangabe (Zusatz beginnt mit *mandatory*). Im Handbuch werden folgende Zusätze in gekürzter Form dargestellt:

cat-id	cat
completion	compl
correction-state	corr
generation	gen
lower-case	low
manual-release	man
odd-possible	odd
path-completion	path-compl
separators	sep
temporary-file	temp-file
underscore	under
user-id	user
version	vers
wildcard-constr	wild-constr
wildcards	wild

Für den Datentyp integer enthält Tabelle 3 außerdem kursiv gesetzte Einheiten, die nicht Bestandteil der Syntax sind. Sie dienen lediglich als Lesehilfe.
Für Sonderdatentypen, die durch die Implementierung geprüft werden, enthält Tabelle 3 kursiv gesetzte Zusätze (siehe Zusatz *special*), die nicht Bestandteil der Syntax sind.

Die Beschreibung der Zusätze zu den Datentypen gilt für alle Kommandos und Anweisungen. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle 3 erläutert.

Metasyntax

Kennzeichnung	Bedeutung	Beispiele
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Schlüsselwörter (Kommando-, Anweisungs-, Operandennamen, Schlüsselwortwerte) und konstante Operandenwerte. Schlüsselwortwerte beginnen mit *.	**HELP-SDF** **SCREEN-STEPS** = ***NO**
GROSSBUCHSTABEN in Halbfett	Großbuchstaben in Halbfett kennzeichnen garantierte bzw. vorgeschlagene Abkürzungen der Schlüsselwörter.	**GUID**ANCE**-MODE** = ***Y**ES
=	Das Gleichheitszeichen verbindet einen Operandennamen mit den dazugehörenden Operandenwerten.	**GUID**ANCE**-MODE** = ***NO**
< >	Spitze Klammern kennzeichnen Variablen, deren Wertevorrat durch Datentypen und ihre Zusätze beschrieben wird (siehe Tabellen 2 und 3).	**SYNTAX-F**ILE = <filename 1..54>
Unterstreichung	Der Unterstrich kennzeichnet den Default-Wert eines Operanden.	**GUID**ANCE**-MODE** = ***NO**
/	Der Schrägstrich trennt alternative Operandenwerte.	**NEXT-FIELD** = ***NO** / ***Y**ES
(...)	Runde Klammern kennzeichnen Operandenwerte, die eine Struktur einleiten.	,**UNGUID**ED**-DIA**LOG = ***Y**ES (...) / ***NO**
[ ]	Eckige Klammern kennzeichnen struktureinleitende Operandenwerte, deren Angabe optional ist. Die nachfolgende Struktur kann ohne den einleitenden Operandenwert angegeben werden.	**SELECT** = [***BY-ATTR**IBUTES](...)
Einrückung	Die Einrückung kennzeichnet die Abhängigkeit zu dem jeweils übergeordneten Operanden.	,**GUID**ED**-DIA**LOG = ***Y**ES (...) ***Y**ES(...) **SCREEN-STEPS** = ***NO** / ***Y**ES

Tabelle 1: Metasyntax (Teil 1 von 2)

Kennzeichnung	Bedeutung	Beispiele
|	Der Strich kennzeichnet zusammengehörende Operanden einer Struktur. Sein Verlauf zeigt Anfang und Ende einer Struktur an. Innerhalb einer Struktur können weitere Strukturen auftreten. Die Anzahl senkrechter Striche vor einem Operanden entspricht der Strukturtiefe.	**SUP**PORT = ***TAPE**(...) ***TAPE(...)** | **VOL**UME = ***ANY**(...) | ***ANY**(...) | | ...
,	Das Komma steht vor weiteren Operanden der gleichen Strukturstufe.	**GUID**ANCE**-MODE** = ***NO** / ***Y**ES ,**SDF-COM**MANDS = ***NO** / ***Y**ES
list-poss(n):	Aus den list-poss folgenden Operandenwerten kann eine Liste gebildet werden. Ist (n) angegeben, können maximal n Elemente in der Liste vorkommen. Enthält die Liste mehr als ein Element, muss sie in runde Klammern eingeschlossen werden.	list-poss: ***SAM** / ***ISAM** list-poss(40): <structured-name 1..30> list-poss(256): ***OMF** / ***SYSLST**(...) / <filename 1..54>
Kurzname:	Der darauf folgende Name ist ein garantierter Aliasname des Kommando- bzw. Anweisungsnamens.	**HELP-SDF** Kurzname: **HPSDF**

Tabelle 1: Metasyntax (Teil 2 von 2)

Datentypen

Datentyp	Zeichenvorrat	Besonderheiten
alphanum-name	A…Z 0…9 $, #, @	
cat-id	A…Z 0…9	maximal 4 Zeichen; darf nicht mit der Zeichenfolge PUB beginnen
command-rest	beliebig	
composed-name	A…Z 0…9 $, #, @ Bindestrich Punkt Katalogkennung	alphanumerische Zeichenfolge, die in mehrere durch Punkt oder Bindestrich getrennte Teilzeichenfolgen gegliedert sein kann. Ist auch die Angabe eines Dateinamens möglich, so kann die Zeichenfolge mit einer Katalogkennung im Format :cat: beginnen (siehe Datentyp filename).
c-string	EBCDIC-Zeichen	ist in Hochkommata einzuschließen; der Buchstabe C kann vorangestellt werden; Hochkommata innerhalb des c-string müssen verdoppelt werden
date	0…9 Strukturkennzeichen: Bindestrich	Eingabeformat: jjjj-mm-tt jjjj: Jahr; wahlweise 2- oder 4-stellig mm: Monat tt: Tag
device	A…Z 0…9 Bindestrich	Zeichenfolge, die maximal 8 Zeichen lang ist und einem im System verfügbaren Gerät entspricht. In der Dialogführung zeigt SDF die zulässigen Operandenwerte an. Hinweise zu möglichen Geräten sind der jeweiligen Operandenbeschreibung zu entnehmen.
fixed	+, - 0…9 Punkt	Eingabeformat: [zeichen][ziffern].[ziffern] [zeichen]: + oder - [ziffern]: 0...9 muss mindestens eine Ziffer, darf aber außer dem Vorzeichen maximal 10 Zeichen (0…9, Punkt) enthalten

Tabelle 2: Datentypen (Teil 1 von 6)

Datentyp	Zeichenvorrat	Besonderheiten
filename	A…Z 0…9 $, #, @ Bindestrich Punkt	Eingabeformat: [:cat:][$user.] { datei | datei(nr) | gruppe | gruppe { (*abs) | (+rel) | (-rel) } } :cat: wahlfreie Angabe der Katalogkennung; Zeichenvorrat auf A…Z und 0…9 eingeschränkt; max. 4 Zeichen; ist in Doppelpunkte einzuschließen; voreingestellt ist die Katalogkennung, die der Benutzerkennung laut Eintrag im Benutzerkatalog zugeordnet ist. $user. wahlfreie Angabe der Benutzerkennung; Zeichenvorrat ist A…Z, 0…9, $, #, @; max. 8 Zeichen; darf nicht mit einer Ziffer beginnen; $ und Punkt müssen angegeben werden; voreingestellt ist die eigene Benutzerkennung. $. (Sonderfall) System-Standardkennung datei Datei- oder Jobvariablenname; kann durch Punkt in mehrere Teilnamen gegliedert sein: $name_1$[.$name_2$[...]] $name_i$ enthält keinen Punkt und darf nicht mit Bindestrich beginnen oder enden; datei ist max. 41 Zeichen lang, darf nicht mit $ beginnen und muss mindestens ein Zeichen aus A…Z enthalten.

Tabelle 2: Datentypen (Teil 2 von 6)

Datentyp	Zeichenvorrat	Besonderheiten
filename (Forts.)		#datei (Sonderfall) @datei (Sonderfall) \# oder @ als erstes Zeichen kennzeichnet je nach Systemparameter temporäre Dateien und Jobvariablen. datei(nr) Banddateiname nr: Versionsnummer; Zeichenvorrat ist A…Z, 0…9, $, #, @. Klammern müssen angegeben werden. gruppe Name einer Dateigenerationsgruppe (Zeichenvorrat siehe unter „datei“) gruppe{(*abs) (+rel) (-rel)} (*abs) absolute Generationsnummer (1..9999); * und Klammern müssen angegeben werden. (+rel) (-rel) relative Generationsnummer (0..99); Vorzeichen und Klammern müssen angegeben werden.
integer	0…9, +, -	+ bzw. - kann nur erstes Zeichen (Vorzeichen) sein.
name	A…Z 0…9 $, #, @	darf nicht mit einer Ziffer beginnen.

Tabelle 2: Datentypen (Teil 3 von 6)

Datentyp	Zeichenvorrat	Besonderheiten
partial-filename	A…Z 0…9 $, #, @ Bindestrich Punkt	Eingabeformat: [:cat:][$user.][teilname.] :cat: siehe filename $user. siehe filename teilname wahlfreie Angabe des gemeinsamen ersten Namensteils von Dateien und Dateigenerationsgruppen in der Form: $name_1$.[$name_2$.[...]] $name_i$ siehe filename. Das letzte Zeichen von teilname muss ein Punkt sein. Es muss mindestens einer der Teile :cat:, $user. oder teilname angegeben werden.
posix-filename	A...Z 0...9 Sonderzeichen	Zeichenfolge, die maximal 255 Zeichen lang ist. Besteht entweder aus einem oder zwei Punkten, oder aus alphanumerischen Zeichen und Sonderzeichen; Sonderzeichen sind mit dem Zeichen \ zu entwerten. Nicht erlaubt ist das Zeichen /. Muss in Hochkommata eingeschlossen werden, wenn alternative Datentypen zulässig sind, Separatoren verwendet werden oder das erste Zeichen ?, ! bzw. ^ ist. Zwischen Groß- und Kleinschreibung wird unterschieden.
posix-pathname	A...Z 0...9 Sonderzeichen Strukturkennzeichen: Schrägstrich	Eingabeformat: [/]$part_1$[/.../$part_n$] wobei $part_i$ ein posix-filename ist; maximal 1023 Zeichen; muss in Hochkommata eingeschlossen werden, wenn alternative Datentypen zulässig sind, Separatoren verwendet werden oder das erste Zeichen ?, ! bzw. ^ ist.

Tabelle 2: Datentypen (Teil 4 von 6)

Datentyp	Zeichenvorrat	Besonderheiten
product-version	A…Z 0…9 Punkt Hochkomma	Eingabeformat: [[C']][V][m]m.naso['] Korrekturstand Freigabestand wobei m, n, s und o jeweils eine Ziffer und a ein Buchstabe ist. Ob Freigabe- und/oder Korrekturstand angegeben werden dürfen oder ob sie angegeben werden müssen, bestimmen Zusätze zu dem Datentyp (siehe Tabelle 3, Zusätze without-corr, without-man, mandatory-man und mandatory-corr). product-version kann in Hochkommata eingeschlossen werden, wobei der Buchstabe C vorangestellt werden kann. Die Versionsangabe kann mit dem Buchstaben V beginnen.
structured-name	A…Z 0…9 $, #, @ Bindestrich	alphanumerische Zeichenfolge, die in mehrere durch Bindestrich getrennte Teilzeichenfolgen gegliedert sein kann; erstes Zeichen: A…Z oder $, #, @
text	beliebig	Das Eingabeformat ist den jeweiligen Operandenbeschreibungen zu entnehmen.
time	0…9 Strukturkennzeichen: Doppelpunkt	Angabe einer Tageszeit Eingabeformat: { hh:mm:ss / hh:mm / hh } hh: Stunden mm: Minuten ss: Sekunden führende Nullen können weggelassen werden
vsn	a) A…Z 0…9 b) A…Z 0…9 $, #, @	a) Eingabeformat: pvsid.folgenummer max. 6 Zeichen; pvsid: 2-4 Zeichen; Eingabe von PUB nicht erlaubt folgenummer: 1-3 Zeichen b) max. 6 Zeichen; PUB darf vorangestellt werden, dann dürfen jedoch nicht $, #, @ folgen.

Tabelle 2: Datentypen (Teil 5 von 6)

Datentyp	Zeichenvorrat	Besonderheiten
x-string	Sedezimal: 00…FF	ist in Hochkommata einzuschließen; der Buchstabe X muss vorangestellt werden; die Anzahl der Zeichen darf ungerade sein.
x-text	Sedezimal: 00…FF	ist nicht in Hochkommata einzuschließen; der Buchstabe X darf nicht vorangestellt werden; die Anzahl der Zeichen darf ungerade sein.

Tabelle 2: Datentypen (Teil 6 von 6)

Zusätze zu Datentypen

Zusatz	Bedeutung
x..y *unit*	beim Datentyp integer: Intervallangabe x Mindestwert, der für integer erlaubt ist. x ist eine ganze Zahl, die mit einem Vorzeichen versehen werden darf. y Maximalwert, der für integer erlaubt ist. y ist eine ganze Zahl, die mit einem Vorzeichen versehen werden darf. *unit* Dimension. Folgende Angaben werden verwendet: *days* *byte* *hours* *2Kbyte* *minutes* *4Kbyte* *seconds* *Mbyte* *milliseconds*
x..y *special*	bei den übrigen Datentypen: Längenangabe Bei den Datentypen catid, date, device, product-version, time und vsn wird die Längenangabe nicht angezeigt. x Mindestlänge für den Operandenwert; x ist eine ganze Zahl. y Maximallänge für den Operandenwert; y ist eine ganze Zahl. x=y Der Operandenwert muss genau die Länge x haben. *special* Zusatzangabe zur Beschreibung eines Sonderdatentyps, der durch die Implementierung geprüft wird. Vor *special* können weitere Zusätze stehen. Folgende Angaben werden verwendet: *arithm-expr* arithmetischer Ausdruck (SDF-P) *bool-expr* logischer Ausdruck (SDF-P) *string-expr* String-Ausdruck (SDF-P) *expr* beliebiger Ausdruck (SDF-P) *cond-expr* bedingter Ausdruck (JV) *symbol* CSECT- oder Entry-Name (BLS)
with	Erweitert die Angabemöglichkeiten für einen Datentyp.
-compl	Bei Angaben zu dem Datentyp date ergänzt SDF zweistellige Jahresangaben der Form jj-mm-tt zu: 20jj-mm-tt falls jj < 60 19jj-mm-tt falls jj ≥ 60
-low	Groß- und Kleinschreibung wird unterschieden.
-path-compl	Bei Angaben zu dem Datentyp filename ergänzt SDF die Katalog- und/oder die Benutzerkennung, falls diese nicht angegeben werden.
-under	Erlaubt Unterstriche '_' bei den Datentypen name und composed-name.

Tabelle 3: Zusätze zu Datentypen (Teil 1 von 7)

<table>
<tr><th>Zusatz</th><th>Bedeutung</th></tr>
<tr><td>with (Forts.)

-wild(n)</td><td>Teile eines Namens dürfen durch die folgenden Platzhalter ersetzt werden. n bezeichnet die maximale Eingabelänge bei Verwendung von Platzhaltern. Mit Einführung der Datentypen posix-filename und posix-pathname akzeptiert SDF neben den bisher im BS2000 üblichen Platzhaltern auch Platzhalter aus der UNIX-Welt (nachfolgend POSIX-Platzhalter genannt). Da derzeit nicht alle Kommandos POSIX-Platzhalter unterstützen, kann ihre Verwendung bei Datentypen ungleich posix-filename und posix-pathname zu Semantikfehlern führen.
Innerhalb einer Musterzeichenfolge sollten entweder nur BS2000- oder nur POSIX-Platzhalter verwendet werden. Bei den Datentypen posix-filename und posix-pathname sind nur POSIX-Platzhalter erlaubt. Ist eine Musterzeichenfolge mehrdeutig auf einen String abbildbar, gilt der erste Treffer.
<table>
<tr><th>BS2000-Platzhalter</th><th>Bedeutung</th></tr>
<tr><td>*</td><td>Ersetzt eine beliebige, auch leere Zeichenfolge. Ein * an erster Stelle muss verdoppelt werden, sofern dem * weitere Zeichen folgen und die eingegebene Zeichenfolge nicht mindestens einen weiteren Platzhalter enthält.</td></tr>
<tr><td>Punkt am Ende</td><td>Teilqualifizierte Angabe eines Namens.
Entspricht implizit der Zeichenfolge „./*“, d.h. nach dem Punkt folgt mindestens ein beliebiges Zeichen.</td></tr>
<tr><td>/</td><td>Ersetzt genau ein beliebiges Zeichen.</td></tr>
<tr><td><s_x:s_y></td><td>Ersetzt eine Zeichenfolge, für die gilt:
– sie ist mindestens so lang wie die kürzeste Zeichenfolge (s_x oder s_y)
– sie ist höchstens so lang wie die längste Zeichenfolge (s_x oder s_y)
– sie liegt in der alphabetischen Sortierung zwischen s_x und s_y; Zahlen werden hinter Buchstaben sortiert (A…Z, 0…9)
– s_x darf auch die leere Zeichenfolge sein, die in der alphabetischen Sortierung an erster Stelle steht
– s_y darf auch die leere Zeichenfolge sein, die an dieser Stelle für die Zeichenfolge mit der höchst möglichen Codierung steht (enthält nur die Zeichen X’FF’)</td></tr>
</table>
</td></tr>
</table>

Tabelle 3: Zusätze zu Datentypen (Teil 2 von 7)

Zusatz	Bedeutung	
with-wild(n) (Forts.)	<s_1,…>	Ersetzt alle Zeichenfolgen, auf die eine der mit s angegebenen Zeichenkombinationen zutrifft. s kann auch die leere Zeichenfolge sein. Jede Zeichenfolge s kann auch eine Bereichsangabe „s_x:s_y“ sein (siehe Seite 557).
	-s	Ersetzt alle Zeichenfolgen, die der angegebenen Zeichenfolge s nicht entsprechen. Das Minuszeichen darf nur am Beginn der Zeichenfolge stehen. Innerhalb der Datentypen filename bzw. partial-filename kann die negierte Zeichenfolge -s genau einmal verwendet werden, d.h., -s kann einen der drei Namensteile cat, user oder datei ersetzen.
	Platzhalter sind in Generations- und Versionsangaben von Dateinamen nicht erlaubt. In Benutzerkennungen ist die Angabe von Platzhaltern der Systemverwaltung vorbehalten. Platzhalter können nicht die Begrenzer der Namensteile cat (Doppelpunkte) und user ($ und Punkt) ersetzen.	
	POSIX-Platzhalter	Bedeutung
	*	Ersetzt eine beliebige, auch leere Zeichenfolge. Ein * an erster Stelle muss verdoppelt werden, sofern dem * weitere Zeichen folgen und die eingegebene Zeichenfolge nicht mindestens einen weiteren Platzhalter enthält.
	?	Ersetzt genau ein beliebiges Zeichen. Ist als erstes Zeichen außerhalb von Hochkommata nicht zulässig.
	[c_x-c_y]	Ersetzt genau ein Zeichen aus dem Bereich c_x und c_y einschließlich der Bereichsgrenzen. c_x und c_y müssen einfache Zeichen sein.
	[s]	Ersetzt genau ein Zeichen aus der Zeichenfolge s. Die Ausdrücke [c_x-c_y] und [s] können kombiniert werden zu [s_1c_x-c_ys_2]
	[!c_x-c_y]	Ersetzt genau ein Zeichen, das nicht in dem Bereich c_x und c_y einschließlich der Bereichsgrenzen enthalten ist. c_x und c_y müssen einfache Zeichen sein. Die Ausdrücke [!c_x-c_y] und [!s] können kombiniert werden zu [!s_1c_x-c_ys_2]
	[!s]	Ersetzt genau ein Zeichen, das nicht in der Zeichenfolge s enthalten ist. Die Ausdrücke [!s] und [!c_x-c_y] können kombiniert werden zu [!s_1c_x-c_ys_2]

Tabelle 3: Zusätze zu Datentypen (Teil 3 von 7)

<table>
<tr><th>Zusatz</th><th>Bedeutung</th></tr>
<tr><td>with (Forts.)

-wild-constr(n)</td><td>Angabe einer Konstruktionszeichenfolge, die angibt, wie aus einer zuvor angegebenen Auswahlzeichenfolge mit Musterzeichen (siehe with-wild) neue Namen zu bilden sind. n bezeichnet die maximale Eingabelänge bei Verwendung von Platzhaltern.
Die Konstruktionszeichenfolge kann aus konstanten Zeichenfolgen und Musterzeichen bestehen. Ein Musterzeichen wird durch diejenige Zeichenfolge ersetzt, die durch das entsprechende Musterzeichen in der Auswahlzeichenfolge ausgewählt wird.
Folgende Platzhalter können zur Konstruktionsangabe verwendet werden:
<table>
<tr><th>Platzhalter</th><th>Bedeutung</th></tr>
<tr><td>*</td><td>Entspricht der Zeichenfolge, die durch den Platzhalter * in der Auswahlzeichenfolge ausgewählt wird.</td></tr>
<tr><td>Punkt am Ende</td><td>Entspricht der teilqualifizierten Angabe eines Namens in der Auswahlzeichenfolge.
Entspricht der Zeichenfolge, die durch den Punkt am Ende der Auswahlzeichenfolge ausgewählt wird.</td></tr>
<tr><td>/ oder ?</td><td>Entspricht dem Zeichen, das durch den Platzhalter / oder ? in der Auswahlzeichenfolge ausgewählt wird.</td></tr>
<tr><td><n></td><td>Entspricht der Zeichenfolge, die durch den n-ten Platzhalter in der Auswahlzeichenfolge ausgewählt wird; n = <integer></td></tr>
</table>
Zuordnung der Platzhalter zu entsprechenden Platzhaltern in der Auswahlzeichenfolge:
In der Auswahlzeichenfolge werden alle Platzhalter von links nach rechts aufsteigend nummeriert (globaler Index).
Gleiche Platzhalter in der Auswahlzeichenfolge werden zusätzlich von links nach rechts aufsteigend nummeriert (platzhalter-spezifischer Index).
In der Konstruktionsangabe können Platzhalter auf zwei, sich gegenseitig ausschließende Arten angegeben werden:
1. Platzhalter werden über den globalen Index angegeben: <n>
2. Angabe desselben Platzhalters, wobei die Ersetzung gemäß dem platzhalter-spezifischen Index entsprechend erfolgt:
z.B. der zweite „/“ entspricht der Zeichenfolge, die durch den zweiten „/ “ in der Auswahlzeichenfolge ausgewählt wird.</td></tr>
</table>

Tabelle 3: Zusätze zu Datentypen (Teil 4 von 7)

Zusatz	Bedeutung
with-wild-constr(n) (Forts.)	Bei Konstruktionsangaben sind folgende Regeln zu beachten: – Die Konstruktionsangabe kann nur Platzhalter der Auswahlzeichenfolge enthalten. – Soll die Zeichenkette, die der Platzhalter <...> bzw. [...] auswählt, in der Konstruktionsangabe verwendet werden, muss die Index-Schreibweise gewählt werden. – Die Index-Schreibweise muss gewählt werden, wenn die Zeichenkette, die einen Platzhalter der Auswahlzeichenfolge bezeichnet, in der Konstruktionsangabe mehrfach verwendet werden soll: Bei der Auswahlangabe „A/“ muss z.B. statt „A//“ die Konstruktionszeichenfolge „A<n><n>“ angegeben werden. – Der Platzhalter * kann auch die leere Zeichenkette sein. Insbesondere ist zu beachten, dass bei mehreren Sternen in Folge (auch mit weiteren Platzhaltern) nur der letzte Stern eine nicht leere Zeichenfolge sein kann: z.B. bei „****“ oder „*//*“. – Aus der Konstruktionsangabe sollten gültige Namen entstehen. Darauf ist sowohl bei der Auswahlangabe als auch bei der Konstruktionsangabe zu achten. – Abhängig von der Konstruktionsangabe können aus unterschiedlichen Namen, die in der Auswahlangabe ausgewählt werden, identische Namen gebildet werden: z.B. „A/*“ wählt die Namen „A1“ und „A2“ aus; die Konstruktionsangabe „B*“ erzeugt für beide Namen denselben neuen Namen „B“. Um dies zu vermeiden, sollten in der Konstruktionsangabe alle Platzhalter der Auswahlangabe mindestens einmal verwendet werden. – Wird die Konstruktionsangabe mit einem Punkt abgeschlossen, so muss auch die Auswahlzeichenfolge mit einem Punkt enden. Die Zeichenfolge, die durch den Punkt am Ende der Auswahlzeichenfolge ausgewählt wird, kann in der Konstruktionsangabe nicht über den globalen Index angegeben werden.

Tabelle 3: Zusätze zu Datentypen (Teil 5 von 7)

<table>
<tr><th>Zusatz</th><th>Bedeutung</th></tr>
<tr><td>with-wild-constr(n) (Forts.)</td><td>Beispiele:
<table>
<tr><th>Auswahlmuster</th><th>Auswahl</th><th>Konstruktionsmuster</th><th>neuer Name</th></tr>
<tr><td>A//*</td><td>AB1
AB2
A.B.C</td><td>D<3><2></td><td>D1
D2
D.CB</td></tr>
<tr><td>C.<A:C>/<D,F></td><td>C.AAD
C.ABD
C.BAF
C.BBF</td><td>G.<1>.<3>.XY<2></td><td>G.A.D.XYA
G.A.D.XYB
G.B.F.XYA
G.B.F.XYB</td></tr>
<tr><td>C.<A:C>/<D,F></td><td>C.AAD
C.ABD
C.BAF
C.BBF</td><td>G.<1>.<2>.XY<2></td><td>G.A.A.XYA
G.A.B.XYB
G.B.A.XYA
G.B.B.XYB</td></tr>
<tr><td>A//B</td><td>ACDB
ACEB
AC.B
A.CB</td><td>G/XY/</td><td>GCXYD
GCXYE
GCXY. 1)
G.XYC</td></tr>
</table>
1) Punkt am Ende des Namens kann Namenskonvention widersprechen (z.B bei vollqualifizierten Dateinamen)</td></tr>
<tr><td>without</td><td>Schränkt die Angabemöglichkeiten für einen Datentyp ein.</td></tr>
<tr><td>-cat</td><td>Die Angabe einer Katalogkennung ist nicht erlaubt.</td></tr>
<tr><td>-corr</td><td>Eingabeformat: [[C]’][V][m]m.na[’]
Angaben zum Datentyp product-version dürfen den Korrekturstand nicht enthalten.</td></tr>
<tr><td>-gen</td><td>Die Angabe einer Dateigeneration oder Dateigenerationsgruppe ist nicht erlaubt.</td></tr>
<tr><td>-man</td><td>Eingabeformat: [[C]’][V][m]m.n[’]
Angaben zum Datentyp product-version dürfen weder Freigabe- noch Korrekturstand enthalten.</td></tr>
<tr><td>-odd</td><td>Der Datentyp x-text erlaubt nur eine gerade Anzahl von Zeichen.</td></tr>
<tr><td>-sep</td><td>Beim Datentyp text ist die Angabe der folgenden Trennzeichen nicht erlaubt: ; = ( ) < > ␣ (also Strichpunkt, Gleichheitszeichen, runde Klammer auf und zu, Größerzeichen, Kleinerzeichen und Leerzeichen)</td></tr>
<tr><td>-temp-file</td><td>Die Angabe einer temporären Datei ist nicht erlaubt (siehe #datei bzw. @datei bei filename).</td></tr>
</table>

Tabelle 3: Zusätze zu Datentypen (Teil 6 von 7)

Zusatz	Bedeutung
without (Forts.)	
-user	Die Angabe einer Benutzerkennung ist nicht erlaubt.
-vers	Die Angabe der Version (siehe „datei(nr)“) ist bei Banddateien nicht erlaubt.
-wild	Die Datentypen posix-filename bzw. posix-pathname dürfen keine Musterzeichen enthalten.
mandatory	Bestimmte Angaben sind für einen Datentyp zwingend erforderlich.
-corr	Eingabeformat: [[C]’][V][m]m.naso[’] Angaben zum Datentyp product-version müssen den Korrekturstand (und damit auch den Freigabestand) enthalten.
-man	Eingabeformat: [[C]’][V][m]m.na[so][’] Angaben zum Datentyp product-version müssen den Freigabestand enthalten. Die Angabe des Korrekturstands ist optional möglich, wenn dies nicht durch den Zusatz without-corr untersagt wird.
-quotes	Angaben zu den Datentypen posix-filename bzw. posix-pathname müssen in Hochkommata eingeschlossen werden.

Tabelle 3: Zusätze zu Datentypen (Teil 7 von 7)

15.2 Kommando-Returncode

Alle SDF-P-Kommandos liefern Returncodes, die den Benutzer über die Ausführung des Kommandos informieren. Dieser Kommando-Returncode ist vergleichbar mit dem Returncode auf Programmebene. Der Kommando-Returncode ermöglicht es dem Benutzer, auf Fehlersituationen zu reagieren.

Der Kommando-Returncode besteht aus drei Teilen:

- dem Subcode1, der die aufgetretene Fehlersituation in eine Fehlerklasse einordnet, aus der abgeleitet werden kann, wie schwer wiegend ein Fehler ist. Der Wert von Subcode1 wird *dezimal* ausgegeben. Folgende fünf Fehlerklassen sind im BS2000 definiert:
  - Klasse A: kein Fehler
  - Der Wert ist Null. Es kann normal weitergearbeitet werden.
  - Klasse B: Syntaxfehler
  - Der Wert ist eine Zahl zwischen 1 und 31. Das Kommando wurde syntaktisch falsch eingegeben. Eine Wiederholung der Eingabe ist nur nach Korrektur des Syntaxfehlers sinnvoll.
  - Klasse C: interner Fehler (Systemfehler)
  - Der Wert ist 32. Eine Wiederholung der Eingabe ist nur sinnvoll, wenn der interne Fehler behoben wurde.
  - Klasse D: Fehler, die keiner anderen Fehlerklasse zuzuordnen sind
  - Der Wert ist eine Zahl zwischen 64 und 127. Zur Bestimmung der weiteren Vorgehensweise sollte der Maincode ausgewertet werden.
  - Klasse E: Kommando kann vorübergehend nicht ausgeführt werden
  - Der Wert ist eine Zahl zwischen 128 und 130. Die Eingabe kann unverändert wiederholt werden. Das Kommando kann nach einer Wartezeit wieder ausgeführt werden. Die Dauer der Wartezeit wird eingeteilt in kurzfristig, langfristig und unbefristet.
  - Kurzfristig entspricht dem Wert 128 und bedeutet, dass ein Warten im Dialog als sinnvoll angesehen wird.
  - Langfristig entspricht dem Wert 129 und bedeutet, dass ein Warten im Stapelbetrieb als sinnvoll angesehen wird.
  - Unbefristet entspricht dem Wert 130 und bedeutet, dass es unsicher ist, ob der Fehler überhaupt behoben wird.
- dem Subcode2, der Zusatzinformationen zur Fehlerklasse enthalten kann.
- dem Maincode, der einem Meldungsschlüssel entspricht und differenzierte Fehlerinformationen liefert. Mit diesem Meldungsschlüssel kann über die vordefinierte Funktion MSG( ) oder mit dem Kommando HELP-MSG-INFORMATION die entsprechende Fehlermeldung ausgegeben werden (siehe Handbuch „Kommandos, Bd. 1-5“ [3]).

Der Kommando-Returncode kann mit den vordefinierten Funktionen SUBCODE1( ), SUBCODE2( ) und MAINCODE( ) abgefragt werden.

Es gibt für jedes Kommando eigene Returncodes. Auch gibt es neben den eigenen Returncodes für bestimmte Kommandotypen noch übergreifende Returncodes, die hier anschließend aufgelistet werden.

Hinweise

– Die Ausführung eines Kommandos wird gewöhnlich beendet, wenn ein Fehler auftritt. Wenn mehr als ein Fehler auftreten sollte, kann nicht immer festgestellt werden, dass der gemeldete Fehler auch der zuerst aufgetretene Fehler ist, weil die Reihenfolge der überprüften Operanden nicht garantiert ist.

– Die Returncodes und Meldungen sind nur für S-Prozeduren garantiert, nicht aber für Nicht-S-Prozeduren oder Stapelaufträge.

– Die folgenden Kommandos werden teilweise oder sogar ganz während der Voranalyse einer S-Prozedur überprüft. Jeder während dieser Voranalyse auftretende Fehler ist ein Fehler im Kommando während der Prozedurvorbereitung, und das Kommando wird nicht ausgeführt. Darum sind die Fehlermeldungen, die tatsächlich während der Ausführung des Kommandos auftauchen, sehr begrenzt. Wenn eines dieser Kommandos durch Ausdrucksersetzung erzeugt wird, tritt ein Kontextfehler auf. Die Kommandos sind:

BEGIN-BLOCK	EXIT-BLOCK
BEGIN-PARAMETER-DECLARATION	FOR
CYCLE	GOTO
DECLARE-PARAMETER	IF
ELSE	IF-BLOCK-ERROR
ELSE-IF	IF-CMD-ERROR
END-BLOCK	INCLUDE-BLOCK
END-FOR	REPEAT
END-IF	SET-PROCEDURE-OPTIONS
END-PARAMETER-DECLARATION	UNTIL
END-WHILE	WHILE

– Wenn ein Fehler während der Ausführung eines Kommandos auftritt, das einen Block einleitet, dann kann dieser Fehler nur in einem IF-BLOCK-ERROR-Block bearbeitet werden; und zwar nach dem Ende des Blocks, der durch das fehlerhafte Kommando gestartet wurde.

– Die Voreinstellung ist CMD0001 für den Maincode und jeweils 0 für den Subcode1 und Subcode2

Die allgemeinen Returncodes (d. h. Returncodes, die bei jedem Kommando auftreten können) sind:

(SC2)	SC1	Maincode	Bedeutung[1)]
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

[1)]Enthält die Tabelle auch garantierte Meldungen, wird die Bedeutungsspalte mit „/ garantierte Meldungen“ ergänzt.

Bei allen Kommandos, Anweisungen und Datensätzen in denen eine Ausdrucksersetzung stattfindet, können die folgenden Returncodes auftreten, wenn bei der Ausdrucksersetzung Fehler passieren:

(SC2)	SC1	Maincode	Bedeutung
	1	SDP0140	Syntaxfehler während der Ersetzung
	64	SDP0141	Semantikfehler während der Ersetzung

Bei allen Datenzeilen in falschem Kontext kann folgender Returncode auftreten:

(SC2)	SC1	Maincode	Bedeutung
	64	SDP0091	Semantikfehler

Bei allen Anweisungen in falschem Kontext kann folgender Returncode auftreten:

(SC2)	SC1	Maincode	Bedeutung
	64	SDP0091	Semantikfehler

15.3 Privilegien

Mit wenigen Ausnahmen können alle Kommandos von Benutzern mit den folgenden Privilegien aufgerufen werden:

STD-PROCESSING
OPERATING
HARDWARE-MAINTENANCE
SECURITY-ADMINISTRATION
SAT-FILE-MANAGEMENT
SAT-FILE-EVALUATION

Ausnahmen

- Kommando ENTER-PROCEDURE
  erforderliche Privilegien: STD-PROCESSING oder HARDWARE-MAINTENANCE
- Benutzer mit den Privilegien SECURITY-ADMINISTRATION, SAT-FILE-MANAGEMENT oder SAT-FILE-EVALUATION können die folgenden Kommandos nur in Prozeduren benutzen:

BEGIN-BLOCK	FOR
BEGIN-PARAMETER-DECLARATION	GOTO
CYCLE	IF
DECLARE-PARAMETER	IF-BLOCK-ERROR
ELSE	IF-CMD-ERROR
ELSE-IF	INCLUDE-BLOCK
END-BLOCK	INCLUDE-PROCEDURE
END-FOR	REPEAT
END-IF	SET-PROCEDURE-OPTIONS
END-PARAMETER-DECLARATION	TRACE-PROCEDURE
END-WHILE	UNTIL
EXECUTE-CMD	WHILE
EXIT-BLOCK	

15.4 Kommandos

ASSIGN-STREAM
S-Variablenstrom zuweisen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando ASSIGN-STREAM weist einen S-Variablenstrom für strukturierte Ausgaben einem (Ausgabe-)Server zu, der die weitere Verarbeitung des Variablenstroms steuert.

(Zum Variablenstrom siehe Abschnitt „S-Variablenströme zuweisen“ auf Seite 196.)

Format

ASSIGN-STREAM

STREAM-NAME = **SYSVAR** / **SYSMSG** / **SYSINF** / <structured-name 1..20>

,**TO** = ***STD** / <structured-name 1..20> / ***DUMMY** / ***SAME-AS-CALLING-PROC** / ***VAR**IABLE(...) / ***SERVER**(...)

***VAR**IABLE(...)

VARIABLE**-NAME** = ***NONE** / <composed-name 1..255>(...)

<composed-name 1..255>(...)

WRITE**-MODE** = ***EXTEND** / ***PREFIX**

,**RET**URN**-VAR**IABLE**-NAME** = ***NONE** / <composed-name 1..255>(...)

<composed-name 1..255>(...)

WRITE**-MODE** = ***EXTEND** / ***PREFIX**

,**CONTR**OL**-VAR-NAME** = ***NONE** / <composed-name 1..255>(...)

<composed-name 1..255>(...)

WRITE**-MODE** = ***EXTEND** / ***PREFIX**

,**RET-CONTROL-VAR-NAME** = ***NONE** / <composed-name 1..255>(...)

<composed-name 1..255>(...)

WRITE**-MODE** = ***EXTEND** / ***PREFIX**

***SERVER**(...)

SERVER-NAME = <structured-name 1..30>

,**SERVER-INFO**RMATION = ***NONE** / <c-string 1..1800>

Operandenbeschreibung

STREAM-NAME = <structured-name 1..20> / SYSVAR / SYSMSG / SYSINF
Name des S-Variablenstroms. Die konstanten Werte SYSINF, SYSMSG und SYSVAR sind reservierte Wörter. Sie dürfen nicht abgekürzt werden.

SYSINF: überträgt strukturierte Ausgaben von Kommandos und Programmen
SYSMSG: überträgt strukturierte Ausgaben garantierter Meldungen
SYSVAR: überträgt strukturierte Ausgaben von Kommandos, Programmen und garantierten Meldungen. Die verschiedenen Informationen können aber getrennt weiterverarbeitet werden

TO =
Gibt den Server an, der mit dem S-Variablenstrom verknüpft wird.

TO = *STD
Standardzuweisung.
Die folgende Tabelle gibt darüber Auskunft, welche Werte der Default-Wert von TO intern bei den verschiedensten Kombinationen mit dem Operanden STREAM-NAME annimmt.

STREAM-NAME=	TO=*STD	Enthaltene Informationen
SYSINF	SYSVAR	Strukt. Ausgaben von Kommandos und Programmen
SYSMSG	SYSVAR	Strukt. garantierte Meldungen
SYSVAR	*DUMMY	Strukt. Ausgaben von Kommandos, Programmen und garantierten Meldungen
<structured-name 1..20>	*DUMMY	Benutzer-Variablenstrom

TO = <structured-name 1..20>
Name des Benutzer-Servers.
Schleifen in verketteten Zuweisungen von S-Variablenströmen werden zurückgewiesen; z.B.

```
ASSIGN-STREAM S3,*DUMMY
ASSIGN-STREAM S2,S3
ASSIGN-STREAM S3,S2   →   SDP0511
```

TO = *DUMMY
Keine Zuweisung.
Übertragene Variablen werden entfernt. Der Client wird durch eine Warnung davon informiert.

TO = *SAME-AS-CALLING-PROC
Weist den Server der aufrufenden Prozedur zu.
Gibt es keine Zuweisung in der aufrufenden Prozedur, wird die Zuweisung abgewiesen und der S-Variablenstrom bleibt unverändert.

TO = *VARIABLE(...)
Der Server ist SDF-P.
Die übertragenen Variablen werden in die angegebene S-Variable geschrieben bzw. die Return-Informationen aus den angegebenen S-Variablen gelesen.

VARIABLE-NAME =
Gibt die S-Variable an, in die die übertragene S-Variable geschrieben wird (siehe dazu auch die Beschreibung des Kommandos TRANSMIT-BY-STREAM, Seite 796).

VARIABLE-NAME = *NONE
Die übertragene Output-Variable wird ignoriert.

VARIABLE-NAME = <composed-name 1..255>(...)
Name der S-Variablen, in die die übertragene S-Variable geschrieben wird.
Die angegebene S-Variable muss eine Liste von Strukturen sein.

WRITE-MODE =
Gibt an, wie die zugewiesene Input-Liste verarbeitet wird.

WRITE-MODE = *EXTEND
Die übertragenen Variablen werden als letztes Element der zugewiesenen S-Variable angehängt. Variablen von unterschiedlichen Übertragungen können akkumuliert werden.

WRITE-MODE = *PREFIX
Die übertragenen Variablen werden als erstes Element der zugewiesenen S-Variable angehängt. Variablen von unterschiedlichen Übertragungen können akkumuliert werden.

RETURN-VARIABLE-NAME =
Gibt die lokale S-Variable an, deren Inhalt in die entfernte Return-Variable übertragen wird (siehe dazu auch Kommando TRANSMIT-BY-STREAM, Seite 796).

Hinweis
Wenn bei RETURN-VARIABLE-NAME dieselbe Variable wie bei VARIABLE-NAME mit demselben WRITE-MODE angegeben wird, wird dies bei einer Übertragung nicht berichtigt. In diesem Fall wird die Return-Variable mit den Daten der Variablen überschrieben.

RETURN-VARIABLE-NAME = *NONE
Keine Übertragung. Die lokale und die entfernte Return-Variable werden nicht verändert.

RETURN-VARIABLE-NAME = <composed-name 1..255>(...)
Name der S-Variablen, die von der entfernten Return-Variablen gelesen wird.
Die angegebene S-Variable muss eine Liste von Strukturen sein.

WRITE-MODE =
Gibt an, wie die Return-Variable bzw. die Liste von Strukturen verarbeitet wird.

WRITE-MODE = *EXTEND
Das letzte Element der angegebenen Liste wird entfernt.
Durch die nächste Übertragung wird das dann folgende letzte Element entfernt.
Wenn die Liste leer ist, wird sie wie bei RETURN-VARIABLE-NAME = *NONE verarbeitet.

WRITE-MODE = *PREFIX
Das erste Element der angegebenen Liste wird entfernt.
Durch die nächste Übertragung wird das dann folgende erste Element entfernt.
Wenn die Liste leer ist, wird sie wie bei RETURN-VARIABLE-NAME = *NONE verarbeitet.

CONTROL-VAR-NAME =
Gibt die S-Variable an, in die die übertragene Kontroll-Variable geschrieben wird (siehe dazu auch Kommando TRANSMIT-BY-STREAM, Seite 796).

CONTROL-VAR-NAME = *NONE
Die übertragene Kontroll-Variable wird ignoriert.

CONTROL-VAR-NAME = <composed-name 1..255>(...)
Name der S-Variable, in die die übertragene Kontroll-Variable geschrieben wird.
Die angegebene S-Variable muss eine Liste von Strukturen sein.

WRITE-MODE =
Gibt an, wie die Kontroll-Variable bzw. die Liste von Strukturen verarbeitet wird.

WRITE-MODE = *EXTEND
Die übertragenen Kontroll-Variablen werden als letztes Element der zugewiesenen S-Variable angehängt. Kontroll-Variablen von unterschiedlichen Übertragungen können akkumuliert werden.

WRITE-MODE = *PREFIX
Die übertragenen Kontroll-Variablen werden als erstes Element der zugewiesenen S-Variable angehängt. Kontroll-Variablen von unterschiedlichen Übertragungen können akkumuliert werden.

RET-CONTROL-VAR-NAME =
Gibt die S-Variable an, von der die übertragene Return-Kontroll-Variable gelesen wird (siehe dazu auch Kommando TRANSMIT-BY-STREAM, Seite 796).

RET-CONTROL-VAR-NAME = *NONE
Die Return-Kontroll-Variable wird ignoriert, d.h. sie bleibt unverändert.

RET-CONTROL-VAR-NAME = <composed-name 1..255>(...)
Name der S-Variable, von der die übertragene Return-Kontroll-Variable gelesen wird. Die angegebene S-Variable muss eine Liste von Strukturen sein.

WRITE-MODE =
Gibt an, wie die Return-Kontroll-Variable bzw. die Liste von Strukturen verarbeitet wird.

WRITE-MODE = *EXTEND
Das letzte Element der angegebenen Liste wird entfernt.
Durch die nächste Übertragung wird das dann folgende letzte Element entfernt.
Wenn die Liste leer ist, wird sie wie bei RET-CONTROL-VAR-NAME = *NONE verarbeitet.

WRITE-MODE = *PREFIX
Das erste Element der angegebenen Liste wird entfernt.
Durch die nächste Übertragung wird das dann folgende erste Element entfernt.
Wenn die Liste leer ist, wird sie wie bei RET-CONTROL-VAR-NAME = *NONE verarbeitet.

TO = *SERVER(...)
Der S-Variablenstrom ist mit dem angegebenen Server verbunden.

SERVER-NAME = <structured-name 1..30>
Name des Servers.

SERVER-INFORMATION =
Information, die zum Server gesendet werden muss: z.B. der Name der Formatbibliothek für FHS.

SERVER-INFORMATION = *NONE
Es muss keine Information zum Server gesendet werden.

SERVER-INFORMATION = <c-string 1..1800>
Text der Mitteilung als String.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
2	0	SDP0531	Warnung vom Server; Prozess wird fortgesetzt
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	64	SDP0532	Server-Fehler; Kommando abgewiesen
	64	SDP0534	Interner Server-Fehler; Kommando abgebrochen. Server-Verbindung abgebrochen nach unerwartetem Ereignis oder bei mangelhaften oder fehlenden System-Ressourcen.
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommandos SHOW-STREAM-ASSIGNMENT, Seite 762

TRANSMIT-BY-STREAM, Seite 796

BEGIN-BLOCK
Kommandoblock einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Kommandoblöcke, die als logische Einheit behandelt werden sollen, werden mit dem Kommando BEGIN-BLOCK eingeleitet und mit dem Kommando END-BLOCK abgeschlossen. Diese Kommandoblöcke werden auch als BEGIN-Blöcke bezeichnet (BEGIN-Block siehe Abschnitt „Einfache Kommandoblöcke definieren“ auf Seite 93).

Der Kommandoblock kann über eine Marke benannt werden. Diese Marke kann auch als Sprungziel verwendet werden (zu Marken siehe Seite 54).

Der Operand PROGRAM-INPUT steuert die Behandlung von Kommandos innerhalb von Eingaben an Programme (Anweisungen und Daten) und die Behandlung der Returncodes von Programmanweisungen.

Format

BEGIN-**BLOCK**

PROGRAM-**INP**UT = ***STD** / ***MIX**ED-**WITH-CMD**(...)

***MIX**ED-**WITH-CMD**(...)

PROPAGATE-**STMT-RC** = ***STD** / ***TO-CMD-RC**

Operandenbeschreibung

PROGRAM-INPUT =
Legt fest, ob innerhalb von Eingaben an Programme (Anweisungen und Daten) auch Kommandos erlaubt sind und steuert die Behandlung der Returncodes von Programmanweisungen. Die Einstellung wird nicht in nachfolgende Prozeduraufrufe vererbt.

PROGRAM-INPUT = *STD
Kommandos werden wie im umschließenden BEGIN-Block behandelt. Im äußersten BEGIN-Block bzw. falls überhaupt kein BEGIN-Block existiert, müssen Kommandos durch die Kommandos HOLD-PROGRAM und RESUME-PROGRAM eingeschlossen werden. Das heißt, wenn die Datenzeilen durch ein Kommando (außer durch HOLD-PROGRAM) unterbrochen werden, wird die Dateiendebedingung (EOF) erzeugt.

PROGRAM-INPUT = *MIXED-WITH-CMD(...)
Es wird nicht zwischen Anweisungen / Datensätzen einerseits und Kommandos andererseits unterschieden. Das heißt, Kommandos erzeugen keine Dateiendebedingung (EOF).

PROPAGATE-STMT-RC =
Legt fest, ob Returncodes von Programmanweisungen als Kommando-Returncodes interpretiert werden **und** ob diese Returncodes eine SDF-P-Fehlerbehandlung auslösen sollen.

PROPAGATE-STMT-RC = *STD
Die Returncodes von Anweisungen und die SDF-P-Fehlerbehandlung werden wie im umschließenden BEGIN-Block behandelt. Im äußersten BEGIN-Block bzw. falls überhaupt kein BEGIN-Block existiert, werden die Returncodes von Anweisungen ignoriert; eine Fehlerbehandlung auf Kommandoebene ist nur durch Verwendung der vordefinierten Funktion STMT-SPINOFF() möglich.

PROPAGATE-STMT-RC = *TO-CMD-RC
Die Returncodes von Programmanweisungen stehen als Kommando-Returncodes zur Verfügung und steuern die SDF-P-Fehlerbehandlung. In der weiteren Verarbeitung wird nicht zwischen Returncodes von Kommandos und Programmanweisungen unterschieden.
Die SDF-P-Fehlerbehandlung wird nur ausgelöst, wenn SUBCODE1 ungleich Null ist.

Hinweis
Die Verwendung der vordefinierten Funktion STMT-SPINOFF( ) ist hier nicht sinnvoll, da sie an dieser Stelle niemals den Wert „YES“ liefert.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiele

Programmanweisungen und Kommandos voneinander separiert:

Programmanweisungen sind durch die Kommandos RESUME-PROGRAM und HOLD-PROGRAM eingeschlossen. Ein BEGIN-Block wird nicht benötigt.

```
/start-lms
/ hold-program
/ library = 'my-library'
/ write-text 'Elements of &library.'
/ resume-program
// show-element-attributes *library-element(&library.)
/ hold-program
/ library = 'my-second-library'
/ write-text 'Elements of &library.'
/ resume-program
// show-element-attributes *library-element(&library.)
//end
```

Programmanweisungen und Kommandos gemischt:

Die Kommandos und Programmanweisungen sind in einem BEGIN-Block eingeschlossen. Kommandos und Programmanweisungen werden automatisch erkannt.

```
/begin-block program-input = *mixed-with-cmd
/ start-lms
/ library = 'my-library'
/ write-text 'Elements of &library.'
// show-element-attributes *library-element(&library.)
/ library = 'my-second-library'
/ write-text 'Elements of &library.'
// show-element-attributes *library-element(&library.)
// end
/end-block
```

Returncodes von Programmanweisungen

Die Returncodes von Programmanweisungen werden wie die von Kommandos behandelt.

```
/begin-block program-input = *mixed-with-cmd(propagate-stmt-rc = *to-cmd-rc)
/ &* Start a program which generates statement returncodes
/ start-executable-program my-new-program
// my-statement1
/ if-cmd-error &* Test statement returncode
/ write-text 'Error during execution of my-statement'
/ write-text 'Maincode: &mc; Subcode1: &sc1; Subcode2: &sc2'
/ end-if
/ show-file-attributes &my-file.
```

```
/ if-cmd-error &* Test command returncode
/ write-text 'Error &mc during access to my-file'
/ end-if
// my-statement2
/ save-returncode &* Save statement returncode
/ if (maincode <> 'CMD0001')
/ write-text 'Warning &mc during execution of my-statement2'
/ end-if
// end
/end-block
```

Verarbeitung der STEP-Anweisung:

Die STEP-Anweisung setzt die durch fehlerhafte Anweisungen ausgelöste SDF-P-Fehlerbehandlung zurück. Wenn der Returncode benötigt wird, muss eine eigene Fehlerbehandlung in einem SDF-P-Fehlerblock erfolgen (/IF-BLOCK-ERROR oder /IF-CMD-ERROR statt //STEP).

```
/begin-block program-input = *mixed-with-cmd(propagate-stmt-rc = *to-cmd-rc)
/ &* Das folgende Programm liefert Returncodes für Programmanweisungen
/ start-executable-program <name>
// <anweisung>
// step
/ &* die SDF-P-Fehlerbehandlung wird zurückgesetzt
// end
/end-block
```

Verarbeitung der END-Anweisung:

Die END-Anweisung (//END) beendet die Programmausführung und gleichzeitig auch eine durch fehlerhafte Anweisungen ausgelöste SDF-P-Fehlerbehandlung. Wenn der Returncode benötigt wird, muss die Fehlerbehandlung in einem SDF-P-Fehlerblock vor der END-Anweisung erfolgen (/IF-BLOCK-ERROR oder /IF-CMD-ERROR).

```
/begin-block program-input = *mixed-with-cmd(propagate-stmt-rc = *to-cmd-rc)
/ &* Start a program which generates statement returncodes
/ start-executable-program <name>
// my-statement
// end
/ if-block-error
/ &* the //END statement has been processed,
/ &* the return code which is available is the command
/ &* return code of the /start-exec-prog and NOT the statement
/ &* return code of //my-statement
/ write-text 'Error &mc returned by /start-exec-prog'
/ end-if
/end-block
```

BEGIN-PARAMETER-DECLARATION
Deklaration der Prozedurparameter einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Die Prozedurparameter werden im Prozedurkopf mit dem Kommando DECLARE-PARAMETER deklariert. Wenn das Kommando DECLARE-PARAMETER mehrfach aufgerufen werden soll, werden diese Aufrufe in einen Kommandoblock eingeschlossen, der mit dem Kommando BEGIN-PARAMETER-DECLARATION eingeleitet und mit dem Kommando END-PARAMETER-DECLARATION abgeschlossen wird.

Das Kommando BEGIN-PARAMETER-DECLARATION ist auch notwendig, wenn ein- oder mehrere Male das Kommando OPEN-VARIABLE-CONTAINER in den Prozedurkopf eingefügt werden soll. Das ist erforderlich, wenn ein Prozedurparameter durch eine permanente Variable inialisiert werden soll (siehe dazu Abschnitt „S-Variable als Behälter“ auf Seite 169).

Format

BEGIN-**PAR**AMETER-**DECL**ARATION

Kommando-Returncode

Die Returncodes können nur auftreten, wenn das Kommando außerhalb des Prozedurkopfes verwendet wird. Fehler im Prozedurkopf erkennt SDF-P bei der Voranalyse und beendet den Prozeduraufruf.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

BEGIN-STRUCTURE
Statische Struktur deklarieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Wenn ein Strukturlayout deklariert wird, kennzeichnet BEGIN-STRUCTURE den Beginn der Deklaration des Strukturlayouts. Das Strukturlayout muss vor den statischen Strukturen, die sich darauf beziehen sollen, deklariert werden. Das Kommando END-STRUCTURE beendet die Deklaration des Strukturlayouts.

Wenn eine statische Struktur mit der Angabe *BY-SYSCMD deklariert wird , muss direkt auf das Kommando DECLARE-VARIABLE, in dem die Struktur deklariert wird, das Kommando BEGIN-STRUCTURE folgen. Es leitet die Elementdeklarationen ein.

(Zur Deklaration von Strukturen siehe Abschnitt „Variablendeklaration“ auf Seite 143.)

Format

BEGIN-**STRUC**TURE

NAME = ***NONE** / <structured-name 1..20>(...)

<structured-name 1..20>(...)

SCOPE = ***CURR**ENT / ***PROC**EDURE / ***TASK**(...)

***TASK**(...)

STATE = ***ANY** / ***NEW**

Operandenbeschreibung

NAME =
Kennzeichnet den Beginn der Deklaration eines Strukturlayouts oder den Beginn der Elementdeklaration einer statischen Struktur.

NAME = *NONE
Kennzeichnet den Beginn der Elementdeklaration der statischen Struktur, die im Kommando DECLARE-VARIABLE mit TYPE = *STRUCTURE(*BY-SYSCMD) eingeleitet wurde.

NAME = <structured-name 1..20>(...)
Name eines Strukturlayouts.
Mit NAME kann im Kommando DECLARE-VARIABLE mit TYPE = *STRUCTURE (DEFINITION = <structured-name 1..20>) auf das Strukturlayout Bezug genommen werden. Der dort angegebene Name für ein Strukturlayout muss mit dem übereinstimmen, der hier im Operanden NAME angegeben wird. Auf diese Weise wird der Struktur eindeutig ein Strukturlayout zugeordnet.

SCOPE =
Definiert den Geltungsbereich für das Strukturlayout.

SCOPE = *CURRENT
Entspricht in CALL-Prozeduren der Angabe PROCEDURE.
In INCLUDE-Prozeduren bedeutet CURRENT, dass das Strukturlayout in der aktuellen INCLUDE-Prozedur angelegt wird. Das Strukturlayout ist dann in dieser Include-Prozedur sichtbar (und in allen tiefergeschachtelten INCLUDE-Prozeduren).
Das Strukturlayout verschwindet mit dem (dynamischen) Ende der CALL- bzw. INCLUDE-Prozedur.

SCOPE = *PROCEDURE
Das Strukturlayout wird in der aktuellen CALL-Prozedur angelegt. In einer Include-Prozedur ist die aktuelle Prozedur immer die aufrufende CALL-Prozedur.
Das Strukturlayout ist in der aktuellen CALL-Prozedur sichtbar sowie in allen Prozeduren, die in der aktuellen CALL-Prozedur mit INCLUDE-PROCEDURE aufgerufen werden. Das Layout wird in der aktuellen CALL-Prozedur angelegt. Es bleibt daher bis zum Ende dieser Prozedur bestehen, auch wenn es in einer INCLUDE-Prozedur deklariert wurde, die in der aktuellen CALL-Prozedur aufgerufen wurde.

SCOPE = *TASK(...)
Die Lebensdauer des Strukturlayouts wird durch die Lebensdauer der Task bestimmt. Das Strukturlayout ist sichtbar in allen Prozeduren, in denen keine Struktur mit dem selben Namen und mit anderem Geltungsbereich (d.h. *CURRENT oder *PROCEDURE) deklariert wurde.

STATE = *ANY
Wenn in der Task bereits ein Strukturlayout dieses Namens existiert, so wird das bereits bestehende Strukturlayout genommen. Es wird kein neues Strukturlayout angelegt. Bei solchen Mehrfachdeklarationen gilt die Regel, dass das hier deklarierte Strukturlayout mit dem schon vorhandenen übereinstimmen muss. Wenn in der Task noch kein Strukturlayout dieses Namens existiert, wird ein neues Strukturlayout definiert.

STATE = *NEW
Das Strukturlayout darf in der Task noch nicht vorhanden sein. Es wird ein neues Strukturlayout deklariert.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommando DECLARE-ELEMENT, Seite 600.

CALL-PROCEDURE
Kommandofolge starten

Anwendungsgebiet: **PROCEDURE**

Das Kommando ist Bestandteil von BS2000/OSD-BC (Stand der Beschreibung: V7.0). Der Aufrufer benötigt (im Gegensatz zu den SDF-P-Kommandos) die Privilegien STD-PROCESSING bzw. HARDWARE-MAINTENANCE.

Kommandobeschreibung

Das Kommando CALL-PROCEDURE startet eine gespeicherte Kommandofolge (Prozedur). Bei Abarbeitung werden darin enthaltene symbolische Parameter durch die im Kommandoaufruf angegebenen Werte (Operand PROCEDURE-PARAMETERS) ersetzt.

S-Prozeduren:

– Als Aktualparameter können Variablen übergeben werden, über die die Prozedur auch Ausgabewerte zurückliefert.
– Die Aktualparameter können als Stellungs- oder als Schlüsselwortparameter übergeben werden. Die Reihenfolge der Stellungsparameter entspricht dabei der dynamischen Reihenfolge der DECLARE-PARAMETER-Kommandos; die Namen der Schlüsselwörter entsprechen den Namen der formalen Prozedurparameter. Die Schlüsselwörter dürfen bis zur Eindeutigkeit abgekürzt werden.
– Die Protokolierung wird im Kommandoaufruf eingestellt; ebenso, ob ein bereits geladenes Programm entladen werden darf.

Prozeduren können gespeichert werden als:

– katalogisierte (auch temporäre) SAM- oder ISAM-Datei mit Sätzen variabler Länge
– Element vom Typ J oder SYSJ einer PLAM-Bibliothek
– S-Variable vom Typ Liste

Prozedur-Formate:

– Text-Prozedur
  Die S-Prozedur liegt im ursprünglichen Textformat vor. Volle SDF-P-Funktionalität kann nur verwendet werden, wenn bei Aufruf der Prozedur das kostenpflichtige Subsystem SDF-P geladen ist. In Bibliotheken sollte für Text-Prozeduren der Elementtyp J verwendet werden.
– Objekt-Prozedur
  Die S-Prozedur im Textformat wurde mit dem Kommando COMPILE-PROCEDURE in ein Zwischenformat kompiliert. Diese Objekt-Prozedur kann, unabhängig von der Verfügbarkeit des Subsystems SDF-P, die volle SDF-P-Funktionalität nutzen (mit Ausnahme des Kommandos COMPILE-PROCEDURE). In Bibliotheken sollte für Objekt-Prozeduren der Elementtyp SYSJ verwendet werden (Default-Wert bei COMPILE-PROCEDURE).

Format

CALL-PROCEDURE	Kurzname: **CL** / **CLP**

```
FROM-FILE = <filename 1..54 without-gen> / *LIBRARY-ELEMENT(...) / *VARIABLE(...)
    *LIBRARY-ELEMENT(...)
        LIBRARY = <filename 1..54 without-gen>
        ,ELEMENT = <composed-name 1..64>(...)
            <composed-name 1..64>(...)
                VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
        ,TYPE = *STD / *BY-LATEST-MODIFICATION / <alphanum-name 1..8>
    *VARIABLE(...)
        VARIABLE-NAME = <composed-name 1..255>
,PROCEDURE-PARAMETERS = *NO / <text 0..1800 with-low>
,LOGGING = *PARAMETERS(...) / YES / *NO /
    *PARAMETERS(...)
        CMD = *BY-PROC-TEST-OPTION / *YES / *NO
        ,DATA = *BY-PROC-TEST-OPTION / *YES / *NO
,UNLOAD-ALLOWED = *YES / *NO
,EXECUTION = *YES / *NO
```

Operandenbeschreibung

FROM-FILE = <filename 1..54 without-gen> / *LIBRARY-ELEMENT(...) / *VARIABLE(...)
Name der Prozedurdatei.

FROM-FILE = *LIBRARY-ELEMENT(...)
Die Prozedur ist in einem PLAM-Bibliothekselement abgelegt.

LIBRARY = <filename 1..54 without-gen>
Name der PLAM-Bibliothek, die die Prozedurdatei als Element (Typ J oder SYSJ; siehe Operand TYPE) enthält.

ELEMENT = <composed-name 1..64>(...)
Name des Elements.

VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
Version des Bibliothekelements. Voreingestellt ist HIGHEST-EXISTING, d.h. die Prozedur wird dem Element mit der höchsten Version entnommen.

TYPE = *STD / *BY-LATEST-MODIFICATION / <alphanum-name 1..8>
Bestimmt den Elementtyp, unter dem die Prozedurdatei in der PLAM-Bibliothek abgelegt ist.

TYPE = *STD
Die Prozedurdatei kann als Element des Typs SYSJ oder J abgelegt sein.
Das angegebene Element wird zuerst unter den Elementen vom Typ SYSJ gesucht. Falls es dort nicht existiert, wird unter den Elementen vom Typ J weitergesucht.

Eine Nicht-S-Prozedur kann nur als Element vom Typ J vorliegen.
Eine S-Prozedur kann sowohl als Text-Prozedur (ursprüngliches Textformat) als auch als Objekt-Prozedur (kompiliertes Zwischenformat) vorliegen. Zur Vereinfachung der Verwaltung beider Formate in einer Bibliothek sollten Text-Prozeduren als Element vom Typ J und Objekt-Prozeduren als Element vom Typ SYSJ ablegt sein. Mit dem Kommando COMPILE-PROCEDURE (Bestandteil des kostenpflichtigen Subsystems SDF-P) wird aus einer Text-Prozedur vom Typ J standardmäßig eine Objekt-Prozedur vom Typ SYSJ (Default-Wert) erzeugt.
Die Angabe von TYPE=*STD (Default-Wert) stellt sicher, dass bei Einhaltung dieser Konvention Objekt-Prozeduren bevorzugt aufgerufen werden.

TYPE = *BY-LATEST-MODIFICATION
Die Prozedurdatei kann als Element des Typs SYSJ oder J abgelegt sein.
Existiert das angegebene Element sowohl als Typ SYSJ als auch J, wird das zuletzt geänderte Element aufgerufen. Bei gleichem Zeitstempel wird des Element vom Typ SYSJ aufgerufen.
Die Angabe von TYPE=*BY-LATEST-MODIFICATION stellt sicher, dass z.B. während der Testphase bei der Erstellung bzw. Erweiterung einer Prozedur das aktuellste Element aufgerufen wird.

TYPE = <alphanum-name 1..8>
Die Prozedurdatei wird ausschließlich unter den Elementen des angegebenen Typs gesucht.

FROM-FILE = *VARIABLE(...)
Die Prozedur ist in einer S-Variablen vom Typ Liste abgelegt.

VARIABLE-NAME = <composed-name 1..255>
Name der S-Variablen.

PROCEDURE-PARAMETERS = *NO / <text 0..1800 with-low>
Definiert die aktuellen Prozedurparameter; die Parameter müssen in Klammern eingeschlossen werden.
Zu Prozedurparametern siehe auch „Prozedurparameter übergeben“ auf Seite 108ff.

LOGGING = *PARAMETERS(...) / *YES / *NO
Steuert die Protokollierung des Prozedurablaufs.
Der Operand LOGGING wird bei Aufruf von *Nicht-S-Prozeduren* ignoriert, da die Protokollierung dort nur im Prozedurkopf vereinbart werden kann (siehe BEGIN-PROCEDURE, Operand LOGGING).
Bei Protokollierung einer S-Prozedur wird jede abgearbeitete Prozedurzeile mit vorangestellter Zeilennummer und Prozedurstufe ausgegeben.

Zu Protokollierung siehe auch „Protokollierung einstellen“ auf Seite 84ff.

LOGGING = *PARAMETERS(...)
Die Protokollierung kann getrennt eingestellt werden für Kommando-/Anweisungszeilen und Datenzeilen.

CMD = *BY-PROC-TEST-OPTION / *YES / *NO
Gibt an, ob Kommandos protokolliert werden sollen. Voreingestellt ist BY-PROC-TEST-OPTION, d.h. keine Protokollierung (entspricht *NO) bzw. der Wert, den der Benutzer mit dem Kommando MODIFY-PROC-TEST-OPTIONS als Voreinstellung gewählt hat (Bestandteil des kostenpflichtigen Subsystems SDF-P).

DATA = *BY-PROC-TEST-OPTION / *YES / *NO
Gibt an, ob Datenzeilen protokolliert werden sollen. Voreingestellt ist BY-PROC-TEST-OPTION, d.h. keine Protokollierung (entspricht *NO) bzw. der Wert, den der Benutzer mit dem Kommando MODIFY-PROC-TEST-OPTIONS als Voreinstellung gewählt hat (Bestandteil des kostenpflichtigen Subsystems SDF-P).

UNLOAD-ALLOWED = *YES / *NO
Legt fest, ob ein Programm, das zum Zeitpunkt des Prozeduraufrufs geladen ist, entladen werden darf.
Der Schutz vor Programmentladung ist nur für das Entladen mit den Kommandos LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM) und CANCEL-PROGRAM gewährleistet.
Die Angabe YES wird ignoriert, wenn der Prozeduraufruf aus einer Prozedur erfolgt, für die UNLOAD-ALLOWED=*NO vereinbart wurde.

EXECUTION = *YES / *NO
Legt fest, ob die Prozedur nur zu Testzwecken analysiert werden oder auch ausgeführt werden soll.
Für *Nicht-S-Prozeduren* kann nur EXECUTION=*YES vereinbart werden.
Der Test ist mit MODIFY-SDF-OPTIONS (Operand MODE) möglich.

Kommando-Returncode

Die nachfolgenden Kommando-Returncodes werden nur zurückgegeben, wenn die aufgerufene Prozedur selbst keinen Kommando-Returncode liefert (z.B. EXIT-PROCEDURE wegen Fehlers nicht ausgeführt).

Kommando-Returncodes, deren Maincode mit „SSM“ beginnt, werden nur bei Aufruf einer Nicht-S-Prozedur zurückgegeben.

Kommando-Returncodes, deren Maincode mit „SDP“ beginnt, werden nur bei Aufruf einer S-Prozedur zurückgegeben.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
2	0	SSM2058	Protokoll-Typ-Fehler
2	0	SSM2065	EOF auf Prozedurdatei, /END-PROC simuliert
	1	SSM2036	Unvollständiger Operand
	1	SSM2054	Symbolischer Operandenfehler
	1	SSM2055	Symbolischer Operandenfehler in /BEGIN-PROC
	1	SDP0138	Fehler bei Voranalyse der Text-Prozedur oder Objekt-Prozedur fehlerhaft
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0093	Nicht-S-Prozedur kann nur Element vom Typ J sein
	64	SDP0144	Fehler bei Parameterübertragung
	64	SSM2052	DVS-Fehler (Open-Fehler)
	64	SSM2053	keine SAM/ISAM-Datei oder Datei beginnt nicht mit /BEGIN-PROC bzw. /PROC
	64	SSM2056	Parameter von /CALL-PROC und /BEGIN-PROC passen nicht zusammen
	64	SSM2061	Fehler bei Zugriff auf Bibilothekselement
	64	SSM2064	Prozedurdatei kann nicht von entferntem Rechner geholt werden
	130	SDP0099	Kein Adressraum mehr verfügbar
xx	xx	xxxxxxx	sonstige Returncodes der aufgerufenen Prozedur

CLOSE-VARIABLE-CONTAINER
Variablenbehälter schließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando CLOSE-VARIABLE-CONTAINER werden die angegebenen Variablenbehälter geschlossen.

Hinweis

Wenn ein Variablenbehälter geschlossen wird, bevor der Geltungsbereich seiner Variablen erschöpft ist, bleiben die Variablen deklariert, aber es kann auf sie nicht länger zugegriffen werden. Jeder Zugriff wird mit der Fehlermeldung SDP1030 abgewiesen.

Format

CLOSE-VARIABLE-**CONTAINER**
CONTAINER-NAME = <composed-name 1..64 with-wild(80)> / list-poss(2000): <composed-name 1..64>

Operandenbeschreibung

CONTAINER-NAME =
Namen des Variablenbehälters.

CONTAINER-NAME = list-poss(2000): <composed-name 1..64>
Namenliste der Variablenbehälter.

CONTAINER-NAME = <composed-name 1..64 with-wild(80)>
Variablenbehälter, der das angegebene Suchmuster erfüllt.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommando SHOW-VARIABLE-CONTAINER-ATTR, Seite 788.

COMPILE-PROCEDURE
Prozedur kompilieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando COMPILE-PROCEDURE konvertiert eine S-Prozedur in eine kompilierte Prozedur, d.h. in ein Zwischenformat, das dann auch in Umgebungen benutzt werden kann, in denen das Subsystem SDF-P nicht zur Verfügung steht.

In kompilierten Prozeduren kann der volle Umfang von SDF-P benutzt werden. Das gilt auch, wenn diese Prozeduren in einer Umgebung gestartet werden, die nur SDF-P-BASYS enthält.

Hinweise

- Das Kommando COMPILE-PROCEDURE ist Teil des Subsystems SDF-P. Es wird abgewiesen, wenn dieses Subsystem nicht geladen ist (auch wenn es in einer kompilierten Prozedur enthalten ist)
- Es ist nicht möglich, bei den Operanden FROM-FILE und TO-FILE Wildcards anzugeben. Bei beiden Operanden ist es aber möglich, sowohl Dateien als auch Bibliothekselemente anzugeben.
- Inkonsistente Angaben bei Ein- und Ausgabe werden als Semantikfehler bewertet, d.h. es wird die dafür vorgesehene Fehlerbehandlung angestoßen.
- Fehlermeldungen werden nur nach SYSOUT gesendet. Diese Meldungen sind bis auf ein paar zusätzliche Compiler-spezifische Meldungen identisch mit denen von CALL-PROCEDURE bei einer aufgerufenen Prozedur.

  Die compiler-spezifischen Meldungen sind:
  ```
  SDP1300 PROZEDUR-COMPILER VERSION '(&00)' GESTARTET
  SDP1301 PROZEDUR-COMPILER NORMAL BEENDET
  SDP1302 PROZEDUR-COMPILER ABNORMAL BEENDET
  ```
- Auch S-Prozeduren, die keine oder nur teilweise kostenpflichtige Funktionen von SDF-P nutzen, können mit COMPILE-PROCEDURE konvertiert und bei Installationen ohne das Subsystem SDF-P mit dem Subsystem SDF-P-BASYS ausgeführt werden.

Format

COMPILE-PROCEDURE

FROM-FILE = <filename 1..54 without-gen> / ***LIB**RARY-**ELEM**ENT(...)

***LIB**RARY-**ELEM**ENT(...)

LIBRARY = <filename 1..54 without-gen>

,**ELEM**ENT = <composed-name 1..64>(...)

<composed-name 1..64>(...)

VERSION = ***HIGH**EST-**EXIST**ING / ***UP**PER-**LIM**IT / <composed-name 1..24>

,**TYPE** = **J** / <alphanum-name 1..8>

,**TO-FILE** = <filename1..54without-gen> **/ *LIB**RARY-**ELEM**ENT(...) / ***DUMMY**

***LIB**RARY-**ELEM**ENT(...)

LIBRARY = ***SAME** / <filename 1..54 without-gen>

,**ELEM**ENT = ***SAME**(...) / <composed-name 1..64>(...)

***SAME**(...)

VERSION = ***SAME** / ***UP**PER-**LIM**IT / ***INCREMENT** / *HIGHEST-**EXIST**ING / <composed-name 1..24>

<composed-name 1..64>(...)

VERSION = ***SAME** **/ *UP**PER-**LIM**IT / ***INCREMENT** / *HIGHEST-**EXIST**ING / <composed-name 1..24>

,**TYPE** = **SYSJ** / <alphanum-name 1..8>

Operandenbeschreibung

FROM-FILE =
Bezeichnet die Quellprozedur.

FROM-FILE = <filename 1..54 without-gen>
Name der Prozedurdatei.

FROM-FILE = *LIBRARY-ELEMENT(...)
Die Prozedur ist in einer PLAM-Bibliothek gespeichert.

LIBRARY = <filename 1..54 without-gen>
Name der PLAM-Bibliothek, die die Prozedur enthält.

ELEMENT = <composed-name 1..64>(...)
Name des Elements

VERSION =
Bezeichnet die Versionsnummer des Elements.

VERSION = *HIGHEST-EXISTING
Wählt die höchste existierende Versionsnummer.

VERSION = *UPPER-LIMIT
Wählt die höchste mögliche Versionsnummer.

VERSION = <composed-name 1..24>
Wählt die angegebene Versionsnummer.

TYPE= J / <alphanum-name 1..8>
Typ des Elements. Default-Typ ist J.

TO-FILE =
Gibt an, wo die kompilierte Prozedur gespeichert werden soll.

TO-FILE = <filename 1..54 without-gen>
Name der Datei, in der die kompilierte Prozedur gespeichert werden soll.

TO-FILE = *DUMMY
Es muss keine Prozedur kompiliert werden. Das Kommando überprüft nur die Prozedur (siehe dazu CALL-PROCEDURE EXECUTION = *NO).

TO-FILE = *LIBRARY-ELEMENT(...)
Die kompilierte Prozedur wird in einer PLAM-Bibliothek gespeichert.

LIBRARY = *SAME / <filename 1..54 without-gen>
Name der PLAM-Bibliothek, die die kompilierte Prozedur enthalten soll. Der Defaultwert ist die Bibliothek der Quellprozedur.

ELEMENT =
Name des Elements für die kompilierte Prozedur.

ELEMENT = *SAME(...)
Der Name des Elements ist derselbe wie bei der Quellprozedur.

VERSION =
Bezeichnet die Versionsnummer des Elements (nur bei S-Prozeduren).

VERSION = *SAME
Wählt dieselbe Versionsnummer wie bei der Quellprozedur.

VERSION = *UPPER-LIMIT
Wählt die höchste mögliche Versionsnummer.

VERSION = *INCREMENT
Die Versionsnummer des Elements wird erhöht.

VERSION = *HIGHEST-EXISTING
Wählt die höchste existierende Versionsnummer.

VERSION = <composed-name 1..24>
Wählt die angegebene Versionsnummer.

ELEMENT = <composed-name 1..64>(...)
Name des Elements

VERSION =
Bezeichnet die Versionsnummer des Elements (nur bei S-Prozeduren).

VERSION = *SAME
Wählt dieselbe Versionsnummer wie bei der Quellprozedur.

VERSION = *UPPER-LIMIT
Wählt die höchste mögliche Versionsnummer.

VERSION = *INCREMENT
Die Versionsnummer des Elements wird erhöht.

VERSION = *HIGHEST-EXISTING
Wählt die höchste existierende Versionsnummer.

VERSION = <composed-name 1..24>
Wählt die angegebene Versionsnummer.

TYPE= SYSJ / <alphanum-name 1..8>
Typ des Elements. Default-Typ ist SYSJ.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0138	Fehler bei Voranalyse der Prozedur garantierte Meldung: SDP0138
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Installation A: SDF-P ist gestartet

```
/ASSIGN-SYSLST TO-FILE=THE-RESULT-LISTING
/MODIFY-JOB-OPTIONS LISTING=*YES
/
/"EINE FEHLERHAFTE PROZEDUR WIRD KOMPILIERT"
/COMPILE-PROCEDURE *LIB(LIB=THE-PROC-LIB,EL=THE-ERRONEOUS-PROC),-
/                  TO-FILE=*LIB-ELEMENT
SDP1300 PROZEDUR-COMPILER VERSION 'V2.4A20' WURDE GESTARTET
SDP0201 ES WURDE EIN UNVOLLSTÄNDIGES BLOCKSCHLIESSKOMMANDO BENUTZT
:
SDP1302 PROZEDUR-COMPILER WURDE MIT FEHLER BEENDET
/
/MODIFY-JOB-OPTIONS LISTING=*NO
/ASSIGN-SYSLST TO-FILE=*PRIMARY
/"LISTING: THE-RESULT-LISTING"
/
/...

/"NACH DER KORREKTUR"
/COMPILE-PROCEDURE *LIB(LIB=THE-PROC-LIB,EL=THE-CORRECT-PROC),-
/                  TO-FILE=*LIB-ELEMENT
SDP1300 PROZEDUR-COMPILER VERSION 'V2.4A20' WURDE GESTARTET
SDP0302 PROZEDUR-COMPILER WURDE BEENDET
```

Installation B: SDF-P ist nicht gestartet:

```
/CALL-PROCEDURE *LIB(LIB=THE-PROC-LIB,EL=THE-CORRECT-PROC)

%          1  1 /SET-PROCEDURE-OPTIONS
%          2  1 /DECLARE-VARIABLE A(TYPE=INTEGER,INITIAL-VALUE=1)
%          3  1 /WHILE (A < 3)
%          4  1 /IF (NOT IS-CAT-FILE ('MYFILE.1'))
%          4  1 /END-IF
%          5  1 /A=A+1
%          6  1 /END-WHILE
%          4  1 /IF (NOT IS-CAT-FILE ('MYFILE.2'))
%          4  1 /CREATE-FILE MYFILE.2
%          4  1 /END-IF
%          5  1 /A=A+1
%          6  1 /END-WHILE
%             1 /EXIT-PROCEDURE ERROR=*NO
```

CYCLE
Schleifendurchlauf abbrechen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando CYCLE kann in Schleifenblöcken (FOR-, WHILE-, REPEAT-Block) aufgerufen werden. Es bricht dann den aktuellen Schleifendurchlauf ab und setzt den Prozedurlauf mit der Ausführung des Abschlusskommandos des Schleifenblocks (END-FOR, END-WHILE, UNTIL) fort. Anschließend wird die Schleifenbedingung erneut überprüft und wenn nötig, der nächste Schleifendurchlauf gestartet. Zu CYCLE siehe auch Abschnitt „Sprung ans Schleifenende“ auf Seite 104.

Die Ausführung des Kommandos CYCLE kann von einer Bedingung abhängig gemacht werden.

Format

CYCLE
BLOCK = ***LAST** / ***ALL** / <structured-name 1..255> ,**COND**ITION = ***NONE** / <text 1..1800 with-low *bool-expr*>

Operandenbeschreibung

BLOCK =
Bezeichnet die Schleife bzw. den Schleifenblock.

BLOCK = *LAST
Bezeichnet den unmittelbar umgebenden Schleifenblock; der Prozedurlauf wird mit dem nächsten Schleifen-Abschlusskommando fortgesetzt.

BLOCK = *ALL
Bezeichnet bei geschachtelten Schleifen den äußersten Schleifenblock; der Prozedurlauf wird mit dem letzten Schleifen-Abschlusskommando fortgesetzt.

BLOCK = <structured-name 1..255>
Name der Schleife, die abgebrochen wird; der Schleifenname entspricht der Marke im Kommandoaufruf des Schleifen-Einleitungskommandos.

CONDITION =
Beschreibt eine Bedingung für die Ausführung des CYCLE-Kommandos.

CONDITION = *NONE
Das Kommando wird immer ausgeführt.

CONDITION = <text 1..1800 with-low *bool-expr*>
Das CYCLE-Kommando wird nur ausgeführt, wenn der angegebene boolsche Ausdruck den Wert „TRUE“ ergibt.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler (unkorrekter Ausdruck)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Beispiel 1

```
/LOOP: WHILE (BED < 9)
:
:
/IF (EING='*SKIP')
/WR-TEXT 'Element wird übergangen'
/CYCLE BLOCK=LOOP
/END-IF
:
:
/END-WHILE       "Dieses Kommando wird nach CYCLE ausgefuehrt"
```

Beispiel 2

```
/J=0
/FOR I=('Zeile 1','Zeile 2','Zeile 3','Zeile 4)
/J=J+1
/CYCLE BLOCK=*LAST,CONDITION=(J=3)
/SHOW-VARIABLE I
/END-FOR
```

Ausgabe:

```
I = Zeile 1
I = Zeile 2
I = Zeile 4
```

Das dritte Element der Liste (I=3) wird nicht ausgegeben, d.h. der Schleifendurchlauf wird beendet und der Prozedurlauf bei END-FOR fortgesetzt.

DECLARE-CONSTANT
Variable mit konstantem Wert deklarieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando DECLARE-CONSTANT deklariert eine oder mehrere Variablen und weist ihnen einen konstanten Wert zu. So sind diese Werte vor Überschreiben geschützt. Variablen mit konstantem Wert können nahezu so behandelt werden wie herkömmliche Variablen. Allerdings kann der Wert nicht mit SET-VARIABLE geändert und auch nicht mit FREE-VARIABLE entfernt werden.

Format

DECLARE**-CONSTANT**

VARIABLE**-NAME** = list-poss(2000): <structured-name 1..20>(...)

<structured-name 1..20>(...)

VALUE = <text 0..1800 with-low *expr*>

,**TYPE** = ***ANY** / ***STRING** / ***INTEGER** / ***BOOL**EAN

,**SCOPE** = ***CURR**ENT(...) / ***PROC**EDURE(...) / ***TASK**(...)

***CURR**ENT(...)

IMPORT-ALLOWED = ***NO** / ***YES**

***PROC**EDURE(...)

IMPORT-ALLOWED = ***NO** / ***YES**

***TASK**(...)

STATE = ***ANY** / ***NEW** / ***OLD**

,**CONTAINER** = ***STD** / <composed-name 1..64> **/ *VAR**IABLE(...)

***VAR**IABLE(...)

VARIABLE**-NAME** = <structured-name 1..20>

,**SCOPE** = ***VISIBLE** / ***TASK**

Operandenbeschreibung

VARIABLE-NAME = list-poss (2000): <structured-name 1..20>(...)
Name der zu deklarierenden konstanten Variablen. Das darf allerdings nur eine einfache Variable, also keine zusammengesetzte Variable oder ein Variablenelement sein.

VALUE = <text 0..1800 with-low *expr*>
Weist einer Variablen einen konstanten Wert zu; der Wert muss zum Datentyp der Variablen passen, er kann auch als Ausdruck angegeben werden.

TYPE =
Weist der Variablen den Datentyp zu.

TYPE = *ANY
Der Variablen kann als Datentyp STRING, INTEGER oder BOOLEAN zugewiesen werden. Aber nach der Deklaration der konstanten Variablen kann er nicht mehr geändert werden.

TYPE = *STRING
Weist der Variablen den Datentyp STRING zu.
Wertebereich: beliebige Zeichenkette

TYPE = *INTEGER
Weist der Variablen den Datentyp INTEGER zu.
Wertebereich: Ganzzahl zwischen -2^{31} und $2^{31}-1$

TYPE = *BOOLEAN
Weist der Variablen den Datentyp BOOLEAN zu.
Wertebereich: TRUE, FALSE, YES, NO, ON, OFF.

SCOPE =
Definiert den Geltungsbereich der Variablen.

SCOPE = *CURRENT(...)
Die Variable ist eine prozedurlokale Variable.
Entspricht in Call-Prozeduren der Angabe PROCEDURE.
In Include-Prozeduren bedeutet CURRENT, dass die Variable in der aktuellen Include-Prozedur angelegt wird. Sie ist dann in dieser Include-Prozedur und in allen tiefergeschachtelten Include-Prozeduren sichtbar (= Geltungsbereich Include).

IMPORT-ALLOWED =
Gibt an, ob die Variable mit IMPORT-VARIABLE in einer gerufenen Prozedur importiert werden kann.

IMPORT-ALLOWED = *NO
Die Variable kann nicht mit IMPORT-VARIABLE in einer gerufenen Prozedur importiert werden.

IMPORT-ALLOWED = *YES
Die Variable kann mit IMPORT-VARIABLE in einer gerufenen Prozedur importiert werden.

SCOPE = *PROCEDURE(...)
Die Variable ist eine prozedurlokale Variable mit dem Geltungsbereich Prozedur.
Die Variable wird in der aktuellen Prozedur angelegt.
Bei Include-Prozeduren ist die aktuelle Prozedur jeweils die umgebende Call-Prozedur, von der aus die Include-Prozedur aufgerufen wurde.
Die Variable ist in dieser Prozedur sichtbar sowie in allen tiefergeschachtelten Include-Prozeduren.

IMPORT-ALLOWED =
Gibt an, ob die Variable mit IMPORT-VARIABLE in einer gerufenen Prozedur importiert werden kann.

IMPORT-ALLOWED = *NO
Die Variable kann nicht mit IMPORT-VARIABLE in einer gerufenen Prozedur importiert werden.

IMPORT-ALLOWED = *YES
Die Variable kann mit IMPORT-VARIABLE in einer gerufenen Prozedur importiert werden.

SCOPE = *TASK(...)
Die Variable ist eine taskglobale Variable.
Wird sie in einer Include-Prozedur angelegt, so ist sie auch in der umgebenden Call-Prozedur, von der aus die Include-Prozedur aufgerufen wurde, und in allen tiefergeschachtelten Include-Prozeduren sichtbar.

STATE = *ANY
Existiert in der Task bereits eine Variable mit dem angegebenen Namen, so wird diese verwendet, andernfalls wird eine neue Variable deklariert.

STATE = *NEW
In der Task darf keine Variable mit dem angegebenen Namen existieren.

STATE = *OLD
In der Task muss eine Variable mit dem angegebenen Namen existieren. Die aktuelle Variablendeklaration muss dann mit der Deklaration der bereits vorhandenen Variablen übereinstimmen.

CONTAINER = *STD
Der Variablen wird kein Variablenbehälter zugeordnet. Der Wert der Variablen wird im Klasse-5-Speicher abgelegt.

CONTAINER = <composed-name 1..64>
Verbindet die aktuell deklarierte Variable mit dem hier angegebenen Variablenbehälter. Dieser Variablenbehälter muss geöffnet sein. Die Angabe von „STD“ ist hier nicht erlaubt, da „STD“ nicht als permanent-bestehender Variablenbehälter interpretiert wird.

CONTAINER = *VARIABLE(...)
Verbindet die aktuell deklarierte Variable über einen Link-Mechanismus mit einer anderen Variablen, die in dieser Prozedur bereits definiert ist. Diese Variable wird dann als Variablenbehälter bezeichnet.
Strukturelemente können nicht als Variablenbehälter angegeben werden.

VARIABLE-NAME = <structured-name 1..20>
Name einer Variablen, die in der Prozedur bereits definiert wurde. Die Attribute der Variablen, die als Variablenbehälter dient, und die Attribute der aktuell deklarierten Variablen müssen miteinander verträglich sein. Diese Variable muss auch mit einem konstanten Wert (demselben konstanten Wert wie bei beim Operanden VARIABLE-NAME) und einem konstanten Datentyp deklariert sein.

SCOPE =
Geltungsbereich der Behälter-Variablen

SCOPE = *VISIBLE
Die Variable ist sichtbar.

SCOPE = *TASK
Task-Variable

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; Element schon deklariert
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler garantierte Meldungen: SDP1018, SDP1030
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/DECLARE-CONSTANT KBYTE(TYPE=*INTEGER,VALUE=1024)
/DECLARE-CONSTANT MBYTE(TYPE=*INTEGER,VALUE=KBYTE*KBYTE)
/DECLARE-CONSTANT PAMPAGE(TYPE=*INTEGER,VALUE=2*KBYTE)

/DECLARE-VARIABLE FILE(TYPE=*STRUCTURE)
/DECLARE-VARIABLE FILES(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST

/EXECUTE-CMD (SHOW-FILE-ATTRIBUTES *ALL),STRUCTURE-OUTPUT=FILES,-
/            TEXT-OUTPUT=*NO
/
/FOR FILE=*LIST(FILES)
/   IF (FILE.F-SIZE * PAMPAGE >= 5 * MBYTE)
/       WRITE-TEXT 'VERY HUGE FILE &(FILE.SHORT-F-NAME)'
/   ELSE-IF (FILE.F-SIZE * PAMPAGE >= 100 * KBYTE)
/       WRITE-TEXT 'HUGE FILE &(FILE.SHORT-F-NAME)'
/   END-IF
/END-FOR
```

Diese Prozedur deklariert die drei Variablen KBYTE, MBYTE und PAMPAGE. Sie werden mit konstanten Werten deklariert, um ihre Korrektheit während der gesamten Prozedur sicherzustellen. Ihre Werte können nicht verändert werden.

Diese Variablen werden gebraucht, um die Größe von Dateien der gegenwärtigen Benutzerkennung zu testen.

DECLARE-ELEMENT
Strukturelement deklarieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Strukturelemente können einfache oder zusammengesetzte Variablen (Arrays, Strukturen, Listen) sein. In der folgenden Operandenbeschreibung werden daher folgende Bezeichnungen verwendet: „einfache Variable“ (wenn das Strukturelement eine einfache Variable ist), „zusammengesetzte Variable“ (wenn das Strukturelement selbst eine zusammengesetzte Variable ist) und „Variable“ (wenn die Aussage sowohl für einfache als auch für zusammengesetzte Variablen gilt).

Variablenattribute, die im Kommando DECLARE-ELEMENT nicht definiert werden können, werden von der übergeordneten Struktur übernommen (z.B. SCOPE-Attribute von BEGIN-STRUCTURE oder DECLARE-VARIABLE).

Ist das Strukturelement eine zusammengesetzte Variable, müssen deren Elemente einzeln initialisiert werden. Zusammengesetzte Variablen können nicht als Ganzes initialisiert werden. Es können auch Elemente von dynamischen Strukturen mit diesem Kommando deklariert werden.

Format

DECLARE-**ELEM**ENT

NAME = list-poss(2000): <composed-name 1..255>(...)

<composed-name 1..255>(...)

INITIAL-**VAL**UE = ***NONE** / <text 0..1800 with-low *expr*>

,**TYPE** = ***ANY** / ***STRING** / ***INTEGER** / ***BOOL**EAN / ***STRUC**TURE(...)

***STRUC**TURE(...)

DEFINITION = ***DYN**AMIC / ***BY-SYSCMD** / <structured-name 1..20>

,**MULT**IPLE-**ELEM**ENTS = ***NO** / ***ARRAY**(...) / ***LIST**(...)

***ARRAY**(...)

LOWER-**BOUND** = **0** / ***NONE** / <integer -2147483648..2147483647>

,**UP**PER-**BOUND** = ***NONE** / <integer -2147483648..2147483647>

***LIST**(...)

LIMIT = ***NONE** / <integer 1..2147483647>

Operandenbeschreibung

NAME = list-poss (2000): <composed-name 1..255>(...)
Deklariert den Variablennamen.

INITIAL-VALUE = *NONE
Die Variable wird nicht initialisiert, ihr wird kein Anfangswert zugewiesen.
Das bedeutet: Der Inhalt einer bereits initialisierten Variablen bleibt unverändert; eine neue Variable erhält keinen Wert, sodass ein Lesezugriff einen Fehler verursacht.

INITIAL-VALUE = <text 0..1800 with-low *expr*>
Weist einer neuen einfachen Variablen einen Anfangswert zu. Der Wert muss zum Datentyp der Variablen passen.
Der Operand kann auch als Ausdruck angegeben werden.
Bei bereits bestehenden einfachen Variablen wird die Angabe ignoriert.
Zusammengesetzte Variablen können nicht als Ganzes initialisiert werden.
Elemente von Strukturlayouts dürfen nicht initialisiert werden.

TYPE =
Bestimmt den Datentyp der Variablen.

TYPE = *ANY
Der Variablen können später beliebig wechselnde Werte der Datentypen STRING, INTEGER und BOOLEAN zugewiesen werden.

TYPE = *STRING
Weist der Variablen den Datentyp STRING zu.
Wertebereich: beliebige Zeichenkette.
Länge: 0 bis 4096 Bytes (Ausnahme: wenn die Variable mit einer Jobvariablen verknüpft ist, darf sie maximal 256 Bytes lang sein.)

TYPE = *INTEGER
Weist der Variablen den Datentyp INTEGER zu.
Wertebereich: Ganzzahl zwischen -2^{31} und $+2^{31}-1$

TYPE = *BOOLEAN
Weist der Variablen den Datentyp BOOLEAN zu.
Wertebereich: TRUE, FALSE, ON, OFF, YES, NO

TYPE = *STRUCTURE(...)
Legt fest, dass das Strukturelement eine zusammengesetzte Variable vom Typ Struktur ist.

DEFINITION = *DYNAMIC
Dynamisch erweiterbare Struktur.

DEFINITION = *BY-SYSCMD
Statische Struktur, deren Elemente anschließend durch Kommandos im SYSCMD-Strom deklariert werden.

DEFINITION = <structured-name 1..20>
Name des Strukturlayouts, über das die statische Struktur definiert wird.

MULTIPLE-ELEMENTS = *NO
Bestimmt, dass das Strukturelement kein Array und keine Liste ist.

MULTIPLE-ELEMENTS = *ARRAY(...)
Deklariert ein Array, das heißt: das Strukturelement wird als zusammengesetzte Variable vom Typ Array deklariert.
Ein Array kann nicht als Ganzes initialisiert werden.

LOWER-BOUND = 0 / <integer -2147483648..2147483647>
Untere Grenze für den Arrayindex

LOWER-BOUND = *NONE
Eine untere Grenze für den Arrayindex wird nicht definiert.

UPPER-BOUND = *NONE
Eine obere Grenze für den Arrayindex wird nicht definiert.

UPPER-BOUND = <integer -2147483648..2147483647>
Obere Grenze für den Arrayindex

MULTIPLE-ELEMENTS = *LIST(...)
Deklariert eine Liste, das heißt: das Strukturelement wird als zusammengesetzte Variable vom Typ Liste deklariert.

LIMIT = *NONE
Die Anzahl der Listenelemente wird nicht begrenzt.

LIMIT = <integer 1..2147483647>
Definiert die maximale Anzahl der Listenelemente.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; Element schon deklariert
2	0	CMD0001	Warnung; Operand INITIAL-VALUE wurde ignoriert
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler garantierte Meldung: SDP1018
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1

```
/BEGIN-STRUCTURE NAME = BANKVERBINDUNG(SCOPE = *TASK)
/DECLARE-ELEMENT BANKLEITZAHL(TYPE = *INTEGER)
/DECLARE-ELEMENT KONTEN(TYPE = *INTEGER),-
/MULTIPLE-ELEMENTS = *ARRAY(LOWER-BOUND = 1, UPPER-BOUND =3)
/END-STRUCTURE
/DECLARE-VARIABLE PERSON(TYPE = *STRUCTURE(DEFINITION = *BY-SYSCMD))
/BEGIN-STRUCTURE

/DECLARE-ELEMENT NAME(TYPE = *STRING)
/DECLARE-ELEMENT VORNAME(TYPE = *STRING)
/DECLARE-ELEMENT BANKVERBINDUNG(TYPE = *STRUCTURE-
/(DEFINITION = BANKVERBINDUNG))
/END-STRUCTURE
```

Erzeugt die prozedurlokale Struktur PERSON:

```
PERSON.NAME
PERSON.VORNAME
PERSON.BANKVERBINDUNG.BANKLEITZAHL
PERSON.BANKVERBINDUNG.KONTEN
```

Die Variablen PERSON.NAME und PERSON.VORNAME wurden mit TYPE = *STRING deklariert, ihnen dürfen also nur Strings zugewiesen werden.
Die Variable PERSON.BANKVERBINDUNG.BANKLEITZAHL wurde mit TYPE = *INTEGER deklariert, ihr dürfen nur Integer-Werte zugewiesen werden.
PERSON.BANKVERBINDUNG.KONTEN ist ein Array mit drei Elementen, dem nur INTEGER-Werte zugewiesen werden dürfen. Die Elemente dieses Arrays werden noch nicht bei der Variablendeklaration erzeugt, sondern erst bei der ersten Zuweisung.

Zum Beispiel sind folgende voneinander unabhängige Zuweisungen möglich:

```
PERSON.BANKVERBINDUNG.BANKLEITZAHL = 70010080
PERSON.BANKVERBINDUNG.KONTEN#1 = 6001023
```

Beispiel 2

```
/DECLARE-VARIABLE VARIABLE-NAME = BAUM(TYPE = *STRUCTURE(*BY-SYSCMD)),-
/MULTIPLE-ELEMENTS = *ARRAY(LOWER-BOUND = 1, UPPER-BOUND = 10),-
/SCOPE = *TASK
/BEGIN-STRUCTURE
/DECLARE-ELEMENT AST,MULTIPLE-ELEMENTS = *ARRAY
/END-STRUCTURE
```

Danach sind folgende Zuweisungen möglich:

```
/BAUM#1.AST#1 =...; BAUM#1.AST#2 =...
/BAUM#2.AST#1 =...; BAUM#2.AST#2 =...
.........
/BAUM#10.AST#1 =....; BAUM#10.AST#2 =...
```

Beispiel 3

```
/BEGIN-STRUCTURE NAME = WOHNEINHEIT
/   DECLARE-ELEMENT ANZAHL-ZIMMER
/   DECLARE-ELEMENT ZIMMERGROESSE, MULTIPLE-ELEMENTS = *ARRAY
/   DECLARE-ELEMENT INHABER-NAME
/END-STRUCTURE
/DECLARE-VARIABLE HAUS(TYPE = *STRUCTURE(*BY-SYS)),-
/       MULTIPLE-ELEMENT = *ARRAY
/BEGIN-STRUCTURE
/DECLARE-ELEMENT WOHNUNG(TYPE = *STRUCTURE(DEF = WOHNEINHEIT)),-
/       MULTIPLE-ELEMENTS = *ARRAY
/DECLARE-ELEMENT HAUSBESITZER(TYPE = *STRUCTURE(DEF = WOHNEINHEIT)),-
/       MULTIPLE-ELEMENTS = *ARRAY
/DECLARE-ELEMENT ADRESSE(TYPE = *STRUCTURE(DEF = WOHNEINHEIT)),-
/       MULTIPLE-ELEMENTS = *ARRAY
/END-STRUCTURE
```

In Zuweisungen werden die Variablen folgendermaßen angesprochen:

```
HAUS#1.WOHNUNG#1.ANZAHL-ZIMMER =
HAUS#1.WOHNUNG#2.ANZAHL-ZIMMER =
HAUS#1.WOHNUNG#3.ANZAHL-ZIMMER =
...
HAUS#5.WOHNUNG#8.ANZAHL-ZIMMER =
...
```

Beispiel 4

Die Deklaration eines Strukturlayouts beginnt mit dem Kommando BEGIN-STRUCTURE und endet mit dem Kommando END-STRUCTURE.

```
/BEGIN-STRUCTURE NAME = AA
/DECLARE-ELEMENT Z
/END-STRUCTURE
/BEGIN-STRUCTURE NAME = BB
/DECLARE-ELEMENT X
/DECLARE-ELEMENT Y
/END-STRUCTURE
/DECLARE-VARIABLE V
/DECLARE-VARIABLE W
/...
/IF (V = W)
/DECLARE-VARIABLE A(TYPE = *STRUCTURE(DEF = AA))
/ELSE
/DECLARE-VARIABLE B(TYPE = *STRUCTURE(DEF = BB))
/END-IF
```

Wenn V = W gilt, wird die Struktur A deklariert (bestehend aus der Variablen A.Z), andernfalls wird die Struktur B deklariert (bestehend aus den Variablen B.X, B.Y). Zwischen BEGIN-STRUCTURE und END-STRUCTURE dürfen Kontrollflusskommandos und Prozeduraufrufe mit INCLUDE-PROCEDURE stehen. In der aufgerufenen Include-Prozedur können jedoch keine Elemente der aufrufenden Prozedur deklariert werden. Die Elemente einer Struktur müssen in der gleichen Prozedur wie ihr einleitendes BEGIN-STRUCTURE und abschließendes END-STRUCTURE deklariert werden. Eine Bezugnahme in der Include-Prozedur auf eine in dieser Weise unvollständig deklarierte Struktur führt zum Fehler.

Beispiel 5

```
/DECLARE-VARIABLE DYN-STRUC(TYPE=*STRUCTURE(DEFINITION=*DYNAMIC))
/DECLARE-ELEMENT DYN-STRUC.SUB.NUMBER(TYPE=*INTEGER)
/DECLARE-ELEMENT DYN-STRUC.LIST,MULTIPLE-ELEMENTS=*LIST
/DYN-STRUC.SUB.STRING='DYNAMISCH ERZEUGTES ELEMENT MIT DATENTYP *ANY'
/DYN-STRUC.SUB.NUMBER=1234
/DYN-STRUC.LIST#1=1
/DYN-STRUC.LIST#2=2
```

Hinweise

– Bei der Deklaration von Elementen für ein Strukturlayout definiert die Operandeneinstellung NAME=list-poss(2000): <structured-name 1..20>(...) nur den Namen des Elements; der Name des Layouts ist damit nicht gegeben.
– Bei der Deklaration von Elementen für eine Variable, die als dynamische Struktur definiert wurde, definiert die Operandeneinstellung NAME=list-poss(2000): <structured-name 1..255>(...) den kompletten Namen des Elements, inklusive dem Name der gesamten Variablen

DECLARE-PARAMETER
Prozedurparameter deklarieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando DECLARE-PARAMETER werden die Prozedurparameter deklariert, die während des Prozedurablaufs eine (konkreten) Wert benötigen. Des Weiteren wird die Art und Weise der Übergabe der Parameterwerte an die Prozedur vereinbart (Anfangswert, Prompting, ...). Die Deklaration von Prozedurparametern ist nur im Prozedurkopf erlaubt.

Prozedurparameter sind in SDF-P prozedurlokale Variablen: Bei der Definition im Prozedurkopf erhalten sie implizit den Geltungsbereich SCOPE = *CURRENT.

Die Namen der Prozedurparameter sind gleichzeitig die Schlüsselwörter der Prozedurparameter im Operanden PROCEDURE-PARAMETERS der Kommandos CALL-, ENTER- und INCLUDE-PROCEDURE.

Format

DECLARE-**PAR**AMETER

NAME = list-poss(2000): <structured-name 1..20>(...)

<structured-name 1..20>(...)

INITIAL-**VAL**UE = ***NONE** / ***PROMPT**(...) / <text 0..1800 with-low *expr*>

***PROMPT**(...)

PROMPT-STRING = ***STD** / <text 0..1800 with-low *string-expr*>

,DEFAULT-**VAL**UE = ***NONE** / <text 0..1800 with-low *expr*>

,SECRET-INPUT = ***NO** / ***Y**ES

,**TYPE** = ***ANY** / ***STRING** / ***INTEGER** / ***BOOL**EAN

,**TRANS**FER-**TYPE** = ***BY-VAL**UE / ***BY-REF**ERENCE

Operandenbeschreibung

NAME = list-poss (2000): <structured-name 1..20>(...)
Bestimmt den Namen des Prozedurparameters.

INITIAL-VALUE =
Bestimmt den Anfangswert des Prozedurparameters.

INITIAL-VALUE = *NONE
Der Prozedurparameter wird nicht initialisiert; es wird kein Anfangswert vereinbart. Dem Prozedurparameter muss beim Prozeduraufruf ein Wert zugewiesen werden (siehe „Prozedurparameter übergeben“ auf Seite 108).

INITIAL-VALUE = *PROMPT(...)
Falls der Prozedurparameter beim ersten lesenden Zugriff noch keinen Wert enthält, wird der Wert im Dialog abgefragt. Ist das der Fall, wird der Wert stets in Großbuchstaben konvertiert. Wenn der Wert in Hochkommata eingeschlossen ist, werden diese entfernt. Ist keine Dialogabfrage möglich, wird die Fehlermeldung SDP0219 ausgegeben.

PROMPT-STRING =
Definiert eine Zeichenfolge, die als Prompt-String (Eingabeaufforderung) ausgegeben wird. Der Prompt-String wird durch den bei DEFAULT-VALUE = ... angegebenen Text ergänzt. Die Eingabeaufforderung endet immer mit einem Doppelpunkt.Als Eingabeaufforderung erscheint dann:
<prompt-string>␣(DEFAULT = <default-value>)␣:

PROMPT-STRING = *STD
Als Voreinstellung wird der bei NAME=... angegebene Parametername (Variablenname) ausgegeben.

PROMPT-STRING = <text 0..1800 with-low ***string-expr***>
Definiert die Zeichenfolge, die als Prompt-String ausgegeben wird.

DEFAULT-VALUE =
Definiert einen Anfangswert für den Fall, dass im Dialog keine Eingabe (d.h. nur DUE) erfolgt oder die Prozedur im Hintergrund abläuft. Der Wert wird (zur Information) als Teil der Eingabeaufforderung mit ausgegeben.

DEFAULT-VALUE = *NONE
Es wird keine (Default-)Zeichenfolge vereinbart.

DEFAULT-VALUE = <text 0..1800 with-low *expr*>
Ausdruck, der als Default für den Anfangswert benutzt wird. Der angegebene Ausdruck muss zum Typ des Parameters passen.

SECRET-INPUT = *NO / *YES
Es kann vereinbart werden, ob die Eingabe im Dialog geschützt über ein dunkelgesteuertes Feld erfolgt; die Eingabe wird in diesem Fall auch nicht protokolliert.

INITIAL-VALUE = <text 0..1800 with-low *expr*>
Bestimmt einen Anfangswert. Der angegebene Ausdruck muss zum Datentyp des Prozedurparameters passen (Hochkommas bei String-Literalen!). Der Anfangswert gilt, wenn beim Aufruf kein anderer Wert übergeben wird.

TYPE =
Bestimmt den Datentyp des Prozedurparameters.

TYPE = *ANY
Bestimmt, dass dem Prozedurparameter beliebig wechselnde Werte vom Typ STRING, INTEGER oder BOOLEAN zugewiesen werden können.

TYPE = *STRING
Weist dem Prozedurparameter den Datentyp STRING zu.
Wertebereich: beliebige Zeichenkette

TYPE = *INTEGER
Weist dem Prozedurparameter den Datentyp INTEGER zu.
Wertebereich: Ganzzahl zwischen -2^{31} und $+2^{31}-1$

TYPE = *BOOLEAN
Weist dem Prozedurparameter den Datentyp BOOLEAN zu.
Wertebereich: TRUE, FALSE, YES, NO, ON, OFF

TRANSFER-TYPE =
Vereinbart, ob die übergebene Zeichenkette als Wert oder als Variablenname interpretiert werden soll.

TRANSFER-TYPE = *BY-VALUE
Die angegebene Zeichenkette ist ein Wert.
Es wird eine prozedurlokale Variable angelegt, die diesen Wert übernimmt. An die aufrufende Prozedur wird nichts zurückgegeben. Der Prozedurparameter dient lediglich als Eingabeparameter.
Dies gleicht dem Übergabemechanismus in Nicht-S-Prozeduren. Der Wert des übergebenen Argumentes überschreibt den Anfangswert aus dem DECLARE-PARAMETER-Kommando. Wird dem Prozedurparameter kein Wert übergeben, gilt der Anfangswert aus dem DECLARE-PARAMETER-Kommando. Falls INITIAL-VALUE = *NONE definiert ist, muss für diesen Prozedurparameter ein Wert übergeben werden.

Der übergebene String muss in den Typ des formalen Prozedurparameters konvertierbar sein. Ein formaler Prozedurparameter vom TYPE = *ANY erhält dann immer den aktuellen Typ STRING.

Da die Prozedurparameter Variablen im Sinne von SDF-P sind, können ihre Werte während des Prozedurablaufes geändert werden.

TRANSFER-TYPE = *BY-REFERENCE
Die angegebene Zeichenkette ist der Name einer Variablen, die den Wert des Prozedurparameters enthält. Jeder Zugriff auf den Prozedurparameter in der aufgerufenen Prozedur ist ein Zugriff auf diese Variable in der rufenden Prozedur.

Kommando-Returncode

Wenn DECLARE-PARAMETER im Prozedurkopf einer S-Prozedur benutzt wird, wird es während der Prozeduranalyse vollständig ausgewertet. Jeder Fehler ist also ein Fehler während der Prozedurvorbereitung; die Prozedur wurde zu diesem Zeitpunkt noch nicht ausgeführt. (Siehe dazu den Kommando-Returncode von CALL-PROCEDURE oder INCLUDE-PROCEDURE.) Nur wenn DECLARE-PARAMETER in einem anderen (d.h. falschen) Kontext benutzt wird, erscheinen die folgenden Returncodes.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1: Veranschaulichung des Prompting

In der Prozedur PROC.PROMPT wird ein Prozedurparameter deklariert, der erst bei Aufruf der Prozedur im Dialog abgefragt bzw. „gepromptet“ wird. Es handelt sich dabei um eine deutsche Farbenbezeichnung, die durch die Prozedur ins Englische übersetzt wird.

```
/SET-PROCEDURE-OPTIONS
/DECLARE-PARAMETER NAME(INITIAL-VALUE=*PROMPT-
/    (PROMPT-STRING='NAME DER ZU UEBERSETZENDEN FARBE EINGEBEN',-
/    DEFAULT-VALUE='ROT')
/COLOR=TRANSLATE(STRING=NAME-,
/,WHEN1='ROT',    THEN1='RED'-,
/,WHEN2='GRUEN',  THEN2='GREEN'-,
/,WHEN3='BLAU',   THEN3='BLUE'-,
/,WHEN4='GELB',   THEN4='YELLOW'-,
/,WHEN5='SCHWARZ',THEN5='BLACK'-,
/,WHEN6='WEISS',  THEN6='WHITE'-,
/,ELSE='UNKNOWN')
/SHOW-VAR NAME
/SHOW-VAR COLOR
```

Nach dem Aufruf der Prozedur erfolgt mit dem ersten Auftreten des zu „promptenden“ Prozedurparameters NAME die Eingabeaufforderung für die zu übersetzende Farbe. Gibt man diese ein, wird sie im Anschluss daran übersetzt.

```
(IN) /CALL-PROC PROC.PROMPT
(OUT) %BITTE NAME DER ZU UEBERSETZENDEN FARBE EINGEBEN (DEFAULT = ROT):
(IN) ROT
(OUT) NAME = ROT
(OUT) COLOR = RED
```

Beispiel 2: Wirkung von TRANSFER-TYPE

In der Prozedur P wird der Prozedurparameter PAR1 deklariert mit TRANSFER-TYPE = *BY-VALUE.

```
/DECLARE-PARAMETER PAR1(TYPE = *STRING,TRANSFER-TYPE = *BY-VALUE)
```

Die Art der Parameterübergabe an eine S-Prozedur beim Prozeduraufruf unterscheidet sich dann nicht von der Art der Parameterübergabe bei Nicht-S-Prozeduren.

Die Prozedur könnte folgendermaßen aufgerufen werden:

```
/CALL-PROCEDURE P, PROCEDURE-PARAMETER = (PAR1 = ABC)
/CALL-PROCEDURE P, PROCEDURE-PARAMETER = (PAR1 = 'ABC')
```

In beiden Fällen wird der String 'ABC' übergeben. Zu beachten ist, dass innerhalb von P auf PAR1 auch schreibend zugegriffen werden kann; dies hat aber auf die Umgebung des Aufrufers keine Wirkung. Soll an PAR1 der Wert einer Variablen X übergeben werden, muss Ausdrucksersetzung verwendet werden:

```
/CALL-PROCEDURE P, PROCEDURE-PARAMETER = (PAR1 = &X)
```

&X bedeutet, dass der Wert der Variablen X übergeben wird. Veränderungen von PAR1 haben für die rufende Prozedur keine Wirkung.

Der Prozedurparameter PAR2 wird in der Prozedur P deklariert mit `TRANSFER-TYPE = *BY-REFERENCE`

```
/DECLARE-PARAMETER PAR2(TYPE = *STRING, TRANSFER-TYPE = *BY-REFERENCE)
```

Die Prozedur P könnte folgendermaßen aufgerufen werden:

```
/CALL-PROCEDURE P, PROCEDURE-PARAMETER = (PAR2 = ABC)
/CALL-PROCEDURE P, PROCEDURE-PARAMETER = (PAR2 = 'ABC')
```

In beiden Fällen gilt, dass jeder Zugriff auf die Variable PAR2 in Wirklichkeit ein Zugriff auf die Variable ABC in der rufenden Prozedur ist.

Beispiel 3

Im Dialog wird eine Variable ABC deklariert und ihr der Wert LEVEL0 zugewiesen. Anschließend wird die Prozedur P aufgerufen.

```
/ABC = 'LEVEL0'
/CALL-PROCEDURE P,(ABC)
```

In der Prozedur P ist der Prozedurparameter PAR3 deklariert:

```
/DECLARE-PARAMETER PAR3(TYPE = *STRING, TRANSFER-TYPE = *BY-REFERENCE)
```

Die Prozedur P enthält folgende Kommandofolge:

```
/ABC = PAR3
/PAR3 = 'LEVEL1'
/SHOW-VARIABLE ABC
/EXIT-PROCEDURE
```

Die Zuweisung ABC = PAR3 bewirkt, dass in der Prozedur P implizit eine Variable ABC angelegt wird, der der Inhalt des Prozedurparameters PAR3 zugewiesen wird, das heißt der Inhalt der Variablen ABC, die im Dialog initialisiert wurde ('LEVEL0').
Anschließend wird dem Prozedurparameter PAR3 der Wert 'LEVEL1' zugewiesen, und damit gleichzeitig der Variablen ABC der Dialogebene.
Mit dem Kommando SHOW-VARIABLE wird der Inhalt der Variablen ABC ausgegeben, die in der Prozedur P sichtbar ist, also der Variablen, die implizit bei der ersten Zuweisung in der Prozedur P angelegt wurde (Inhalt: LEVEL0).
Mit EXIT-PROCEDURE wird die Prozedur P beendet.
Im Dialog wird jetzt folgendes Kommando aufgerufen:

```
/SHOW-VARIABLE ABC
```

Dieses Kommando greift jetzt auf die im Dialog angelegte Variable ABC zu. Da der Inhalt dieser Variablen über den Prozedurparameter PAR3 in der Prozedur P beeinflusst wird, zeigt SHOW-VARIABLE als Variableninhalt LEVEL1 (in der Prozedur P wurde dem Prozedurparameter PAR3 dieser Wert zugewiesen).

Beispiel 4

```
PROC.1

/SET-PROCEDURE-OPTIONS
/DECLARE-VARIABLE GARTEN(TYPE = *STRUCTURE(DEFINITION=*DYNAMIC))
/GARTEN.STUHL = 4
/GARTEN.TISCH = 1
/GARTEN.MOEBEL = 0
/CALL-PROCEDURE PROC.2, (,&(GARTEN.TISCH),-
/&(GARTEN.STUHL), GARTEN.MOEBEL)
/SHOW-VARIABLE GARTEN.MOEBEL
GARTEN.MOEBEL = 5

PROC.2

/SET-PROCEDURE-OPTIONS
/BEGIN-PARAMETER-DECLARATION
/DECLARE-PARAMETER ART('WINTERGARTEN',TRANSFER-TYPE=*BY-VALUE)
/DECLARE-PARAMETER TISCH(*NONE,TYPE=*INTEGER,TRANSFER-TYPE=*BY-VALUE)
/DECLARE-PARAMETER STUEHLE(0,TYPE=*INTEGER,TRANSFER-TYPE=*BY-VALUE)
/DECLARE-PARAMETER GESAMT(0,TRANSFER-TYPE=*BY-REFERENCE)
/END-PARAMETER-DECLARATION
/GESAMT=(TISCH)+(STUEHLE)
/SHOW-VARIABLE ART
/SHOW-VARIABLE TISCH
/SHOW-VARIABLE STUEHLE
/SHOW-VARIABLE GESAMT
ART = WINTERGARTEN
TISCH = 1
STUEHLE = 4
GESAMT = 5
```

Der formale Prozedurparameter ART wird dem ersten aktuellen Prozedurparameter zugeordnet, der ein leerer Prozedurparameter ist. ART wird mit 'WINTERGARTEN' initialisiert. Der zweite aktuelle Prozedurparameter darf kein leerer Prozedurparameter sein. Dies wird mit INIT-VALUE = *NONE erzwungen. Der Wert von GARTEN.TISCH wird dem formalen Prozedurparameter TISCH übergeben.
STUEHLE erhält den Wert von GARTEN.STUHL. Der Anfangswert 0 wird mit 4 überschrieben.

DECLARE-VARIABLE
Variable deklarieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit DECLARE-VARIABLE werden die Attribute dieser Variablen und evtl. auch ein Anfangswert festgelegt.

Die Einbindung von Jobvariablen in SDF-P ist über den Operanden CONTAINER möglich.

Es ist möglich, in einer lokalen Prozedur eine sonst nur dort zugängliche S-Variable für eine durch CALL-PROCEDURE gerufene Prozedur zugreifbar zu machen. Dazu muss in der lokalen Prozedur /DECLARE-VARIABLE VARIABLE-NAME=...,SCOPE = *CURRENT oder SCOPE = PROCEDURE (IMPORT-ALLOWED=*YES) angegeben werden und in der gerufenen Prozedur /IMPORT-VARIABLE VARIABLE-NAME=...,FROM=*SCOPE(SCOPE= *CALLING-PROCEDURES).

Format

(Teil 1 von 2)

DECLARE-**VAR**IABLE Kurzname: **DCV**

VARIABLE-**NAME** = list-poss(2000): <structured-name 1..20>(...)

<structured-name 1..20>(...)

INITIAL-**VAL**UE = ***NONE** / <text 0..1800 with-low *expr*>

,**TYPE** = ***ANY** / ***STRING** / ***INTEGER** / ***BOOL**EAN / ***STRUC**TURE(...)

***STRUC**TURE(...)

DEFINITION = ***DYN**AMIC / ***BY-SYSCMD** / <structured-name 1..20>

,**MULT**IPLE-**ELEM**ENTS = ***NO** / ***ARRAY**(...) / ***LIST**(...)

***ARRAY**(...)

LOWER-**BOUND** = **0** / ***NONE** / <integer -2147483648..2147483647>

,**UP**PER-**BOUND** = ***NONE** / <integer -2147483648..2147483647>

***LIST**(...)

LIMIT = ***NONE** / <integer 1..2147483647>

Fortsetzung ➠

(Teil 2 von 2)

,**SCOPE** = ***CURR**ENT(...) / ***PROC**EDURE(...) / ***TASK**(...)

***CURR**ENT(...)

IMPORT-ALLOWED = ***NO** / ***YES**

***PROC**EDURE(...)

IMPORT-ALLOWED = ***NO** / ***YES**

***TASK**(...)

STATE = ***ANY** / ***NEW** / ***OLD**

,**CONTAINER** = ***STD** / <composed-name 1..64> **/ *VAR**IABLE(...) / ***JV**(...)

***VAR**IABLE(...)

VARIABLE-NAME = <structured-name 1..20>

,SCOPE = ***VISIBLE** / ***TASK**

***JV**(...)

JV-NAME = <filename 1..54>

,STATE = ***ANY** / ***NEW** / ***OLD**

Operandenbeschreibung

VARIABLE-NAME = list-poss (2000): <structured-name 1..20>(...)
Deklariert den Variablennamen, das heißt den Namen einer einfachen Variablen (die nicht Element einer zusammengesetzten Variablen ist) oder den Namen einer zusammengesetzten Variablen.

INITIAL-VALUE = *NONE
Die Variable wird nicht initialisiert.
Das bedeutet für eine neue Variable: sie erhält keinen Anfangswert; ein Lesezugriff würde einen Fehler verursachen.
Ist die Variable bereits vorhanden, bleibt ihr Inhalt unverändert, ihr wird kein neuer Anfangswert zugewiesen.

INITIAL-VALUE = <text 0..1800 with-low *expr*>
Weist einer neuen Variablen einen Anfangswert zu; der Wert muss zum Datentyp der Variablen passen, er kann auch als Ausdruck angegeben werden.
Bei bereits bestehenden Variablen wird die Angabe ignoriert; ihnen wird kein neuer Anfangswert zugewiesen.
Zusammengesetzte Variablen können nicht als Ganzes initialisiert werden, das heißt, es darf ihnen mit INITIAL-VALUE kein Anfangswert zugewiesen werden.

TYPE =
Weist der Variablen den Datentyp zu.

TYPE = *ANY
Der Variablen können beliebig wechselnde Werte der Datentypen STRING, INTEGER und BOOLEAN zugewiesen werden.

TYPE = *STRING
Weist der Variablen den Datentyp STRING zu.
Wertebereich: beliebige Zeichenkette

TYPE = *INTEGER
Weist der Variablen den Datentyp INTEGER zu.
Wertebereich: Ganzzahl zwischen -2^{31} und $2^{31}-1$

TYPE = *BOOLEAN
Weist der Variablen den Datentyp BOOLEAN zu.
Wertebereich: TRUE, FALSE, YES, NO, ON, OFF.

TYPE = *STRUCTURE(...)
Deklariert eine zusammengesetzte Variable vom Typ Struktur.

DEFINITION = *DYNAMIC
Dynamisch erweiterbare Struktur

DEFINITION = *BY-SYSCMD
Statische Struktur, deren Elemente anschließend durch ein Kommando im SYSCMD-Strom deklariert werden.

DEFINITION = <structured-name 1..20>
Name des Strukturlayouts.

MULTIPLE-ELEMENTS = *NO
Bestimmt, dass die Variable kein Array und keine Liste ist.

MULTIPLE-ELEMENTS = *ARRAY(...)
Deklariert eine zusammengesetzte Variable vom Typ Array.

LOWER-BOUND = 0 / <integer -2147483648..2147483647>
Untere Grenze für den Arrayindex

LOWER-BOUND = *NONE
Es wird keine untere Grenze für den Arrayindex festgelegt.

UPPER-BOUND = *NONE
Es wird keine obere Grenze für den Arrayindex festgelegt.

UPPER-BOUND = <integer -2147483648..2147483647>
Obere Grenze für den Arrayindex. Der angegebene Wert muss größer oder gleich dem Wert bei LOWER-BOUND sein.

MULTIPLE-ELEMENTS = *LIST(...)
Deklariert eine zusammengesetzte Variable vom Typ Liste.

LIMIT = *NONE
Die Anzahl der Listenelemente wird nicht begrenzt.

LIMIT = <integer 1..2147483647>
Maximale Anzahl der Listenelemente.

SCOPE =
Definiert den Geltungsbereich der Variablen.

SCOPE = *CURRENT(...)
Die Variable ist eine prozedurlokale Variable.
Entspricht in Call-Prozeduren der Angabe PROCEDURE.
In Include-Prozeduren bedeutet CURRENT, dass die Variable in der aktuellen Include-Prozedur angelegt wird. Sie ist dann in dieser Include-Prozedur und in allen tiefergeschachtelten Include-Prozeduren sichtbar (= Geltungsbereich Include).

IMPORT-ALLOWED =
Gibt an, ob die Variable mit IMPORT-VARIABLE in der gerufenen Prozedur importiert werden kann.

IMPORT-ALLOWED = *NO
Die Variable kann nicht mit IMPORT-VARIABLE in der gerufenen Prozedur importiert werden.

IMPORT-ALLOWED = *YES
Die Variable kann mit IMPORT-VARIABLE in der gerufenen Prozedur importiert werden.

SCOPE = *PROCEDURE(...)
Die Variable ist eine prozedurlokale Variable mit dem Geltungsbereich Prozedur.
Die Variable wird in der aktuellen Prozedur angelegt.
Bei Include-Prozeduren ist die aktuelle Prozedur jeweils die umgebende Call-Prozedur, von der aus die Include-Prozedur aufgerufen wurde.
Die Variable ist in dieser Prozedur sichtbar sowie in allen tiefergeschachtelten Include-Prozeduren.

IMPORT-ALLOWED =
Gibt an, ob die Variable mit IMPORT-VARIABLE in der gerufenen Prozedur importiert werden kann.

IMPORT-ALLOWED = *NO
Die Variable kann nicht mit IMPORT-VARIABLE in der gerufenen Prozedur importiert werden.

IMPORT-ALLOWED = *YES
Die Variable kann mit IMPORT-VARIABLE in der gerufenen Prozedur importiert werden.

SCOPE = *TASK(...)
Die Variable ist eine taskglobale Variable.
Wird sie in einer Include-Prozedur angelegt, so ist sie auch in der umgebenden Call-Prozedur, von der aus die Include-Prozedur aufgerufen wurde, und in allen tiefergeschachtelten Include-Prozeduren sichtbar.

STATE = *ANY
Existiert in der Task bereits eine Variable mit dem angegebenen Namen, so wird diese verwendet, andernfalls wird eine neue Variable deklariert.

STATE = *NEW
In der Task darf keine Variable mit dem angegebenen Namen existieren.

STATE = *OLD
In der Task muss eine Variable mit dem angegebenen Namen existieren. Die aktuelle Variablendeklaration muss dann mit der Deklaration der bereits vorhandenen Variablen übereinstimmen.

CONTAINER = *STD
Der Variablen wird kein Variablenbehälter zugeordnet. Der Wert der Variablen wird im Klasse-5-Speicher abgelegt.

CONTAINER = <composed-name 1..64>
Verbindet die aktuell deklarierte Variable mit dem hier angegebenen Variablenbehälter. Dieser Variablenbehälter muss geöffnet sein. Die Angabe von „STD“ ist hier nicht erlaubt, da „STD“ nicht als permanent-bestehender Variablenbehälter interpretiert wird.

CONTAINER = *VARIABLE(...)
Verbindet die aktuell deklarierte Variable über einen Link-Mechanismus mit einer anderen Variablen, die in dieser Prozedur bereits definiert ist. Diese Variable wird dann als Variablenbehälter bezeichnet.
Strukturelemente können nicht als Variablenbehälter angegeben werden.

VARIABLE-NAME = <structured-name 1..20>
Name einer Variablen, die in der Prozedur bereits definiert wurde. Die Attribute der Variablen, die als Variablenbehälter dient, und der aktuell deklarierten Variablen müssen miteinander verträglich sein.

SCOPE =
Geltungsbereich der Behälter-Variablen

SCOPE = *VISIBLE
Die Variable ist sichtbar.

SCOPE = *TASK
Task-Variable

CONTAINER = *JV(...)
Definiert eine Jobvariable als Variablenbehälter: Die aktuell deklarierte Variable wird mit einer Jobvariablen verbunden, d. h., der Wert der Variablen wird in der Jobvariablen abgelegt.
Variablenbehälter vom Typ JV können nur mit einfachen Variablen verbunden werden, die mit TYPE = *STRING deklariert wurden, wobei der String höchstens 256 Bytes lang sein darf.
Zusammengesetzte Variablen können nicht mit Variablenbehältern vom Typ JV verbunden werden.
Wird eine taskglobale Variable (SCOPE = *TASK) mit einer Jobvariablen verbunden, so müssen die STATE-Angaben in TASK- und JV-Operanden sinnvoll zusammenpassen. Dabei muss verhindert werden, dass die Neudeklaration einer bereits vorhandenen taskglobalen Variablen eine Jobvariable neu erzeugt.

Jobvariablen sind Bestandteil des kostenpflichtigen Softwareprodukts „Jobvariablen“. Sie sind nur dann verfügbar, wenn das Subsystem JV geladen ist. Nähere Informationen zu Jobvariablen enthält das Handbuch „Jobvariablen“ [5].

JV-NAME = <filename 1..54>
Name der Jobvariablen

STATE = *ANY
Existiert bereits eine Jobvariable dieses Namens, so wird diese Jobvariable verwendet, andernfalls wird eine neue Jobvariable angelegt.

STATE = *OLD
Die Jobvariable muss bereits existieren.

STATE = *NEW
Es wird eine neue Jobvariable angelegt, die Jobvariable darf noch nicht existieren.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; Element schon deklariert
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler garantierte Meldungen: SDP1018, SDP1030
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1

```
/DECLARE-VARIABLE A, SCOPE = *TASK
```

Die Variable A wird als taskglobale Variable mit TYPE = *ANY deklariert.

Beispiel 2

```
/DECLARE-VARIABLE DATA(C'ANTON',*ANY)
```

Die prozedurlokale Variable DATA mit Typ *ANY wird mit dem String 'ANTON' initialisiert.

Beispiel 3

```
/DECLARE-VARIABLE LOGO(TRUE, *BOOLEAN)
```

Die lokale boolesche Variable LOGO wird mit dem booleschen Wert TRUE belegt.

Beispiel 4

```
/DECLARE-VARIABLE LEN, SCOPE = *TASK
```

Die Variable LEN wird deklariert.

Beispiel 5

```
/DECLARE-VARIABLE ANF, TYPE = *STRING, SCOPE = *TASK, CONTAINER = *JV-
/(ANFANG.DAT)
```

Die Jobvariable ANFANG.DAT wird gesucht. Wird sie nicht gefunden, so wird eine Jobvariable dieses Namens katalogisiert. Der Wert von ANF wird immer in der Jobvariablen ANFANG.DAT abgelegt.

Beispiel 6

Auf der Kennung wird eine taskglobale Variable mit Namen A deklariert:

```
/DECLARE-VARIABLE A, SCOPE = *TASK
```

Es wird eine Prozedur erstellt, die eine Variable mit Namen A als SCOPE = *PROCEDURE deklariert:

```
/SET-PROCEDURE-OPTIONS
/DECLARE-VARIABLE A, SCOPE = *PROCEDURE
...
/A = FILE1
/DELETE-FILE &A
...
....
/CREATE-FILE &A, ...
/...
```

Bei jeder Verwendung von &A innerhalb der Prozedur wird auf den Inhalt der lokalen Variable zugegriffen.

Beispiel 7

Auf der Kennung existiert eine taskglobale Variable VAR-A:

```
/DECLARE-VARIABLE VAR-A, SCOPE = *TASK
/VAR-A = 'TASK VARIABLE DER BENUTZERKENNUNG'
```

Die Prozedur PROZEDUR.1 wird aufgerufen:

```
PROZEDUR.1
```

```
/SET-PROCEDURE-OPTIONS
/DECLARE-VARIABLE VAR-A, SCOPE = *PROCEDURE
/VAR-A = 'LOKALE VARIABLE AUS PROZEDUR 1'
/CALL-PROCEDURE PROZEDUR.2
/SHOW-VARIABLE VAR-A
```

Ausgegeben wird: 'LOKALE VARIABLE AUS PROZEDUR 1'; VAR-A ist in dieser Prozedur als lokale Variable deklariert worden.

```
PROZEDUR.2
```

```
/SET-PROCEDURE-OPTIONS
/DECLARE-VARIABLE VAR-A, SCOPE = *TASK
/SHOW-VARIABLE VAR-A
```

Ausgegeben wird: 'TASK VARIABLE DER BENUTZERKENNUNG', da auf die globale Variable zugegriffen wird.

Beispiel 8

Beispiel für eine erweiterbare Struktur, auch wenn implizite Deklarationen verboten sind:

```
/DECLARE-VARIABLE A(TYPE = *STRUCTURE(*DYNAMIC))
/SET-VARIABLE A.B = 7
/SET-VARIABLE A.X = TRUE
/SET-VARIABLE A#1 = 0       "Fehler Array-Elementname"
```

Die Struktur enthält jetzt die Elemente A.B und A.X.

Beispiel 9

Elemente von dynamischen Strukturen können auch explizit deklariert werden:

```
/DECLARE-VARIABLE A(TYPE = *STRUCTURE(*DYNAMIC))
/DECLARE-ELEMENT A.B(7, *INTEGER)
/DECLARE-ELEMENT A.X(TRUE, *BOOLEAN)
/SET-VARIABLE A#1 = 0       "Fehler Array-Elementname"
```

Die Struktur enthält jetzt die Elemente A.B und A.X.

Beispiel 10

Bevor eine Variable bei der Deklaration mit einem Variablenbehälter verbunden wird, müssen die Strukturlayouts deklariert sein. Die Strukturlayouts müssen kompatibel sein (in Relation zum SET-VARIABLE-Kommando) mit der Struktur der Variablen, die in den Variablenbehälter gesichert wurde.

```
/BEGIN-STRUCTURE MYSTRUCT
/  DECLARE-ELEMENT ELEM1
/  DECLARE-ELEMENT ELEM2
/END-STRUCTURE
/OPEN-VARIABLE-CONTAINER MYCONT,FROM-FILE=*LIB-ELEM(MY-LIBRARY), -
/                                        AUTOMATIC-DECLARE=*NONE
/DECLARE-VARIABLE MYVAR(TYPE=*STRUCTURE(MYSTRUCT)), CONTAINER=MYCONT
/ "MYVAR WIRD IM VARIABLENBEHAELTER ERZEUGT"
/MYVAR.ELEM1 = 'FIRST VALUE'
/MYVAR.ELEM2 = 'SECOND VALUE'
/SAVE-VARIABLE-CONTAINER MYCONT
/
/"NUN WIRD DAS LAYOUT MYSTRUCT GEAENDERT(EIN ELEMENT WIRD UNTERDRUECKT,"
/"EIN ANDERES NEU ANGEFUEGT)"

/BEGIN-STRUCTURE MYSTRUCT
/  DECLARE-ELEMENT ELEM2
/  DECLARE-ELEMENT NEU-ELEM
/END-STRUCTURE
/OPEN-VARIABLE-CONTAINER MYCONT,FROM-FILE=*LIB-ELEM(MY-LIBRARY), -
```

```
/                                          AUTOMATIC-DECLARE=*NONE
/DECLARE-VARIABLE MYVAR(TYPE=*STRUCTURE(MYSTRUCT)), CONTAINER=MYCONT
/ "MYVAR WIRD VOM VARIABLENBEHAELTER GEHOLT"
/SHOW-VARIABLE MYVAR

MYVAR.ELEM2 = SECOND VALUE
```

Es wird nur MYVAR.ELEM2 ausgegeben, da MYVAR.ELEM1 unterdrückt wurde und MYVAR.NEU-ELEM bis jetzt nicht initialisiert wurde.

Weiteres Beispiel siehe IMPORT-VARIABLE, Seite 687.

DELETE-STREAM
S-Variablenstrom löschen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando DELETE-STREAM löscht S-Variablenströme. Ihre Zuweisungen werden nicht mehr durch das Kommando SHOW-STREAM-ASSIGNMENT angezeigt.

Es werden nur Ströme gelöscht, die auf der obersten Prozedurebene (im Dialog-Modus) erstellt wurden. Weitere Übertragungen über den gelöschten Strom werden zurückgewiesen.

Variablenströme in Prozeduren werden implizit gelöscht, wenn die Prozedur verlassen wird und im Kommando SET-PROCEDURE-OPTIONS nicht SYSTEM-FILE-CONTEXT= *SAME-AS-CALLER eingestellt ist.

Variablenströme mit reservierten Namen (SYSINF, SYMSG, ...) können nicht gelöscht werden.

Format

DELETE-**STREAM**
STREAM-NAME = <composed-name 1..20 with-wild(40)> / list-poss(100): <structured-name 1..20>

Operandenbeschreibung

STREAM-NAME =
Name des S-Variablenstroms, der gelöscht werden soll.

STREAM-NAME = <structured-name 1..20 with-wild(40)>
Alle S-Variablenströme, die dieses Suchmuster erfüllen, werden mit *DUMMY belegt und unterdrückt.

STREAM-NAME = list-poss(100): <structured-name 1..20>
Liste von S-Variablenstrom-Namen, die mit *DUMMY belegt und unterdrückt werden sollen.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
2	0	SDP0516	Angegebener Variablenstrom-Name existiert nicht oder das Suchmuster wurde nicht gefunden; Prozess wird fortgesetzt
1	0	SDP0518	Kein Treffer bei Wildcard. Prozess wird fortgesetzt
2	0	SDP0535	Warnung vom Server bei der Löschung eines bestimmten S-Variablenstroms; Prozess wird fortgesetzt
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	64	SDP0515	Angegebener Variablenstrom nicht auf oberster Ebene erzeugt
	64	SDP0536	Server-Fehler; das Löschen des bestimmten S-Variablenstroms wurde abgewiesen.
	64	SDP0537	Interner Server-Fehler; das Löschen des bestimmten S-Variablenstroms wurde abgewiesen. Server-Verbindung abgebrochen nach unerwartetem Ereignis oder bei mangelhaften oder fehlenden System-Ressourcen

DELETE-VARIABLE
Variable löschen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

DELETE-VARIABLE löscht die Deklaration einer S-Variablen innerhalb des aktuellen Geltungsbereichs, d.h. auch Deklarationen von importierten Task-Variablen.

Der Name der S-Variablen kann danach nicht länger benutzt werden und der Wert ist gelöscht.

Es können einfache und zusammengesetzte Variablen gelöscht werden, aber nicht einzelne Elemente von zusammengesetzten Variablen.

Folgende Variablendeklarationen können mit DELETE-VARIABLE nicht gelöscht werden:

- Prozedurparameter
- Elemente von zusammengesetzten Variablen
- Systemvariablen (z.B. SYSSWITCH)
- Behälter-JVs
- nicht-permanente Behältervariablen
- Strukturlayouts

Format

DELETE**-VAR**IABLE
VARIABLE**-NAME** = <structured-name 1..20 with-wild(40)> / list-poss(2000): <structured-name 1..20>

Operandenbeschreibung

VARIABLE-NAME =
Name der S-Variablen, die gelöscht werden soll.

VARIABLE-NAME = <structured-name 1..20 with-wild(40)>
Alle S-Variablen, die dieses Suchmuster erfüllen, werden gelöscht.

VARIABLE-NAME = list-poss(2000):<structured-name 1..20>
Liste von S-Variablen, die gelöscht werden sollen.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; keine Aktion durchgeführt
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Hinweis

Der Fehler SDP1098 erscheint nicht bei Angaben von Variablennamen mit Wildcards.

Beispiel 1

Eingabe

```
/DECLARE-VARIABLE PROBE
/SET-VARIABLE PROBE=15
/SHOW-VARIABLE PROBE
```

Ausgabe

```
PROBE = 15
```

Eingabe

```
/DELETE-VARIABLE PROBE
/SHOW-VARIABLE PROBE
```

Ausgabe

```
SDP1008 VARIABLE/LAYOUT 'PROBE' EXISTIERT NICHT
SDP0234 OPERAND 'NAME' UNVOLLSTAENDIG
```

Beispiel 2

Eingabe

```
/DELETE-VARIABLE SYS*  "Es wird keine Fehlermeldung zurückgegeben"
/DELETE-VARIABLE SYSSWITCH
```

Ausgabe

```
SDP1098 AUF DIE VARIABLE 'SYSSWITCH' DARF KEIN DELETE-VARIABLE ANGEWENDET
WERDEN
```

ELSE
ELSE-Zweig im IF-BLock einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Im IF-Block leitet das ELSE-Kommando den letzten Zweig ein. Die Kommandos zwischen ELSE-Kommando und dem Abschlusskommando END-IF werden ausgeführt, wenn keine der zuvor in IF- oder ELSE-IF-Kommandos geprüften Bedingungen zutrifft (siehe Abschnitt „Verzweigungen definieren“ auf Seite 94).

Im IF-BLOCK-ERROR- und IF-CMD-ERROR-Block wird der ELSE-Zweig durchlaufen, wenn kein Fehler aufgetreten ist (siehe Abschnitt „Fehlerbehandlungsblöcke“ auf Seite 70).

Format

ELSE

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe IF-Kommando, Seite 681.

ELSE-IF
Im IF-Block einen Alternativzweig einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Der ELSE-IF-Zweig wird ausgeführt, wenn die im ELSE-IF-Kommando angegebene Bedingung zutrifft; andernfalls wird der nächste Zweig des IF-Blocks bzw. ein END-IF-Kommando gesucht.
Zum ELSE-IF-Zweig gehören alle Kommandos, die zwischen dem aktuellen ELSE-IF-Kommando und den nachfolgenden ELSE-IF-, ELSE- oder END-IF-Kommando stehen. (Ausführliche Informationen siehe Abschnitt „Verzweigungen definieren“ auf Seite 94.)

Format

ELSE-IF
CONDITION = <text 0..1800 with-low *bool-expr*>

Operandenbeschreibung

CONDITION = <text 0..1800 with-low *bool-expr*>
Logischer Ausdruck.
Legt die Bedingung fest, die erfüllt sein muss, damit die Kommandos des aktuellen ELSE-IF-Zweiges ausgeführt werden (zu „logischer Ausdruck“ siehe Kapitel „Ausdrücke“ auf Seite 255).

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe IF-Kommando, Seite 681.

END-BLOCK
Kommandoblock abschließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

END-BLOCK schließt einen BEGIN-Block ab, das heißt einen Kommandoblock, der mit dem Kommando BEGIN-BLOCK eingeleitet wurde.

Format

END-BLOCK
BLOCK = ***LAST** / <structured-name 1..255>

Operandenbeschreibung

BLOCK =
Bezeichnet den BEGIN-Block, der abgeschlossen werden soll.

BLOCK = *LAST
Verweis auf den zuletzt geöffneten BEGIN-Block.

BLOCK = <structured-name 1..255>
Verweis auf die Marke des zuletzt geöffneten BEGIN-Blocks; die Angabe einer anderen Blockmarke führt zu einer Fehlermeldung.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe BEGIN-BLOCK-Kommando, Seite 573.

END-FOR
FOR-Block abschließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

END-FOR schließt einen FOR-Block ab, d. h. eine FOR-Schleife, die mit dem FOR-Kommando eingeleitet wurde.

Bei Ausführung des Kommandos END-FOR wird der Schleifenvariablen im FOR-Kommando das nächste Element der Werteliste zugewiesen. Anschließend wird die Ausführung beim ersten Kommando nach dem FOR-Kommando fortgesetzt. Ist die Werteliste vollständig abgearbeitet, wird die Schleife beendet und der Prozedurlauf mit dem Kommando fortgesetzt, das dem END-FOR-Kommando folgt. (Ausführliche Informationen siehe Abschnitt „Schleifen definieren“ auf Seite 97.)

Format

END-FOR
BLOCK = ***LAST** / <structured-name 1..255>

Operandenbeschreibung

BLOCK =
Bezeichnet den FOR-Block, der abgeschlossen werden soll.

BLOCK = *LAST
Verweis auf den zuletzt geöffneten FOR-BLOCK

BLOCK = <structured-name 1..255>
Verweis auf die Marke des zuletzt geöffneten FOR-Blocks; die Angabe einer anderen Blockmarke führt zu einer Fehlermeldung.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0139	Back Branch-Grenze erreicht
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe FOR-Kommando, Seite 668.

END-IF
IF-Block abschließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

END-IF schließt Blöcke mit bedingten Kommandofolgen ab. Das sind:

– IF-Block
– IF-BLOCK-ERROR-Block
– IF-CMD-ERROR-Block

Der Prozedurlauf wird anschließend mit dem Kommando fortgesetzt, das auf END-IF folgt. (Nähere Informationen siehe Abschnitt „Verzweigungen definieren“ auf Seite 94.)

Format

END-IF
BLOCK = ***LAST** / <structured-name 1..255>

Operandenbeschreibung

BLOCK =
Bezeichnet den IF-, IF-BLOCK-ERROR- oder IF-CMD-ERROR-Block, der abgeschlossen werden soll.

BLOCK = *LAST
Verweis auf den zuletzt geöffneten IF-, IF-BLOCK-ERROR- oder IF-CMD-ERROR-Block

BLOCK = <structured-name 1..255>
Verweis auf die Marke des zuletzt geöffneten IF-, IF-BLOCK-ERROR- oder IF-CMD-ERROR-Blocks; die Angabe einer anderen Blockmarke führt zu einer Fehlermeldung.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe IF-Kommando, Seite 681.

END-PARAMETER-DECLARATION Prozedurparameterdeklaration abschließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

END-PARAMETER-DECLARATION beendet den Kommandoblock, der mit dem Kommando BEGIN-PARAMETER-DECLARATION eingeleitet wurde; in diesem Block werden die Prozedurparameter deklariert (siehe Abschnitt „Prozedurparameter deklarieren“ auf Seite 90).

Format

END-PARAMETER-**DECL**ARATION

Kommando-Returncode

Wenn END-PARAMETER-DECLARATION am Ende eines Prozedurkopfs einer S-Prozedur benutzt wird, wird es während der Vorbereitung der Prozedur vollständig ausgewertet und ausgeführt. Jeder Fehler während des Ablaufs des Kommandos ist also ein Fehler während der Prozedurvorbereitung; die Prozedur wurde zu diesem Zeitpunkt noch nicht ausgeführt. Wenn das Kommando korrekt ausgeführt wird, wird auch der Parametertransfer korrekt ausgeführt. Nur wenn END-PARAMETER-DECLARATION in einem anderen (d.h. falschen) Kontext benützt wird, erscheinen die folgenden Returncodes.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

END-STRUCTURE
Ende einer Strukturdeklaration kennzeichnen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

END-STRUCTURE schließt einen Strukturdeklarationsblock ab, der mit dem Kommando BEGIN-STRUCTURE eingeleitet wurde.

Format

END-STRUCTURE
NAME = ***LAST** / <structured-name 1..20>

Operandenbeschreibung

NAME =
Bezeichnet den Namen der Struktur, die abgeschlossen werden soll.

NAME = *LAST
Verweis auf den zuletzt mit BEGIN-STRUCTURE eingeleiteten Strukturdeklarationsblock

NAME = <structured-name 1..20>
Verweis auf den Namen des zuletzt geöffneten Strukturdeklarationsblocks

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
2	0	CMD0001	Warnung; Struktur ist leer
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommando DECLARE-ELEMENT, Seite 600.

END-WHILE
WHILE-Block abschließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

END-WHILE schließt einen WHILE-Block ab, d.h. eine Schleife, die mit dem WHILE-Kommando eingeleitet wurde.
Bei der Ausführung des END-WHILE-Kommandos wird die Schleifenbedingung im WHILE-Kommando überprüft. Falls sie erfüllt ist (TRUE), wird mit dem ersten Kommando des WHILE-Blocks der nächste Schleifendurchgang gestartet. Andernfalls wird die Schleife beendet und der Prozedurlauf mit dem ersten Kommando fortgesetzt, das auf END-WHILE folgt (nähere Information siehe „WHILE-Block“ auf Seite 99).

Format

END-WHILE
BLOCK = ***LAST** / <structured-name 1..255>

Operandenbeschreibung

BLOCK =
Bezeichnet den WHILE-Block, der abgeschlossen werden soll.

BLOCK = *LAST
Verweis auf den zuletzt geöffneten WHILE-Block

BLOCK = <structured-name 1..255>
Verweis auf die Marke des zuletzt geöffneten WHILE-Blocks; die Angabe einer anderen Blockmarke führt zu einer Fehlermeldung.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0139	Back Branch-Grenze erreicht
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe WHILE-Kommando, Seite 805.

ENTER-PROCEDURE
Prozedur im Hintergrund als Stapelauftrag starten

Anwendungsgebiet: **PROCEDURE, JOB**

Das Kommando ist Bestandteil von BS2000/OSD (Stand der Beschreibung: V7.0).
Der Aufrufer benötigt (im Gegensatz zu den SDF-P-Kommandos) die Privilegien STD-PROCESSING bzw. HARDWARE-MAINTENANCE.

Kommandobeschreibung

Mit dem Kommando ENTER-PROCEDURE kann der Benutzer eine Prozedur als Stapelauftrag starten. Im Gegensatz zu dem Kommando ENTER-JOB muss der Benutzer keine gesonderte Enter-Datei erzeugen. Die Prozedurparameter sind somit auch bei jedem asynchronen Prozedurablauf (Hintergrund-Prozedur) variabel. Enter-Dateien können nur mit dem Kommando ENTER-JOB gestartet werden.

Verfahren

1. Die Prozedurdatei wird als Kopie unter dem Namen `S.PROC.tsn.datum.uhrzeit` angelegt, wobei *datum* das Format yyyy-mm-dd und *uhrzeit* das Format hh.mm.ss besitzen.
2. Es wird eine Enter-Datei unter dem Namen `S.E.tsn.datum.uhrzeit` mit folgendem Inhalt angelegt:

```
/SET-LOGON-PARAMETERS
 .
 .
/CALL-PROCEDURE FROM-FILE=S.PROC.tsn.datum.uhrzeit, -
/               PROCEDURE-PARAMETERS=(parameter)
 .
 .
/EXIT-JOB SYSTEM-OUTPUT=option
```

   Der Wert von *parameter* entspricht der Angabe im Operanden PROCEDURE-PARAMETERS. Die Kopie der Prozedurdatei wird nach Prozedurablauf gelöscht. Der Wert von *option* entspricht der Angabe im Operanden SYSTEM-OUTPUT.
3. Die Enter-Datei wird mit ENTER-JOB gestartet. Die Angaben für die Operanden PROCESSING-ADMISSION, JOB-CLASS, JOB-NAME, MONJV, JV-PASSWORD, JOB-PRIORITY, RERUN-AFTER-CRASH, FLASH-AFTER-SHUTDOWN, SCHEDULING-TIME, START, REPEAT-JOB, LIMIT, RESOURCES, LISTING und JOB-PARAMETER werden entsprechend in das ENTER-JOB-Kommando übernommen.

Format

(Teil 1 von 2)

ENTER-PROCEDURE Kurzname: **ENP**

FROM-FILE = <filename 1..54 without-gen> / ***LIB**RARY-**ELEM**ENT(...)

***LIB**RARY-**ELEM**ENT(...)

LIBRARY = <filename 1..51 without-gen>
,**ELEM**ENT = <composed-name 1..38>

,**PROC**EDURE-**PAR**AMETERS = **_*NO_** / <text 0..1800 with-low>

,**PROC**ESSING-**ADMIS**SION = **_*SAME_** / ***PAR**AMETERS(...)

***PAR**AMETERS(...)

USER-IDENTIFICATION = **_*NONE_** / <name 1..8>
,**ACCOUNT** = **_*NONE_** / <alphanum-name 1..8>
,**PASS**WORD = **_*NONE_** / <c-string 1..8> / <c-string 9..32> / <x-string 1..16> / ***SECRET**

,**PROC**EDURE-**PASS**WORD = **_*NONE_** / <x-string 1..8> / <c-string 1..4> /
<integer -2147483648..2147483647> / ***SECRET**

,**CRYPTO-PASS**WORD = **_*NONE_** / <c-string 1..8> / <x-string 1..16> / ***SECRET**

,**HOST** = **_*STD_** / <c-string 1..8> / ***ANY**

,**JOB-CLASS** = **_*STD_** / <name 1..8>

,**JOB-NAME** = **_*NO_** / <name 1..8>

,**MONJV** = **_*NONE_** / <filename 1..54 without-gen-vers>

,**JV-PASS**WORD = **_*NONE_** / <c-string 1..4> / <x-string 1..8> / ***SECRET** /
<integer -2147483648..2147483647>

,**JOB-PRIO**RITY = **_*STD_** / <integer 1..9>

,**RERUN-AF**TER-**CRASH** = **_*NO_** / ***Y**ES

,**FLUSH-AF**TER-**SHUTDOWN** = **_*NO_** / ***Y**ES

,**SCHED**ULING-**TIME** = **_*STD_** / ***PAR**AMETERS(...) / ***BY-CALENDAR**(...)

***PAR**AMETERS(...)

START = **_*STD_** / ***SOON** / ***IMMED**IATELY / ***AT-STREAM**-STARTUP / ***WITHIN**(...) / ***AT**(...) /
***EARL**IEST(...) / ***LATEST**(...)

***WITHIN**(...)

HOURS = **_0_** / <integer 0..23 *hours*>
,**MINUT**ES = **_0_** / <integer 0..59 *minutes*>

***AT**(...)

DATE = **_*TODAY_** / <date>
,**TIME** = <time>

Fortsetzung ➠

(Teil 2 von 2)

```
        *EARLIEST(...)
            DATE = *TODAY / <date>
            ,TIME = <time>
        *LATEST(...)
            DATE = *TODAY / <date>
            ,TIME = <time>
    ,REPEAT-JOB = *STD / *NO / *DAILY / *WEEKLY / *AT-STREAM-STARTUP / *PERIOD(...)
        *PERIOD(...)
            HOURS = 0 / <integer 0..23 hours>
            ,MINUTES = 0 / <integer 0..59 minutes>
*BY-CALENDAR(...)
    CALENDAR-NAME = <filename 1..54 without-gen-vers>
    ,SYMBOLIC-DATE = <filename 1..20 without-cat-user-vers> /
                     <partial-filename 2..20 without-cat-user>
,LIMIT = *STD / <integer 1..32767> / *BY-DATE(...)
*BY-DATE(...)
    DATE = <date>
    ,TIME = <time>
,RESOURCES = *PARAMETERS (...)
*PARAMETERS(...)
    RUN-PRIORITY = *STD / <integer 30..255>
    ,CPU-LIMIT = *STD / *NO / <integer 1..32767 seconds>
    ,SYSLST-LIMIT = *STD / *NO / <integer 0..999999>
    ,SYSOPT-LIMIT = *STD / *NO / <integer 0..999999>
,LOGGING = *STD / *YES / *NO
,LISTING = *NO / *YES
,JOB-PARAMETER = *NO / <c-string 1..127>
,SYSTEM-OUTPUT = *STD / *PRINT / *DELETE
,ASSIGN-SYSTEM-FILES = *STD / *PARAMETERS(...)
*PARAMETERS(...)
    SYSLST = *STD / *PRIMARY / *DUMMY / <filename 1..54>
    ,SYSOUT = *STD / *PRIMARY / *DUMMY / <filename 1..54>
,PROTECTION = *NONE / *CANCEL
```

Operandenbeschreibung

FROM-FILE = *LIBRARY-ELEMENT(...) / <filename 1..54 without-gen>
Name der Datei oder des PLAM-Bibliothekelements, die die Prozedur enthalten.
Die Prozedur darf nicht mit dem Kommando SET-LOGON-PARAMETERS bzw. LOGON beginnen, also keine Enter-Datei sein.
Ist der Auftraggeber nicht Eigentümer der Datei (verschiedene Benutzerkennungen), muss die Datei zugreifbar sein (siehe Handbuch „Kommandos, Bd. 1-5“ [3], Operand PROTECTION= *PARAMETERS im Kommando CREATE-FILE bzw. MODIFY-FILE-ATTRIBUTES).
Der Auftraggeber muss in jedem Fall mindestens das Recht zum Ausführen besitzen, wenn die Datei mit Basic-ACL bzw. GUARDS geschützt ist.
Ist die Datei mit einem Kennwort gegen Ausführung geschützt, so muss das Kennwort im Operanden PROCEDURE-PASSWORD angegeben werden.

FROM-FILE = *LIBRARY-ELEMENT(...)
Die Prozedur ist in einer PLAM-Bibliothek abgelegt.

LIBRARY = <filename 1..51 without-gen>
Name der Bibliothek, die die Prozedur als Element enthält.

ELEMENT = <composed-name 1..38>
Name des Elements.
Die Summe der Längen des Bibliotheksnamens (ohne Katalog- und Benutzerkennung gerechnet) und des Elementnamens darf bei einstelliger Katalogkennung max. 39 Zeichen betragen. Bei mehrstelliger Katalogkennung verringert sich die Anzahl der Zeichen entsprechend.

PROCEDURE-PARAMETERS = *NO / <text 0..1800 with-low>
Parameterwerte, die an Stelle der entsprechenden symbolischen Parameter zu setzen sind. Die Parameterwerte müssen in runde Klammern eingeschlossen werden. Zu Prozedurparametern siehe auch Abschnitt „Prozedurparameter übergeben“ auf Seite 108ff..

PROCESSING-ADMISSION =
Gibt an, unter welcher Benutzerkennung der Stapelauftrag laufen soll.

PROCESSING-ADMISSION = *SAME
Der Stapelauftrag soll unter der eigenen Benutzerkennung laufen (d.h. unter der ENTER-PROCEDURE gegeben wurde).

PROCESSING-ADMISSION = *PARAMETERS(...)
Angaben zur LOGON-Berechtigung der Benutzerkennung.

USER-IDENTIFICATION = *NONE / <name 1..8>
Benutzerkennung, unter der der Stapelauftrag laufen soll.

ACCOUNT = *NONE / <alphanum-name 1..8>
Abrechnungsnummer der Benutzerkennung.

PASSWORD = *NONE / <c-string 1..8> / <c-string 9..32> / <x-string 1..16> / *SECRET
Kennwort der Benutzerkennung.
Die Eingabe eines „langen“ Kennworts (entspricht <c-string 9..32>) wird unterstützt. Ein Hash-Algorithmus konvertiert das „lange“ Kennwort in die interne 8 Byte lange Darstellung. Zur Vereinbarung „langer“ Kennwörter siehe Kommando MODIFY-USER-PROTECTION im Handbuch „Kommandos, Bd. 1-5“ [3].
Der Operand PASSWORD hat folgende Besonderheiten:
- Der eingegebene Wert wird nicht protokolliert.
- Im geführten Dialog ist das Eingabefeld automatisch dunkelgesteuert.
- Bei Angabe von *SECRET oder ^ stellt SDF im ungeführten Dialog und in Vordergrundprozeduren ein dunkelgesteuertes Eingabefeld zur verdeckten Eingabe des Kennwortes zur Verfügung.

PROCEDURE-PASSWORD = *NONE / <c-string 1..4> / <x-string 1..8> / <integer -2147483648..2147483647> / *SECRET
Kennwort, mit dem die Prozedurdatei gegen Ausführung geschützt ist.
Der Operand PROCEDURE-PASSWORD hat folgende Besonderheiten:
- Der eingegebene Wert wird nicht protokolliert.
- Im geführten Dialog ist das Eingabefeld automatisch dunkelgesteuert.
- Bei Angabe von *SECRET oder ^ stellt SDF im ungeführten Dialog und in Vordergrundprozeduren ein dunkelgesteuertes Eingabefeld zur verdeckten Eingabe des Kennwortes zur Verfügung.

CRYPTO-PASSWORD = *NONE / <c-string 1..8> / <x-string 1..16> / *SECRET
Kennwort, mit dem die Prozedurdatei verschlüsselt ist. Die Kopie der Prozedurdatei (S.PROC-Datei) wird mithilfe des Crypto-Kennworts entschlüsselt.
Der Operand CRYPTO-PASSWORD hat folgende Besonderheiten:
- Der eingegebene Wert wird nicht protokolliert.
- Im geführten Dialog ist das Eingabefeld automatisch dunkelgesteuert.
- Bei Angabe von *SECRET oder ^ stellt SDF im ungeführten Dialog und in Vordergrundprozeduren ein dunkelgesteuertes Eingabefeld zur verdeckten Eingabe des Kennwortes zur Verfügung.

HOST =
Legt den Rechner fest, auf dem der Auftrag ablaufen soll.
Die Operandenwerte ungleich *STD stehen nur dem Anwender mit dem Softwareprodukt HIPLEX MSCF zur Verfügung.

HOST = *STD
Der Auftrag soll auf dem lokalen Rechner ablaufen.

HOST = <c-string 1..8>
Host-Name des Rechners, auf dem der ENTER-Auftrag laufen soll.

HOST = *ANY
Die Angabe ist nur innerhalb eines XCS-Rechnerverbundes erlaubt. Näheres siehe Handbuch „HIPLEX MSCF“ [9]).

JOB-CLASS = *STD / <name 1..8>
Jobklasse, in die der Auftrag eingereiht werden soll. Die Berechtigung zu den verschiedenen Jobklassen kann mit dem Kommando SHOW-USER-ATTRIBUTES oder SHOW-JOB-CLASS abgefragt werden.

JOB-NAME = *NO / <name 1..8>
Name für den ENTER-Auftrag. Über diesen Namen kann der ENTER-Auftrag angesprochen werden (z.B. mit SHOW-JOB-STATUS). Er wird auch auf das Deckblatt des Druckerprotokolls gedruckt.
Default-Wert ist *NO, d.h. der ENTER-Auftrag erhält den Namen des kommandogebenden Auftrags.

MONJV = *NONE / <filename 1..54 without-gen-vers>
nur möglich, wenn das kostenpflichtige Subsystem JV geladen ist
Name der Jobvariablen (JV), die den Stapelauftrag überwachen soll. Über diese JV kann der Benutzer seinen Stapelauftrag ansprechen.
Während der Verarbeitung des Stapelauftrags setzt dann das System die JV auf entsprechende Werte:

$S Auftrag in Warteschlange
$R Auftrag läuft
$T Auftrag normal beendet
$A Auftrag abnormal beendet
$M Auftrag wurde mit MOVE-JOBS exportiert

JV-PASSWORD = *NONE / <c-string 1..4> / <x-string 1..8> / <integer -2147483648..2147483647> / *SECRET
Kennwort der JV.
Der Operandenwert JV-PASSWORD hat folgende Besonderheiten:

– Der eingegebene Wert wird nicht protokolliert.
– Im geführten Dialog ist das Eingabefeld automatisch dunkelgesteuert.
– Bei Angabe von *SECRET oder ^ stellt SDF im ungeführten Dialog und in Vordergrundprozeduren ein dunkelgesteuertes Eingabefeld zur verdeckten Eingabe des Kennwortes zur Verfügung.

JOB-PRIORITY = *STD / <integer 1..9>
Jobpriorität, die der Stapelauftrag erhalten soll.
Je niedriger der Wert, desto höher die Priorität. Der maximal zulässige Wert ist in der Jobklassendefinition festgelegt und kann mit dem Kommando SHOW-JOB-CLASS abgefragt werden.

JOB-PRIORITY = *STD
Es gilt die für die Jobklasse festgelegte Standardpriorität.

RERUN-AFTER-CRASH = *NO / *YES
Gibt an, ob der Stapelauftrag im nächsten Systemlauf neu zu starten ist, wenn die Bearbeitung auf Grund eines Systemfehlers oder Systemlaufende abgebrochen wurde.

Hinweis

Der Operand wird nicht ausgewertet, wenn im Operanden REPEAT Auftragswiederholung vereinbart wird.

FLUSH-AFTER-SHUTDOWN = *NO / *YES
Gibt an, ob der Stapelauftrag aus der Auftragswarteschlange zu entfernen ist, wenn er bis Systemlaufende nicht bearbeitet wurde.

Hinweis

Der Operand wird nicht ausgewertet, wenn im Operanden REPEAT Auftragswiederholung vereinbart wird.
Für Kalenderjobs wird die Angabe FLUSH-AFTER-SHUTDOWN=*YES zurückgewiesen.

SCHEDULING-TIME = *STD / *PARAMETERS(...) / *BY-CALENDAR(...)
Bestimmt die Art der Startzeitangabe für den Stapelauftrag.

SCHEDULING-TIME = *STD
Es gilt die Standardwerte der Startzeitangaben START und REPEAT-JOB für die gewählte Jobklasse (siehe Operanden in der Struktur SCHEDULING-TIME=*PARAMETERS(...)).
Wird der Stapelauftrag durch den Operator an der Bedienstation gestartet, werden die Angabe für START und REPEAT-JOB aus den gleichnamigen Operanden des SET-LOGON-PARAMETERS der Enter-Datei übernommen. Ist dort keine Angabe enthalten, wird der für die Jobklasse jeweils festgelegte Standardwert angenommen.
Tasks mit dem Privileg OPERATING können im Operanden DEFAULT-FROM-FILE diesen Default-Mechanismus einstellen.

SCHEDULING-TIME = *PARAMETERS(...)
Für den Stapelauftrag wird ein Startzeitpunkt festgelegt. Zusätzlich können Auftragswiederholungen vereinbart werden (Repeatjob).

START =
Startzeitpunkt des Stapelauftrags. Angaben abweichend von *STD werden nur ausgeführt, wenn sie gemäß Jobklassendefinition erlaubt sind (siehe Kommando SHOW-JOB-CLASS).

START = *STD
Es gilt der Standardwert für die gewählte Jobklasse.

START = *SOON
Der Auftrag soll unter Berücksichtigung seiner Priorität so bald als möglich gestartet werden.

START = *IMMEDIATELY
Der Auftrag soll unmittelbar gestartet werden.

START = *AT-STREAM-STARTUP
Der Auftrag soll nach dem nächsten Startup des Jobschedulers gestartet werden.

START = *WITHIN(...)
Der Auftrag soll innerhalb des nachfolgend angegebenen Zeitraums gestartet werden.

HOURS = 0 / <integer 0..23 *hours*>
Anzahl Stunden.

MINUTES = 0 / <integer 0..59 *minutes*>
Anzahl Minuten.

START = *AT(...)
Der Auftrag soll exakt zum nachfolgend angegebenen Zeitpunkt gestartet werden.

DATE = *TODAY / <date>
Datum. Der Benutzer kann das Datum in der Form [yy]yy-mm-dd angeben, wobei jedoch nur die letzten zwei Ziffern der Jahreszahl ausgewertet werden, d.h. bei vierstelligen Jahreszahlen wird die Jahrhundertangabe ignoriert!
Jahreszahlen < 80 werden mit 20, Angaben ≥ 80 mit 19 ergänzt.

TIME = <time>
Uhrzeit im Format hh:mm, wobei hh = Stunden und mm = Minuten sind.
Sekundenangabe wird ignoriert.

START = *EARLIEST(...)
Der Auftrag soll frühestens zum nachfolgend angegebenen Zeitpunkt gestartet werden.

DATE = *TODAY / <date>
Datum. Der Benutzer kann das Datum in der Form [yy]yy-mm-dd angeben, wobei jedoch nur die letzten zwei Ziffern der Jahreszahl ausgewertet werden, d.h. bei vierstelligen Jahreszahlen wird die Jahrhundertangabe ignoriert!
Jahreszahlen < 80 werden mit 20, Angaben ≥ 80 mit 19 ergänzt.

TIME = <time>
Uhrzeit im Format hh:mm, wobei hh = Stunden und mm = Minuten sind.
Sekundenangabe wird ignoriert.

START = *LATEST(...)
Der Auftrag soll spätestens zum nachfolgend angegebenen Zeitpunkt gestartet werden.

DATE = *TODAY / <date>
Datum. Der Benutzer kann das Datum in der Form [yy]yy-mm-dd angeben, wobei jedoch nur die letzten zwei Ziffern der Jahreszahl ausgewertet werden, d.h. bei vierstelligen Jahreszahlen wird die Jahrhundertangabe ignoriert!
Jahreszahlen < 80 werden mit 20, Angaben ≥ 80 mit 19 ergänzt.

TIME = <time>
Uhrzeit im Format hh:mm, wobei hh = Stunden und mm = Minuten sind. Sekundenangabe wird ignoriert.

REPEAT-JOB =
Zeitintervall, in dem der Stapelauftrag wiederholt werden soll. Angaben abweichend von *STD werden nur ausgeführt, wenn sie gemäß Jobklassendefinition erlaubt sind (siehe Kommando SHOW-JOB-CLASS). Die Zeitbasis für die Wiederholungen ist abhängig von der Angabe im Operanden START; siehe dazu den Hinweis „Kombinationen der Operanden START und REPEAT-JOB“. Für die Wiederholungen gilt:

– Die i-te Wiederholung (i≥1) eines Auftrages wird nur dann gestartet, wenn die (i-1)-te Ausführung beendet ist.
– Abbrechen des gerade laufenden Auftrages (i) hat keine Auswirkung auf den Start von (i+1); (i≥0).
– Abbruch des gesamten Auftrages: es muss sowohl der gerade laufende Auftrag (i) als auch der Folgeauftrag (i+1) abgebrochen werden, (i≥0); (CANCEL-JOB-Kommando oder mit Kommando MODIFY-JOB..., REPEAT-JOB=*NO den Auftrag (i) zum letzten Auftrag der Folge machen).

REPEAT-JOB = *STD
Es gilt der Standardwert für die gewählte Jobklasse.

REPEAT-JOB = *NO
Der Stapelauftrag wird nicht wiederholt.

REPEAT-JOB = *DAILY
Tägliche Wiederholung zu der mit START angegebenen Uhrzeit.

REPEAT-JOB = *WEEKLY
Wöchentliche Wiederholung zu der mit START angegebenen Uhrzeit.

REPEAT-JOB = *AT-STREAM-STARTUP
Wiederholung nach jedem Startup des Jobschedulers.

REPEAT-JOB = *PERIOD(...)
Wiederholung nach dem angegebenen Zeitintervall.

HOURS = 0 / <integer 0..23 *hours*>
Anzahl Stunden.

MINUTES = 0 / <integer 0..59 *minutes*>
Anzahl Minuten.

SCHEDULING-TIME = *BY-CALENDAR(...)
Der Startzeitpunkt des Stapelauftrags und mögliche Wiederholungen werden durch ein symbolisches Datum, das in einer Kalenderdatei definiert ist, festgelegt (Kalenderjob). Die Einträge einer Kalenderdatei können mit dem Kommando SHOW-CALENDAR ausgegeben werden. Die Erstellung von Kalenderdateien mit dem Dienstprogramm CALENDAR ist im Handbuch „CALENDAR“ [23] beschrieben.

CALENDAR-NAME = <filename 1..54 without-cat-user-gen-vers>
Name der Kalenderdatei.

SYMBOLIC-DATE = <filename 1..20 without-cat-user-vers> / <partial-filename 2..20 without-cat-user>
Symbolisches Datum, das den Startzeitpunkt und ggf. Wiederholungszyklen innerhalb der Kalenderdatei bezeichnet. Das symbolische Datum kann auch teilqualifiziert angegeben werden. Damit können bei entsprechender Definition von SYMDATs mehrere Startzeiten für einen Kalendertag vereinbart werden.

Beispiel: Definition von SYMDATs in der Kalenderdatei:
- WORK.DAY.1 (jeden zweiten Tag um 06:00 Uhr)
- WORK.DAY.2 (jeden zweiten Tag um 18:00 Uhr)
- WORK.WEEK.1 (jeden Freitag um 21:00 Uhr)

Mit `SYMBOLIC-DATE=WORK.` wird ein Kalenderjob gestartet, der alle 3 Startzeitpunkte berücksichtigt.

LIMIT = *STD / <integer 1..32767> / *BY-DATE(...)
Bestimmt die Lebensdauer eines Kalenderjobs. Diese Begrenzung gilt zusätzlich zu den Grenzen, die durch den Kalender gesetzt sind.

LIMIT = *STD
Die Lebensdauer des Kalenderjobs bestimmt sich allein aus dem Eintrag des symbolischen Datums im Kalender.

LIMIT = <integer 1..32767>
Die Angabe ist nur für Kalenderjobs zulässig.
Anzahl der maximalen Auftragswiederholungen des Kalenderjobs.
Nach Beendigung wird geprüft, ob der Ablaufzähler die maximale Anzahl erreicht bzw. überschritten hat. Trifft dies zu, wird der gesamte Kalenderjob beendet. Andernfalls wird der Ablaufzähler um 1 erhöht.

LIMIT = *BY-DATE(...)
Die Angabe ist nur für Kalenderjobs zulässig.
Nach Erreichen des angegebenen Datums werden keine Wiederholungsaufträge des Kalenderjobs gestartet. Ein noch laufender Wiederholungsauftrag wird bei Erreichen des Datums abgebrochen.
Das angegebene Datum bezieht sich nur auf das errechnete Startdatum der Wiederholungsaufträge. Überschreitungen, die sich durch das Nachholen ausgefallener Wiederholungen oder durch Verzögerungen des Job-Schedulers ergeben werden zugelassen.

Das Datum wird bestimmt durch Angabe des Tages und der Uhrzeit:

DATE = <date>
Datum. Der Benutzer kann das Datum in der Form [yy]yy-mm-dd angeben, wobei jedoch nur die letzten zwei Ziffern der Jahreszahl ausgewertet werden, d.h. bei vierstelligen Jahreszahlen wird die Jahrhundertangabe ignoriert!
Jahreszahlen < 80 werden mit 20, Angaben ≥ 80 mit 19 ergänzt.

TIME = <time>
Angabe einer Tageszeit.

RESOURCES = *PARAMETERS(...)
Angaben zu Run-Priorität, CPU-Zeit, Maximalanzahl SYSLST- und SYSOPT-Sätze.

RUN-PRIORITY = *STD / <integer 30..255>
Run-Priorität, die der Stapelauftrag erhalten soll. Je niedriger der Wert, desto höher die Priorität.
Der Wert für die maximal zulässige Priorität ist das numerische Minimum der Maximalwerte (also der günstigere Wert) aus dem Benutzerkatalog und aus der Jobklassendefinition.
Ist für die Jobklasse kein Maximalwert definiert, gelten folgende Regeln:
– Ist der explizit angegebene Wert numerisch kleiner als der Wert im Benutzereintrag, wird die Meldung JMS0045 ausgegeben. Der Stapelauftrag erhält den numerisch größeren Wert (also den ungünstigeren Wert) aus dem direkten Vergleich zwischen der Run-Priorität im Benutzereintrag und der Standard-Run-Priorität der Jobklasse.
– Ohne Angabe eines Wertes bzw. bei expliziter Angabe von *STD erhält der Stapelauftrag die Standard-Run-Priorität der Jobklasse.

Die Werte können mit den Kommandos SHOW-USER-ATTRIBUTES und SHOW-JOB-CLASS abgefragt werden.

RUN-PRIORITY = *STD
Es gilt die für die Jobklasse festgelegte Standardpriorität.

CPU-LIMIT = *STD / *NO / <integer 1..32767 *seconds*>
Maximale CPU-Zeit in Sekunden, die der Stapelauftrag verbrauchen darf. Die maximal erlaubte Zeit hängt von der vereinbarten Jobklasse ab.
Siehe auch Abschnitt „Zeitlimitierungen im BS2000“ im Handbuch „Kommandos, Bd. 1-5“ [3].

CPU-LIMIT = *STD
Es gilt der Standardwert für die gewählte Jobklasse.

CPU-LIMIT = *NO
Der ENTER-Auftrag soll ohne Zeitbegrenzung laufen (NTL = No Time Limit). Der Operandenwert ist nur erlaubt, wenn eine entsprechende Berechtigung im Benutzereintrag vorliegt.

SYSLST-LIMIT = *STD / *NO / <integer 0..999999>
Bezeichnet die maximale Anzahl von Sätzen, die vom Auftrag in die Systemdateien SYSLST, SYSLST01, SYSLST02, ..., SYSLST99 ausgegeben werden. Datensätze in der Systemdatei SYSOUT, die gleichzeitig nach SYSLST geschrieben werden, zählen nicht mit.
Die Angabe darf die in der Jobklassendefinition festgelegte Grenze nicht überschreiten. Dies kann mit dem Kommando SHOW-JOB-CLASS abgefragt werden.

SYSLST-LIMIT = *STD
Standardwert der gewählten Jobklasse. Bei Überschreitung der angegebenen Anzahl gilt:
– im Stapelbetrieb wird der Auftrag abnormal beendet;
– im Dialogbetrieb kann der Anwender angeben, ob der Auftrag fortgesetzt oder beendet werden soll. Bei Fortsetzung wird wieder bis „Anzahl“ ausgegeben.

SYSLST-LIMIT = *NO
Anzahl der Sätze ist nicht begrenzt.

SYSOPT-LIMIT = *STD / *NO / <integer 0..999999>
Bezeichnet die maximale Anzahl von Sätzen, die vom Auftrag in die Systemdatei SYSOPT ausgegeben werden.
Die Angabe darf die in der Jobklassendefinition festgelegte Grenze nicht überschreiten. Dies kann mit dem Kommando SHOW-JOB-CLASS abgefragt werden.

SYSOPT-LIMIT = *STD
Standardwert der gewählten Jobklasse. Bei Überschreitung der angegebenen Anzahl gilt:
– im Stapelbetrieb wird der Auftrag abnormal beendet;
– im Dialogbetrieb kann der Anwender angeben, ob der Auftrag fortgesetzt oder beendet werden soll. Bei Fortsetzung wird wieder bis „Anzahl“ ausgegeben.

SYSOPT-LIMIT = *NO
Anzahl der Sätze ist nicht begrenzt.

LISTING = *NO / *YES
Gibt an, ob der Auftragsablauf zusätzlich auf SYSLST zu protokollieren ist.

LOGGING =
Steuert die Protokollierung des Auftragablaufs. Für Nicht-S-Prozeduren wird der Operand LOGGING ignoriert, da die Protokollierung dort im Kommando BEGIN-PROCEDURE festgelegt wird.

LOGGING = *STD
Es wird nur protokolliert, falls die Prozedurdatei nicht lesegeschützt ist.

LOGGING = *YES
Der Auftragsablauf wird protokolliert. Der Ablauf einer kennwortgeschützten Datei kann nur protokolliert werden, wenn das Kennwort für den Lesezugriff mit dem Kommando ADD-PASSWORD in die Kennwort-Tabelle des Auftraggebers eingetragen wurde oder im Operanden PROCEDURE-PASSWORD angegeben ist.

LOGGING = *NO
Der Auftragsablauf wird nicht protokolliert.

JOB-PARAMETER =
Angabe zusätzlicher Attribute für die gewählte Jobklasse, sofern die Systembetreuung solche definiert und bekannt gegeben hat.

JOB-PARAMETER = *NO
Keine zusätzlichen Attribute.

JOB-PARAMETER = <c-string 1..127>
Zeichenfolge = Folge beliebiger Zeichen; wird von der Systembetreuung zur Kennzeichnung weiterer Jobklassenattribute vergeben.

SYSTEM-OUTPUT =
Steuert die Ausgabe der Systemdateien SYSLST und SYSOUT bei Auftragsbeendigung (siehe auch Kommando EXIT-JOB, Operand SYSTEM-OUTPUT).

SYSTEM-OUTPUT = *STD
Gibt bei fehlerhaftem Ablauf die Systemdateien SYSLST und SYSOUT auf Drucker aus.

SYSTEM-OUTPUT = *PRINT
Gibt die Systemdateien SYSLST und SYSOUT auf Drucker aus.

SYSTEM-OUTPUT = *DELETE
Die Ausgabe der Systemdateien SYSLST und SYSOUT wird unterdrückt.

ASSIGN-SYSTEM-FILES =
Gibt, an welche Zuordnung für die Systemdateien SYSLST und SYSOUT zu Beginn des Stapelauftrags gelten soll. Der Operand ermöglicht es, den beiden Systemdateien bei einem asynchronen Prozedurlauf katalogisierte Dateien zuzuweisen. Die Prozedurdatei kann somit z.B. unverändert im Dialog weiterhin nach SYSOUT ausgeben und im Stapelbetrieb in eine katalogisierte Datei. Die Zuordnung zu Pseudodateien (*DUMMY) wird ebenfalls unterstützt (siehe auch FILE-NAME=*DUMMY im Kommando ADD-FILE-LINK).

ASSIGN-SYSTEM-FILES = *STD
Voreingestellt ist die Primärzuweisung für SYSLST und SYSOUT, d.h. Ausgabe auf Drucker nach Beendigung der Task (abhängig von SYSTEM-OUTPUT), wenn innerhalb der Prozedur keine Änderung der Zuweisung erfolgt.

ASSIGN-SYSTEM-FILES = *PARAMETERS(...)
Gibt an, welche Zuweisung für SYSLST und SYSOUT bei Beginn des Stapelauftrags gelten soll.

SYSLST = *STD / *PRIMARY / *DUMMY / <filename 1..54>
Ausgabeziel, dem die Systemdatei SYSLST zuzuordnen ist. Voreingestellt ist *STD, d.h. die bestehende Zuordnung wird nicht verändert.

SYSOUT = *STD / *PRIMARY / *DUMMY / <filename 1..54>
Ausgabeziel, dem die Systemdatei SYSOUT zuzuordnen ist. Voreingestellt ist *STD, d.h. die bestehende Zuordnung wird nicht verändert.

PROTECTION = *NONE / *CANCEL
Gibt an, ob der Stapelauftrag gegen eine versehentliche Beendigung mit dem Kommando CANCEL-JOB geschützt sein soll.

PROTECTION = *NONE
Der Stapelauftrag ist nicht gegen eine versehentliche Beendigung geschützt.

PROTECTION = *CANCEL
Der Stapelauftrag ist gegen eine versehentliche Beendigung geschützt. In Dialogaufträgen, die diesen Stapelauftrag mit dem Kommando CANCEL-JOB beenden wollen, fordert das System zusätzlich eine Bestätigung an. Ein versehentliches Beenden des Stapelauftrags durch fehlerhafte Angabe der Auftragsnummer soll somit verhindert werden.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Kommando ausgeführt
2	0	CMD0002	Kommando ausgeführt mit Warnung z.B. DELETE= *YES ignoriert für Repeatjob oder bei Angabe eines Bibliothekelements
	1	CMD0202	Syntaxfehler im Kommando
	32	CMD0221	Systemfehler
	64	JMS0630	Semantik- oder Privilegienfehler z.B. Rechner, Katalogkennung, Jobklasse unbekannt; MONJV nicht zugreifbar
	64	JMS0640	Fehler bei Zugriff auf die Prozedurdatei z.B. keine SAM- bzw. ISAM-Datei, Datei leer, fehlendes Zugriffsrecht
	64	JMS0670	Fehler bei einem REMOTE-Auftrag
	130	JMS0620	Kein weiterer Speicherplatz oder TSN verfügbar; oder angegebene MONJV überwacht bereits einen Auftrag
	130	JMS0650	MSCF nicht verfügbar oder keine Verbindung zum angegebenen Rechner

Hinweise

– Kombinationen der Operanden START und REPEAT-JOB:

	REPEAT-JOB		
START	**AT-STREAM-STARTUP**	**DAILY bzw. WEEKLY**	**PERIOD**
IMMEDIATELY bzw. SOON	a)	c)	c)
AT bzw. EARLIEST	a)	d)	f)
LATEST bzw. WITHIN	a)	c)	g)
AT-STREAM-STARTUP	b)	e)	h)

a) Der erste und alle weiteren Starts des Auftrages erfolgen wie spezifiziert.

b) Der erste Start des Auftrages erfolgt mit START=*AT-STREAM-STARTUP. Alle weiteren Starts erfolgen nach dem Startup des Jobschedulers mit START= *SOON.

c) Zeitbasis für den Wiederholungszyklus ist der Zeitpunkt der Jobannahme.

d) Der angegebene Zeitpunkt (START=...., TIME=....) ist die Zeitbasis für den Wiederholungszyklus.

e) Der erste Start des Auftrages erfolgt nach dem Startup des Jobschedulers. Diese Startzeit ist die Zeitbasis für den Wiederholungszyklus. Die weiteren Starts erfolgen mit START=SOON.

f) Der angegebene Zeitpunkt (START=...., TIME=....) ist die Zeitbasis für den Wiederholungszyklus. Der zweite und alle weiteren Starts erfolgen mit START= *SOON.

g) Zeitbasis für den Wiederholungszyklus ist der Zeitpunkt der Jobannahme. Alle weiteren Starts erfolgen mit START=*SOON.

h) Zeitbasis für den Wiederholungszyklus ist die erste Startzeit. Der erste Start des Auftrages erfolgt nach dem Startup des Jobschedulers. Die weiteren Starts erfolgen mit START=*SOON.

– Für die Angaben *WITHIN, *AT und *LATEST im Operand START gilt:
Nach dem angegebenen Zeitpunkt bzw. Zeitraum werden nicht gestartete Aufträge behandelt wie Aufträge, die mit START=*SOON und höchster Jobpriorität gestartet wurden.

Beispiel

Ein Auftrag mit START=*LATEST(...) konnte bis zum gewünschten Zeitpunkt nicht gestartet werden, da der Jobscheduler nicht aktiv war. Der Start erfolgt dann (innerhalb derselben Session) baldmöglichst nach dem nächsten Startup des Jobschedulers.

– Ermittlung des Startzeitpunktes eines Kalenderjobs:

  – Für die erste Ausprägung des Kalenderjobs wird bei der Auswertung der Jobattribute das im Operanden SYMBOLIC-DATE angegebene symbolische Datum (SYMDAT) an die Komponente CALENDAR übergeben. Diese liefert als konkreten Startzeitpunkt den im Vergleich mit der aktuellen Zeit nächstfolgenden Zeitpunkt (Datum und Uhrzeit) zurück, der durch die in der Kalenderdatei definierten SYMDATs vorgegeben ist.
  Bei Angabe eines teilqualifizierten SYMDATs wird für jedes SYMDAT, das mit dieser Zeichenfolge beginnt, ein Startzeitpunkt zurückgeliefert und es werden entsprechend viele Jobs gestartet.
  – Die Startzeitpunkte der folgenden Ausprägungen werden analog während der Terminierung des Vorgängerjobs ermittelt.

  Änderungen in der Kalenderdatei wirken somit nur auf Ausprägungen von Kalenderjobs, deren Startzeitpunkte nach der Aktualisierung der Kalenderdatei ermittelt werden. Insbesondere kann die Anzahl der Ausprägungen eines Kalenderjobs, der mit teilqualifiziertem SYMDAT gestartet wurde, nachträglich erweitert (Definition neuer SYMDATs) oder reduziert (Löschen von SYMDATs) werden.

– Jobvariablen (JV) stehen nur dem Anwender mit dem Softwareprodukt JV zur Verfügung (siehe auch Handbuch „Jobvariablen“ [5]).

- Ein Stapelauftrag, der auf einem Remote-Rechner ablaufen soll, ist über eine MONJV nur zugreifbar, wenn im MRSCAT der beteiligten Rechnern jeweils die Katalogkennung des Home-Pubsets des Partnerrechners eingetragen ist.
- Der Zugriff über RFA auf Prozedurdateien ist nicht möglich.
- Damit die Verfügbarkeit der Prozedurdatei zum Ausführungszeitpunkt (Start des Stapelauftrags) sichergestellt ist, wird immer eine Kopie der Datei mit dem Präfix S.PROC. erstellt.
- Die restlichen Operanden bestimmen die Ausführung der Enter-Datei. Sie sind identisch mit denen des ENTER-JOB-Kommandos. Folgende Operanden des Kommandos ENTER-JOB werden nicht unterstützt:

FILE-PASSWORD	Die erzeugte Enter-Datei erhält ein Zufallskennwort, mit dem der Operand FILE-PASSWORD versorgt wird.
DELETE	Es gilt immer DELETE=*YES

- Die erzeugte Enter-Datei wird immer mit DELETE=*YES gestartet.
  Die Enter-Datei und die Kopie der Prozedurdatei (S.PROC-Datei) wird bei Auftragsende nicht automatisch gelöscht, wenn der Auftrag wiederholt werden soll (Operand REPEAT).
- Die Kopie der Prozedurdatei (S.PROC-Datei) ist kennwortgeschützt, kann aber ohne Angabe des Kennwortes vom Eigentümer gelöscht werden. Der Benutzer kann somit S.PROC-Dateien löschen, die wegen Systemfehlers nicht mehr gelöscht wurden. S.PROC-Dateien für Wiederholungsaufträge darf der Benutzer nicht löschen.
  Das Gleiche gilt für den Kennwortschutz der S.E-Datei.
- Die S.PROC- und S.E-Dateien werden abhängig von dem Systemparameter DESTLEV mit DESTROY-BY-DELETE=*YES eingerichtet.
- Prozedurdateien können mit Kennwörtern gegen Lesen, Überschreiben und Ausführen geschützt werden. Das Ausführungs-Kennwort oder ein höherwertiges Kennwort muss in die Kennworttabelle des aufrufenden Auftrags eingetragen sein, entweder durch Angabe im Operanden PROCEDURE-PASSWORD oder vorher durch ein ADD-PASSWORD-Kommando.
  Ein Lesekennwort muss angegeben werden, wenn der Auftragsablauf protokolliert werden soll.
  Die Kennwörter werden bei der Verarbeitung des ENTER-PROCEDURE-Kommandos geprüft. Werden die Kennwörter danach geändert, so gilt die erfolgreiche Prüfung der ENTER-PROCEDURE-Verarbeitung, und die Prozedurdatei kann ausgeführt werden.
  Ist die auszuführende Prozedur Element einer PLAM-Bibliothek, so wird ein im Operanden PROCEDURE-PASSWORD angegebenes Kennwort nur als Kennwort für den Zugriff auf die PLAM-Bibliothek interpretiert.

– S.-Dateien sind stets unverschlüsselt. Die Kopie der Prozedurdatei muss bei der Kommandobearbeitung entschlüsselt werden können, sei es durch Angabe des Operanden CRYPTO-PASSWORD oder durch das in der Crypto-Kennworttabelle eingetragene Crypto-Kennwort (s. Kommando ADD-CRYPTO-PASSWORD).
Soll der Stapelauftrag an einem Remote-Rechner ablaufen, kann das Crypto-Kennwort nur über den Operanden spezifiziert werden. Für ferne Rechner mit BS2000/OSD < V6.0 hat der Operand keine Bedeutung.

– Die S.PROC- und S.E-Dateien sind während der Lebensdauer des Stapelauftrags durch eine Dateisperre geschützt.
Dabei ist zu beachten: Die Dateisperren werden mit dem Import des Pubset, auf dem sich die Dateien befinden, gesetzt. Es werden dabei nur solche Dateien berücksichtigt, auf die Stapelaufträge aus dem aktuellen Jobpool (auf dem Home-Pubset) Bezug nehmen. Liegen die Dateien auf einem Shared-Pubset, werden die Dateisperren über den Master-Rechner koordiniert.

Beispiel

Prozeduraufruf:

```
/ENTER-PROCEDURE P, (PAR1 = 'ABC', PAR2 = 'XYZ')
```

Es wird eine Datei S.E.tsn.date.time mit folgendem Inhalt erzeugt:

```
/SET-LOGON-PARAMETERS
...
/CALL-PROCEDURE S.PROC.tsn.date.time,(PAR1='ABC',PAR2 ='XYZ'),LOGGING= *YES
...
/EXIT-JOB
```

Die Hintergrund-Prozedur wird gestartet (SET-LOGON-PARAMETERS).
Darin wird die Prozedur S.PROC.tsn.date.time aufgerufen (CALL-PROCEDURE S.PROC.tsn.date.time). Nach Beendigung des Prozedurlaufs wird die Hintergrund-Prozedur beendet (EXIT-JOB).

EXECUTE-CMD
Kommando ausführen und strukturierte Ausgabe

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando EXECUTE-CMD übergibt das beim Operanden CMD=... angegebene Kommando zur Ausführung und schreibt die Kommandoausgaben (Meldungen, Ausgabeinformation) in eine anzugebende Variable. Es können unstrukturierte und strukturierte Ausgaben erzeugt werden, je nach Leistung des Kommandoservers. Das Verhalten bei fehlerhafter Kommandoausführung kann durch Auswerten des Returncodes gesteuert werden.

Die meisten SHOW-Kommandos liefern strukturierte Ausgaben. Die Ausgabestrukturen sind im Handbuch „Kommandos, Bd. 6“ [4] bzw. im jeweiligen Produkthandbuch beschrieben.

Garantierte Meldungen können strukturiert in Variablen gelenkt werden. Struktur und Vorgang sind im Handbuch „MSGMAKER“ [21] beschrieben.

Hinweis

– Bei EXECUTE-CMD sollten nicht Kommandos wie LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM) u.Ä. angegeben werden. Werden solche Kommandos doch angegeben, so werden alle Operanden von /EXECUTE-CMD ignoriert und die Fehlermeldung SDP0229 wird zurückgegeben. Die Kommandos werden dann ausgeführt wie außerhalb von EXECUTE-CMD.
– AID-Kommandos sind mit EXECUTE-CMD nicht ausführbar.

Format

EXECUTE-**CMD**

CMD = <text 0..1800 with-low>
,**TEXT-OUTPUT** = ***SYSOUT** / ***NONE** / <composed-name 1..255>(...)
<composed-name 1..255>(...)
WRITE-MODE = ***REPLACE** / ***EXTEND**
,**STRUCTURE-OUTPUT** = ***NONE** / <composed-name 1..255>(...)
<composed-name 1..255>(...)
WRITE-MODE = ***REPLACE** / ***EXTEND**
,**MSG-STRUCTURE-OUTPUT** = ***NONE** / <composed-name 1..255>(...)
<composed-name 1..255>(...)
WRITE-MODE = ***REPLACE** / ***EXTEND** /
,**RET**URNCODE = ***STD** / ***NONE** / ***VAR**IABLE(...)
***VAR**IABLE(...)
SUBCODE1 = ***NONE** / <composed-name 1..255>
,**SUBCODE2** = ***NONE** / <composed-name 1..255>
,**MAIN**CODE = ***NONE** / <composed-name 1..255>

Operandenbeschreibung

CMD = <text 0...1800 with-low>
Auszuführendes Kommando (eingeschlossen in runde Klammern, ohne Schrägstrich).

TEXT-OUTPUT =
Ausgabe als Textstring. Falls das auszuführende Kommando durch ein TU (Task Unprivileged)-Programm oder durch eine Prozedur implementiert worden ist, so ist es nicht möglich die SYSOUT-Ausgaben zu ignorieren oder in eine Variable umzulenken.

TEXT-OUTPUT = *SYSOUT
Ausgabe nach SYSOUT

TEXT-OUTPUT = *NONE
Es soll keine unstrukturierte Ausgabe erfolgen.

TEXT-OUTPUT = <composed-name 1..255>(...)
Ausgabe in eine Listenvariable. Die Listenvariable muss deklariert sein mit MULTIPLE-ELEMENTS = *LIST im Kommando DECLARE-VARIABLE. Eine Ausgabezeile des SYSOUT-Layouts entspricht dann einem Listenelement. Das SYSOUT-Layout ist versionsabhängig und kann nicht garantiert werden.

WRITE-MODE = *REPLACE / *EXTEND
Gibt an, ob die Liste überschrieben oder erweitert wird.
Mit *REPLACE wird die Liste überschrieben.

STRUCTURE-OUTPUT =
Strukturierte Ausgabe mit Meldungen.
Wird STRUCTURE-OUTPUT für Kommandos spezifiziert, die diese Ausgaben nicht erbringen, wird keine Fehlermeldung zurückgeschickt. Ausnahmen sind die Kommandos SHOW-USER-STATUS und SHOW-SYSTEM-STATUS.

STRUCTURE-OUTPUT = *NONE
Eine strukturierte Ausgabe soll nicht erzeugt werden. Für das angegebene Kommando erfolgt auch keine strukturierte Ausgabe in einen S-Variablenstrom, unabhängig von der Zuweisung mit dem Kommando ASSIGN-STREAM.

STRUCTURE-OUTPUT = <composed-name 1..255>(...)
Name der Variablen, in welche die strukturierte Ausgabe erfolgen soll. Die Variable muss als Listenvariable deklariert sein (MULTIPLE-ELEMENTS = *LIST im Kommando DECLARE-VARIABLE).

WRITE-MODE = *REPLACE / *EXTEND
Gibt an, ob die Liste überschrieben oder erweitert wird.
Mit *REPLACE wird die Liste überschrieben.

MSG-STRUCTURE-OUTPUT =
Strukturierte Ausgabe der Meldungen.

MSG-STRUCTURE-OUTPUT = *NONE
Eine strukturierte Ausgabe soll nicht erzeugt werden. Für das angegebene Kommando erfolgt auch keine strukturierte Ausgabe in einen S-Variablenstrom, unabhängig von der Zuweisung mit dem Kommando ASSIGN-STREAM.

MSG-STRUCTURE-OUTPUT = <composed-name 1..255>(...)
Name der Variablen, in welche die strukturierte Ausgabe der Meldungen geschrieben wird. Die Variable muss als Listenvariable deklariert sein (MULTIPLE-ELEMENTS = *LIST im Kommando DECLARE-VARIABLE).
Näheres siehe Abschnitt „Strukturierte Ausgaben in S-Variablen“ auf Seite 201.

WRITE-MODE = *REPLACE / *EXTEND
Gibt an, ob die Liste überschrieben oder erweitert wird.
Mit *REPLACE wird die Liste überschrieben.

RETURNCODE =
Legt das Verhalten bei fehlerhafter Kommandoausführung fest.

RETURNCODE = *STD
Bei fehlerhafter Kommandoausführung wird das nächste IF-BLOCK-ERROR-Kommando gesucht und die dort definierte Fehlerbehandlung durchgeführt.

RETURNCODE = *NONE
Es wird keine Fehlerinformation ausgegeben und keine Fehlerbehandlung durchgeführt.

RETURNCODE = *VARIABLE(...)
Die Kommando-Returncodes werden in Variablen ausgegeben; die Fehlerbehandlung wird nicht automatisch durchgeführt.

SUBCODE1 = *NONE / <composed-name 1..255>
Bezeichnet eine Variable, in die der Subcode1 ausgegeben wird (Subcode1 = Fehlerklasse).
Die Variable muss mit TYPE = *ANY oder TYPE = *INTEGER deklariert sein.

SUBCODE2 = *NONE / <composed-name 1..255>
Bezeichnet eine Variable, in die der Subcode2 ausgegeben wird (Subcode2 = Zusatzinformation zu Subcode1).
Die Variable muss mit TYPE = *ANY oder TYPE = *INTEGER deklariert sein.

MAINCODE = *NONE / <composed-name 1..255>
Bezeichnet eine Variable, in die der Fehlerschlüssel ausgegeben wird (Maincode = Fehlerschlüssel).
Die Variable muss mit TYPE = *ANY oder TYPE = *STRING deklariert sein.

Hinweis

Der Defaultwert von STRUCTURE-OUTPUT und MSG-STRUCTURE-OUTPUT ist jeweils *NONE. Das bedeutet, wenn diese Operanden in EXECUTE-CMD nicht explizit angegeben werden bzw. ihnen dieser Wert zugewiesen wird, werden keine Variablen nach SYSINF und/oder SYSMSG geschrieben, auch wenn diesen Strömen vor dem EXECUTE-CMD mit einem ASSIGN-STREAM-Kommando ein Ziel zugewiesen wurde. Denn diese Zuweisung wird im Rahmen der EXECUTE-CMD mit „*NONE“ (d.h. mit einem *DUMMY-Wert) überschrieben.

Kommando-Returncode

Der Returncode hängt von der Einstellung beim Operanden RETURNCODE ab.

*Bei RETURNCODE = *STD ist möglich:*

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar
xx	xx	xxxxxxx	sonstige Returncodes des ausgeführten Kommandos

*Bei Returncode = *NONE / *VARIABLE(...) ist möglich:*

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler (aber nur im EXECUTE-CMD)
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

EXIT-BLOCK
Verarbeitung eines Kommandoblocks abbrechen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

EXIT-BLOCK bricht die Verarbeitung eines Kommandoblocks ab (BEGIN-, FOR-, IF-, WHILE-, REPEAT-Block etc.) und setzt den Prozedurlauf mit dem Kommando fort, das auf das Blockabschlusskommando folgt. Bei einem BEGIN-Block, der mit einem INCLUDE-BLOCK-Kommando aufgerufen wurde, wird der Prozedurlauf mit dem Kommando fortgesetzt, das auf das INCLUDE-BLOCK-Kommando folgt.

Die Ausführung des Kommandos EXIT-BLOCK kann von einer Bedingung abhängig gemacht werden.

Hinweis

– Aus Dialogblöcken ist nur die Rückkehr zur Dialogebene möglich
– Im Kommando ist keine Ausdruckersetzung (&....) möglich.

Format

EXIT-BLOCK
BLOCK = ***LAST** / ***ALL** / <structured-name 1..255> ,**COND**ITION = ***NONE** / <text 1..1800 with-low *bool-expr*>

Operandenbeschreibung

BLOCK =
Bezeichnet den Block, der abgebrochen werden soll.

BLOCK = *LAST
Verweis auf den unmittelbar umgebenden Block; dieser wird abgebrochen.

BLOCK = *ALL
Die Verarbeitung aller umgebenden Blöcke wird abgebrochen; aus einem Dialogblock wird in die Dialogebene zurückgekehrt.

BLOCK = <structured-name 1..255>
Verweis auf die Marke des umgebenden Blocks, der abgebrochen werden soll.

CONDITION =
Beschreibt eine Bedingung für die Ausführung des EXIT-BLOCK-Kommandos.

CONDITION = *NONE
Das Kommando wird immer ausgeführt.

CONDITION = <text 1..1800 with-low *bool-expr*>
Das Kommando wird nur ausgeführt, wenn der angegebene boolsche Ausdruck den Wert „TRUE“ ergibt.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler (unkorrekter Ausdruck)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1

```
/LOOP: WHILE (BED < 9)
:
:
/IF (EING='*END')
/WRITE-TEXT 'Verarbeitung beendet'
/EXIT-BLOCK LOOP
/END-IF
:
:
/END-WHILE
/"Hier geht es nach Ausführung von EXIT-BLOCK weiter"
```

Beispiel 2

```
/J=0
/FOR I=('Zeile 1','Zeile 2','Zeile 3','Zeile 4)
/J=J+1
/EXIT-BLOCK BLOCK=*LAST,CONDITION=(J>3)
/SHOW-VARIABLE I
/END-FOR
```

Ausgabe:

```
I = Zeile 1
I = Zeile 2
I = Zeile 3
```

Das vierte Element der Liste wird nicht mehr ausgewertet.

EXIT-PROCEDURE
Prozedur beenden

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

EXIT-PROCEDURE beendet die aktuelle Prozedur und gibt die Ablaufsteuerung an die aufrufende Ebene zurück.

Auch bei ordnungsgemäßem Abschluss der Prozedur, das heißt ohne Fehler, können Fehlerinformationen weitergereicht werden. Dies muss im Operanden ERROR festgelegt werden. Die Angabe ERROR=*YES stößt die Fehlerbehandlung an, wenn SUBCODE1 nicht Null ist.

Im Operanden RESUME-PROGRAM kann vereinbart werden, dass ein geladenes Programm nach Beendigung der Prozedur fortgesetzt wird.

Fehlt das Kommando EXIT-PROCEDURE am Prozedurende, wird die Prozedur automatisch beendet, wenn bei der Ausführung das Prozedurende erreicht ist. Im Fehlerfall wird auch der Fehlercode an den Aufrufer zurückgegeben.

Format

EXIT-PROCEDURE

ERROR = ***NO** (...) / ***Y**ES(...)

***NO**(...)

SUBCODE2 = **0** / < integer 0..255>

,**MAIN**CODE = **CMD0001** / <alphanum-name 7..7>

***Y**ES(...)

SUBCODE1 = **64** / <integer 0..255>

,**SUBCODE2** = **0** / <integer 0..255>

,**MAIN**CODE = **SDP0018** / <alphanum-name 7..7>

,**RESU**ME-**PROG**RAM = ***NO** / ***Y**ES

Operandenbeschreibung

ERROR =
Returncode, der an den Aufrufer zurückgegeben wird.

ERROR = *NO(...)
Der Returncode der Fehlerklasse „NO-ERROR“ wird zurückgegeben. Mit den Operanden SUBCODE2 und MAINCODE können zusätzliche Informationen übermittelt werden.

SUBCODE2 = 0 / <integer 0..255>
Zusatzinformation zur Fehlerklasse. Voreingestellt ist der Wert Null, d.h. es ist keine Zusatzinformation vorhanden.

MAINCODE = CMD0001 / <alphanum-name 7..7>
Übergibt einen Meldungsschlüssel, dessen Bedeutung sich der Aufrufer mit dem Kommando HELP-MSG-INFORMATION ausgeben lassen kann. Voreingestellt ist CMD0001, d.h. die Prozedur wurde fehlerfrei beendet.

ERROR = *YES(...)
Der Aufrufer erhält einen Returncode, der einen Fehler anzeigt. Mit den Operanden SUBCODE1, SUBCODE2 und MAINCODE können Fehlerklasse und Zusatzinformationen übermittelt werden. Sofern SUBCODE1 nicht Null ist, wird in der rufenden Prozedur die Fehlerbehandlung angestoßen.

SUBCODE1 = 64 / <integer 0..255>
Zahl, die die Fehlerklasse angibt.
Fehlerklasse 64: „SEMANTIC ERROR“.

SUBCODE2 = 0 / <integer 0..255>
Zusatzinformation zur Fehlerklasse. Null: es ist keine Zusatzinformation vorhanden

MAINCODE = SDP 0018
Defaultwert für den Maincode.

MAINCODE = <alphanum-name 7..7>
Übergibt einen Meldungsschlüssel, dessen Bedeutung sich der Aufrufer mit dem SDF-Kommando HELP-MSG-INFORMATION ausgeben lassen kann (siehe auch Handbuch „Systemmeldungen“ [15]).

RESUME-PROGRAM =
Legt fest, ob ein geladenes Programm nach Beendigung der Prozedur fortgesetzt werden soll.

RESUME-PROGRAM = *NO
Das Programm wird nach Prozedurende nicht fortgesetzt.

RESUME-PROGRAM = *YES
Das Programm wird nach Prozedurende fortgesetzt.

Kommando-Returncode

MIT EXIT-PROCEDURE ERROR = *YES kann das Kommando jeden gegebenen Returncode dem Aufrufer melden. Aus der Sicht des Aufrufers ist dies der Returncode des CALL-PROCEDURE oder INCLUDE-PROCEDURE Kommandos. Jedoch wenn die Ausführung des EXIT-PROCEDURE Kommandos selbst zu einem Fehler führt, so wird nicht zum Aufrufer zurückgekehrt, sondern einer der folgenden Returncodes geliefert und die Fehlerbehandlung innerhalb der Prozedur angestoßen.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

FOR
FOR-Block einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando FOR leitet einen FOR-Block ein; das Kommando /END-FOR beendet einen FOR-Block. Das Konstrukt ist eine FOR-Schleife. Der Benutzer definiert eine Laufvariable <composed-name 1..255>. Die Anzahl der Schleifendurchläufe und der Wert der Laufvariablen sind abhängig von der Zuweisung rechts vom Gleichheitszeichen (Ausdruck, Listenvariable (*LIST), „Counter“):

– *LIST(...)
  Die Anzahl der Schleifendurchläufe ist gleich der Anzahl der Variablenelemente. Bei jedem Schleifendurchlauf wird der Laufvariablen in aufsteigender Reihenfolge der Elemente der Wert des Elementes zugewiesen.
– *COUNTER(...)
  Die Anzahl der Schleifendurchläufe wird durch den Anfangswert (FROM=), den Endwert (TO=) und die Schrittweite (INCREMENT=) festgelegt. Die Laufvariable enthält jeweils den aktuellen Schleifenwert.
– <text 0..1800 with-low>
  Die Anzahl der Schleifendurchläufe ist gleich der Anzahl der Elemente der Werteliste. Bei jedem Schleifendurchlauf wird der Laufvariablen - von links beginnend - der jeweilige Ausdruck (String-, arithmetischer -, boolscher Ausdruck, ...) zugewiesen.

Werden <text 0..1800 with-low>, *LIST(...) und *COUNTER(...) in einer Werteliste gemischt, so wird diese Liste - von links beginnend - abgearbeitet, so wie oben angegeben.

Eine Ausdrucksersetzung (&....) in einem der FOR-Operanden erfolgt nur beim Eintritt in die FOR-Schleife, nicht bei jedem Schleifendurchlauf.

Hinweis

Die Operanden des Kommandos FOR werden von SDF-P ausgewertet und sind, wie nachfolgend dargestellt, einzugeben. Für die Operanden gelten die SDF-Abkürzungsregeln. SDF-Funktionen, wie Auskunft über mögliche Operandenwerte oder Korrekturdialog, sind auf Operandenebene nicht verfügbar. Im geführten Dialog stellt SDF nur ein Eingabefeld mit „# =“ zur Verfügung.

Zu FOR-Blöcken siehe auch Abschnitt „Schleifen definieren“ auf Seite 97.

Format

FOR

<composed-name 1..255> = list-poss(2000): ***LIST**(...) / ***COUNTER** (...) / <text 0..1800 with-low *expr*>

***LIST**(...)
LIST-NAME = <composed-name 1..255>

***COUNTER**(...)
FROM = <text 0..1800 *arithm-expr*>
,**TO** = ***UNLIMITED** / <text 0..1800 *arithm-expr*>
,**INCR**EMENT = **1** / <text 0..1800 *arithm-expr*>

,**COND**ITION = ***NONE** / <text 0..1800 with-low *bool-expr*>

Operandenbeschreibung

<composed-name 1..255> =
Bezeichnet die Laufvariable. Die Laufvariable muss vom gleichen Typ sein wie die Elemente rechts vom Gleichheitszeichen oder sie muss konvertierbar sein.

<composed-name 1..255> = *LIST(LIST-NAME=<composed-name 1..255>)
Der Laufvariablen wird eine Listenvariable zugewiesen. Die Anzahl der Schleifendurchläufe ist gleich der Anzahl der Variablenelemente. Bei jedem Schleifendurchlauf wird der Laufvariablen in aufsteigender Reihenfolge der Elemente der Wert des Elementes zugewiesen. Wenn Laufvariable und Listenvariable vom Typ Struktur sind und gleichnamige Elemente haben, so werden die Elemente der Laufvariablen von den gleichnamigen Elementen der Listenvariable überschrieben (siehe Kommando SET-VARIABLE, Operand WRITE-MODE =*REPLACE, Seite 756, Abschnitt „Listen“ auf Seite 145, und Kapitel „Ausdrücke“ auf Seite 255).

LIST-NAME = <composed-name 1..255>
Name der Listenvariablen.

<composed-name 1..255> = *COUNTER (...)
Die Anzahl der Schleifendurchläufe wird durch den Anfangswert (FROM=), den Endwert (TO=) und die Schrittweite (INCREMENT=) festgelegt. Diese Werte können während des Laufs nicht mehr modifiziert werden. Die Laufvariable enthält jeweils den aktuellen Schleifenwert. Der Wert der Laufvariablen kann aber auch während des Schleifendurchlaufs durch direkte Zuweisung modifiziert werden. Die Laufvariable muss vom Typ INTEGER oder konvertierbar sein. Zu „arithmetischer Ausdruck“ siehe Seite 271.

FROM = <text 0..1800 *arithm-expr*>
Anfangswert für die Laufvariable (Zahl oder arithmetischer Ausdruck).

TO = *UNLIMITED / <text 0..1800 *arithm-expr*>
Endwert für die Laufvariable (Zahl oder arithmetischer Ausdruck).
*UNLIMITED bedeutet: Der Endwert ist 2^{31}-1, wenn für die Schrittweite ein Wert > 0 angegeben wurde bzw. -2^{31} für einen Schrittweitenwert < 0. Das Überschreiten dieser Grenzwerte löst einen Fehler aus (Verzweigung zur Fehlerbehandlung).

INCREMENT = 1 / <text 0..1800 *arithm-expr*>
Schrittweite, um die der aktuelle Schleifenwert nach jedem Schleifendurchlauf erhöht wird (Zahl oder arithmetischer Ausdruck). Die Zählrichtung wird durch das Vorzeichen bestimmt. Sie muss in Richtung Endwert führen, sonst wird der Lauf nicht durchgeführt. Ändert das Increment während des Schleifenablaufes seine Zählrichtung (sein Vorzeichen), wird der Lauf abgebrochen.

Hinweis

Bei Schrittweite 0 wird die Schleife solange durchlaufen, bis die angegebene Bedingung (Operand CONDITION=...) nicht mehr erfüllt ist bzw. mit EXIT-BLOCK ein Aussprung aus der Schleife erfolgt.

<composed-name 1..255> = <text 0..1800 with-low *expr*>
Beliebiger Ausdruck (String-, arithmetischer -, boolscher Ausdruck, ...). Die Anzahl der Schleifendurchläufe ist gleich der Anzahl der Elemente der Werteliste. Bei jedem Schleifendurchlauf wird der Laufvariablen - in der Werteliste von links beginnend - der jeweilige Ausdruck (String, Zahl, ...) zugewiesen.

CONDITION = *NONE / <text 0..1800 *bool-expr*>
Bezeichnet eine Bedingung, die am Beginn jeden Schleifendurchlaufs geprüft wird. Die FOR-Schleife wird durchlaufen, wenn die angegebene Bedingung den Wert „TRUE“ besitzt. Der FOR-Lauf wird abgebrochen, wenn die Bedingung den Wert „FALSE“ ergibt.
*NONE bedeutet: Eine Bedingung wird nicht ausgewertet, die Schleife wird immer durchlaufen. Zu „boolschen Ausdruck“ siehe Seite Seite 271.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1

```
/DECLARE-VARIABLE A,MULTIPLE-ELEMENTS=*LIST
/DECLARE-VARIABLE I
/SET-VARIABLE A=1436,WRITE-MODE=*EXTEND
/A=1455,WRITE-MODE=*EXTEND
/A=1577,WRITE-MODE=*EXTEND
/FOR I=*LIST(A)
/CANCEL-JOB JOB-ID=*TSN(&I)
/END-FOR
```

Die FOR-Schleife setzt folgende Kommandos ab:

```
/CANCEL-JOB JOB-ID=*TSN(1436)
/CANCEL-JOB JOB-ID=*TSN(1455)
/CANCEL-JOB JOB-ID=*TSN(1577)
```

Beispiel 2

```
/FOR I=(5,7,12,2,3),CONDITION=(I<10)
/SHOW-VARIABLE I
/END-FOR
```

Ausgabe

```
I = 5
I = 7
```

Die Elemente 12, 2, 3 der Liste werden nicht mehr ausgewertet.

Beispiel 3

Die Primzahlen im Bereich 2 bis zur eingegebenen Zahl werden berechnet und ausgegeben.

```
/       DECL-VAR N(TYPE=*INTEGER)
/       DECL-VAR PRIMARY-NUMBERS(TYPE=*INTEGER),MULT-ELEM=*LIST
/       WR-TEXT 'Bitte Ganzzahl >= 2 eingeben!'
/       READ-VAR N,INPUT=*TERMINAL
/       PRIMARY-NUMBERS# = 2
/LOOP1: FOR I = *COUNTER(FROM=3,TO=N,INCREMENT=2)
/LOOP2:     FOR J = *LIST(LIST-NAME=PRIMARY-NUMBERS)
/               CYCLE LOOP1,COND=(I MOD J == 0)
/           END-FOR LOOP2
/           PRIMARY-NUMBERS = I ,MODE=*EXTEND
/       END-FOR LOOP1
/       WR-TEXT 'Primzahlen im Bereich von 2 bis &(N):'
/       SHOW-VAR PRIMARY-NUMBERS,INF=*PAR(NAME=*NONE)
/       EXIT-PROC
```

Beispiel 4

Verarbeiten einer Listenvariablen unter Mitführung des Index.

```
/DECL-VAR SPRACHEN,MULT-ELEM=*LIST
/SPRACHEN = 'Deutsch', WR-MODE=*EXTEND
/SPRACHEN = 'Englisch', WR-MODE=*EXTEND
/SPRACHEN = 'Franzoesisch', WR-MODE=*EXTEND
/SPRACHEN = 'Spanisch', WR-MODE=*EXTEND
/
/FOR I = *COUNTER(FROM=1,TO=SIZE('SPRACHEN'))
/    WR-TEXT '&(SPRACHEN#I) ist &(I). Sprache'
/END-FOR
```

Ausgabe

```
Deutsch ist 1. Sprache
Englisch ist 2. Sprache
Franzoesisch ist 3. Sprache
Spanisch ist 4. Sprache
```

FREE-VARIABLE
Inhalt einer Variablen löschen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando FREE-VARIABLE löscht den Inhalt einer oder mehrerer S-Variablen.

Lesezugriff auf die Variable ist nach FREE-VARIABLE nicht mehr möglich, da die Variable keinen gültigen Inhalt mehr hat. Die Deklaration der Variablen bleibt jedoch nach dem Löschen erhalten – mit Ausnahme von Variablen mit dynamischer Struktur.

Das Kommando FREE-VARIABLE kann auf einfache und zusammengesetzte Variable angewendet werden. Bei einer Listenvariablen können gezielt ein oder mehrere Elemente gelöscht werden. Bei einer Variablen mit dynamischer Struktur werden nicht nur die Inhalte der Elemente gelöscht, sondern auch die Elemente selbst.

Ist die S-Variable mit einem Variablenbehälter verknüpft, kann auch der Variablenbehälter gelöscht werden. Schreibzugriff auf die Variable ist dann nicht mehr möglich.

Format

FREE-VARIABLE

VARIABLE-**NAME** = <structured-name 1..20 with-wild(40)> / ***LIST**(...) /
list-poss(2000): <composed-name 1..255>

***LIST**(...)
LIST-NAME = <composed-name 1..255>
,**FROM-INDEX** = ***FIRST** / ***LAST** / <integer 1..214783647>
,**NUMBER-OF-ELEM**ENTS = **1** / ***REST** / <integer 1..214783647>

,**DESTROY-CONTAINER**-JV = ***NO** / ***Y**ES

Operandenbeschreibung

VARIABLE-NAME =
Gibt die Variablen an, deren Inhalt gelöscht werden soll.

VARIABLE-NAME = <structured-name 1..20 with-wild(40)>
Löscht den Inhalt der Variablen, deren Namen dem angegebenen Suchmuster entsprechen.

VARIABLE-NAME = *LIST(...)
Löscht den Inhalt von Elementen einer Listenvariablen.

LIST-NAME = <composed-name 1..255>
Name der Listenvariablen.

FROM-INDEX = *FIRST / *LAST / <integer 1..2147483647>
Index des Elementes der Listenvariablen, mit dem beginnend eine spezifizierte Anzahl von Listenelementen gelöscht werden.
*FIRST: Der Löschvorgang beginnt mit dem ersten Element der Liste; Voreinstellung.
Die Angabe *LAST löscht genau das letzte Element der Liste. Der Operand NUMBER-OF-ELEMENTS wird in diesem Fall ignoriert.

NUMBER-OF-ELEMENTS = 1 / *REST / <integer 1..2147483647>
Anzahl der Listenelemente, die gelöscht werden sollen.
Voreinstellung: Ein Element wird gelöscht.
Die Angabe *REST löscht alle Elemente vom angegebenen Startelement (Operand FROM-INDEX) bis zum letzten Element der Liste.

VARIABLE-NAME = list-poss(2000): <composed-name 1..255>
Löscht den Inhalt der einfachen oder zusammengesetzten Variablen, die in der angegebenen Namensliste enthalten sind.

DESTROY-CONTAINER-JV = *NO / *YES
Es kann vereinbart werden, ob der Variablenbehälter gelöscht werden soll, der mit der Variablen verknüpft ist. (Wenn z.B. eine S-Variable mit einer Jobvariablen als Variablenbehälter verknüpft ist, wird die Jobvariable bei DESTROY-CONTAINER-JV = *YES aus dem Dateikatalog gelöscht. Auf die S-Variable ist dann kein schreibender Zugriff mehr möglich.)

Ist die Variable nicht mit einem Variablenbehälter verknüpft, so wird der Operand ignoriert.

Regeln für das Löschen von Listen- und anderen zusammengesetzten Variablen

– Bezeichnet ein Name eine zusammengesetzte Variable, so werden die Inhalte aller Variablenelemente gelöscht. Die Attribute der zusammengesetzten Variablen (TYPE, SCOPE,...) bleiben jedoch gültig. Neu angelegte Elemente dieser Variablen müssen der ursprünglichen Deklaration entsprechen.

Beispiel

```
/DECLARE-VARIABLE L(TYPE=*STRING),MULTIPLE-ELEMENT=*LIST
/L='ELEM1',WRITE-MODE=*EXTEND
/L='ELEM2',WRITE-MODE=*EXTEND
/FREE-VARIABLE L     &* all elements deleted but declaration remains
/SHOW-VARIABLE L     &* Nothing displayed
/L=3,WRITE-MODE=*EXTEND
%  SDP1036 VARIABLE TYPE AND VALUE TYPE DO NOT MATCH
```

– Bezeichnet der Name eine dynamische Struktur, so werden alle Elementdeklarationen gelöscht und entfernt.

Beispiel

```
/DECLARE-VARIABLE S(TYPE=*STRUCTURE(DEFINITION=*DYNAMIC))
/S.ELEM1=1; S.ELEM2='ELEM2'; S.ELEM3 =TRUE;
/FREE-VARIABLE S     &* all element declarations are deleted
/S.ELEM1='ELEM1'     &* no type retained, so modification possible
```

– Sind die Elemente einer zusammengesetzten Variablen dynamische Strukturen, bleibt die Information „Element hat TYPE = *STRUCTURE(*DYNAMIC)“ erhalten (die Elemente der dynamischen Struktur selbst werden gelöscht und entfernt).
Falls die Elemente statische Strukturen sind, bleibt aber die Deklaration des Struktur-Layouts gültig. Entsprechendes gilt für Listen.

Beispiel

```
/DCV L1(TYPE=*STRUCTURE(*DYNAMIC)),MULTIPLE-ELEMENT=*LIST
/L1#.ELEM1=1; L1#.ELEM2='ELEM2'; L1#.ELEM3=TRUE
/FREE-VARIABLE L1#
/L1#.ELEM1='ELEM1'    &* no type retained, so modification possible
/
/BEGIN-STRUCTURE S
/DECLARE-ELEMENT ELEM1(TYPE=*STRING)
/DECLARE-ELEMENT ELEM2(TYPE=*INTEGER)
/END-STRUCTURE
/DCV L2(TYPE=*STRUCTURE(S)),MULTIPLE-ELEMENT=*LIST
/L2#.ELEM1='ELEM1';L2#.ELEM2=2
/FREE-VARIABLE L2#     &* Elements deleted but declaration remains
/L2#.ELEM1=1
%  SDP1036 VARIABLE TYPE AND VALUE TYPE DO NOT MATCH
```

– Ist die zusammengesetzte Variable, auf die das Kommando FREE-VARIABLE angewendet wird, Element einer dynamischen Struktur, werden alle ihre Elemente und die zusammengesetzte Variable selbst gelöscht. So können auch Arrays, Listenvariable oder statische Strukturen gelöscht werden, die explizit deklariert wurden. Sollen solche zusammengesetzten Variablen wieder mit den gleichen Eigenschaften entstehen, müssen sie erneut deklariert werden.

Beispiel

```
/DCV S(TYPE=*STRUCTURE(*DYNAMIC))
/S.L#.ELEM1=1; S.L#.ELEM2='ELEM2'; S.L#.ELEM3=TRUE;
/FREE-VARIABLE S.L
/SHOW-VARIABLE S.L
```

– Listenvariablen werden nach dem Löschen von Elementen wieder neu nummeriert, sodass immer eine lückenlose Nummerierung ab 1 besteht.

Beispiel

```
/DCV L1(TYPE=*INTEGER),MULTIPLE-ELEMENT=*LIST
/L1=1,W-M=*EXT; L1=2,W-M=*EXT; L1=3,W-M=*EXT; L1=4,W-M=*EXT;
/FREE-VARIABLE L1#2
/SHOW-VARIABLE L1,LIST-INDEX-NUMBER=*YES
L1#1 = 1
L1#2 = 3
L1#3 = 4

/DCV L2(TYPE=*INTEGER),MULTIPLE-ELEMENT=*LIST
/L2=*STRING-TO-VARIABLE('(1,2,3,4)')
/SHOW-VARIABLE L2,LIST-INDEX-NUMBER=*YES
L2#1 = 1
L2#2 = 2
L2#3 = 3
L2#4 = 4
/FREE-VARIABLE (L2#1,L2#2) &* Use *LIST syntax to free elem 1 / 2
/SHOW-VARIABLE L2,LIST-INDEX-NUMBER=*YES
L2#1 = 2
L2#2 = 4
```

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; keine Elemente gelöscht; Variable ist bereits gelöscht
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler garantierte Meldung: SDP1008
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiele

```
/DECLARE-VARIABLE B(100,*INTEGER), SCOPE=*PROCEDURE
/FREE-VARIABLE B
/B = 33
/FREE-VARIABLE B ————————————————————————————————————————————————  (1)
/SHOW-VARIABLE B ————————————————————————————————————————————————  (2)
```

(1) Das Kommando FREE-VARIABLE bewirkt, dass der Wert gelöscht wird; anschließend ist nur noch schreibender Zugriff auf die Variable B möglich.

(2) Keine Ausgabe, da die Variable keinen gültigen Wert mehr hat.

Löschen mit Angabe von Musterzeichen

```
/TSDP-023-003-0001-1 = ' ALPHA '
/TSDP-023-003-0001-2 = ' BETA  '
/TSDP-023-003-0001-3 = ' GAMMA '
/FREE-VAR T<R:T><A:E><N:Q>-02/-*-<1,3>
/SHOW-VARIABLE
TSDP-023-003-0001-2 =  BETA
*END-OF-CMD
```

Löschen mit Listenelementen

```
/DECLARE-VARIABLE T-LIST (TYPE= *STRING), MULTIPLE-ELEMENTS= *LIST
/                 T-LIST#1    = 'FIRST'
/                 T-LIST#2    = 'SECOND'
/                 T-LIST#3    = 'THIRD'
/                 T-LIST#4    = 'FOURTH'
/                 T-LIST#5    = 'ANTEPENULTIMATE'
/                 T-LIST#6    = 'PENULTIMATE'
/                 T-LIST#7    = 'LAST'
/FREE-VARIABLE *LIST (LIST-NAME=T-LIST,FROM-INDEX=*LAST) ————————————  (1)
/SHOW-VARIABLE T-LIST ———————————————————————————————————————————  (2)
T-LIST(*LIST) = FIRST
T-LIST(*LIST) = SECOND
T-LIST(*LIST) = THIRD
T-LIST(*LIST) = FOURTH
T-LIST(*LIST) = ANTEPENULTIMATE
T-LIST(*LIST) = PENULTIMATE
/FREE-VARIABLE *LIST(LIST-NAME=T-LIST,FROM-INDEX=4,NUMBER-OF-ELEM= 2 )    (3)
/SHOW-VARIABLE T-LIST ———————————————————————————————————————————  (4)
T-LIST(*LIST) = FIRST
T-LIST(*LIST) = SECOND
T-LIST(*LIST) = THIRD
T-LIST(*LIST) = PENULTIMATE
```

(1) Mit FROM-INDEX=*LAST im Kommando FREE-VARIABLE soll das letzte Element der Liste T-LIST gelöscht werden.

(2) Das Kommando SHOW-VARIABLE zeigt die noch verbleibenden 6 Elemente der Liste an.

(3) Ab dem 4. Element sollen 2 Elemente der Liste gelöscht werden.

(4) Das Kommando SHOW-VARIABLE zeigt die noch verbleibenden 4 Elemente der Liste an. Die zuvor vorhandenen Elemente 4 und 5 sind nicht mehr vorhanden.

GOTO
Zu einer Marke springen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

GOTO springt zu der angegebenen Marke und setzt dort den Prozedurlauf fort. Die Marke muss im gleichen oder in einem umgebenden Block definiert sein (siehe Beispiel 2). Außerdem muss sie innerhalb der Prozedur eindeutig sein. (Zu GOTO siehe auch Abschnitt „Beliebige Sprungziele“ auf Seite 105.)

Format

GOTO
LABEL = <structured-name 1..255>

Operandenbeschreibung

LABEL = <structured-name 1..255>
Name der Marke, bei der der Prozedurlauf fortgesetzt wird; der Name wird ohne abschließenden Doppelpunkt angegeben.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0139	Back Branch-Grenze erreicht
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1

```
:
:
:
/GOTO MARKE10
:
:
:
/MARKE10: CREATE-FILE ...
```

Beispiel 2

```
/LOOP1: WHILE (A < B)
/ADD1:      X = X + A
/LOOP2:     WHILE (X < Y)
/ADD2:          A = A + 1
/               GOTO ADD1               "Erlaubt, weil umgebende Schleife"
/           END-WHILE LOOP2
/           GOTO ADD2                   "Verboten, weil innere Schleife"
/       END-WHILE LOOP1
```

IF
IF-Block einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando IF leitet einen IF-Block, das heißt eine bedingte Kommandofolge ein: Wenn die Bedingung im IF-Kommando zutrifft, wird die auf das IF-Kommando folgende Kommandofolge durchlaufen. Andernfalls werden im aktuellen IF-Block weitere ELSE-IF- bzw. ELSE-Kommandos gesucht. Enthält der aktuelle IF-Block keine ELSE-IF- bzw. ELSE-Kommandos, wird der Prozedurlauf mit dem Kommando fortgesetzt, das auf das zugehörige END-IF-Kommando folgt (siehe Abschnitt „Verzweigungen definieren“ auf Seite 94).

Format

IF
CONDITION = <text 0..1800 with-low *bool-expr*>

Operandenbeschreibung

CONDITION = <text 0..1800 with-low *bool-expr*>
Logischer Ausdruck als Bedingung für das Durchlaufen der Kommandofolge zwischen IF- und ELSE-IF- oder ELSE-Kommando (logischer Ausdruck siehe Kapitel „Ausdrücke“ auf Seite 255). Falls in dem logischen Ausdruck ein einzelnes „=“ -Zeichen enthalten ist, muss dieser in Klammern eingeschlossen werden.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/A = 2
/B = 3
/IF (A = B) ————————————————————————————————————————————————————————  (1)
/WRITE-TEXT 'A UND B SIND RICHTIG INITIALISIERT'
/ELSE-IF (A > B) ———————————————————————————————————————————————————  (2)
/WRITE-TEXT 'A IST ZU GROSS'
/ELSE
/WRITE-TEXT 'B IST ZU GROSS'
/END-IF
B IST ZU GROSS
```

Anmerkung zur Klammerung der logischen Ausdrücke:

(1) Die Klammerung kann entfallen, wenn das „=“-Zeichen verdoppelt wird: `/IF  A==B`

(2) Die Klammerung kann entfallen: `/ELSE-IF  A > B`

IF-BLOCK-ERROR
Block-Fehlerbehandlung einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

IF-BLOCK-ERROR leitet eine Kommandofolge ein, die in folgenden Fällen durchlaufen wird:

– Im aktuellen Block ist ein Fehler aufgetreten.
– In einem Block, der in den aktuellen Block geschachtelt ist, ist ein Fehler aufgetreten, der vor dem Verlassen dieses eingeschlossenen Blocks nicht abgefangen wurde.

Im IF-BLOCK-ERROR-Block kann mit dem Kommando ELSE ein ELSE-Zweig definiert werden. Abgeschlossen wird der IF-BLOCK-ERROR-Block mit dem Kommando END-IF.

Wird das Kommando IF-BLOCK-ERROR aufgerufen, obwohl kein Fehler auftrat, wird der ELSE-Zweig durchlaufen - wenn vorhanden - oder die Kommandoausführung nach dem zugehörigen END-IF-Kommando fortgesetzt. Wenn der Kommando-Returncode im Nichtfehlerfall, im ELSE-Zweig oder nach dem END-IF-Kommando ausgewertet werden soll, ist ein SAVE-RETURNCODE-Kommando vor dem IF-BLOCK-ERROR-Kommando erforderlich.

Format

IF-BLOCK-ERROR

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/BL1: BEGIN-BLOCK
/...
/          BL2: BEGIN-BLOCK
/          ...
/          "Hier tritt Fehler1 auf"
/          ...
/          END-BLOCK BLOCK = BL2
/...
/"Hier tritt Fehler2 auf"
/...
/IF-BLOCK-ERROR ... "Hier wird der Fehler abgefangen"
/...
/END-IF
/END-BLOCK BLOCK = BL1
```

Der Block BL2 ist in den Block BL1 geschachtelt. Im Block BL2 erfolgt keine Fehlerbehandlung. Fehler, die im BLock BL2 auftreten, werden im IF-BLOCK-ERROR-Block von Block BL1 abgefangen, genauso wie Fehler, die im Block BL1 auftreten.

IF-CMD-ERROR
Kommando-Fehlerbehandlung einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

IF-CMD-ERROR leitet eine Kommandofolge ein, die durchlaufen wird, wenn im unmittelbar vorhergehenden Kommando ein Fehler auftritt. Hiermit ist eine gezielte Fehlerbehandlung für dieses Kommando möglich und eine Blockfehlerbehandlung wird somit verhindert.

IF-CMD-ERROR hat nach folgenden Kommandos keine Wirkung:

– IF / END-IF
– FOR
– WHILE
– REPEAT
– BEGIN-BLOCK
– GOTO
– CYCLE
– EXIT-BLOCK

Im IF-CMD-ERROR-Block kann mit dem Kommando ELSE ein ELSE-Zweig definiert werden. Zusätzlich wird implizit ein SAVE-RETURNCODE-Kommando im ELSE-Zweig ausgeführt, sodass der aktuelle Returncode des Kommandos auch zugreifbar ist, wenn das Kommando ohne Fehler ausgeführt wurde. Abgeschlossen wird der IF-CMD-ERROR-Block mit dem Kommando END-IF.

Wird IF-CMD-ERROR aufgerufen, ohne dass ein Fehler auftrat, wird der ELSE-Zweig durchlaufen – wenn vorhanden – oder die Kommandoausführung nach dem zugehörigen END-IF fortgesetzt.

Format

IF-CMD-ERROR

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Abschnitt „Fehlerbehandlung“ auf Seite 69.

Hinweis

Das dem IF-CMD-ERROR vorangehende Kommando und der IF-CMD-ERROR-Block bilden quasi einen BEGIN-Block.

IMPORT-VARIABLE
Variable importieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando IMPORT-VARIABLE kann eine zuvor deklarierte Variablen in die gerufene Prozedur importiert werden. IMPORT-VARIABLE entspricht so in gewisser Weise dem Kommando DECLARE-VARIABLE, erspart aber die wiederholte Aufzählung aller zugewiesenen Attribute.

Format

IMPORT-**VAR**IABLE

VARIABLE-**NAME** = <structured-name 1..20 with-wild(40)> / list-poss(2000): <structured-name 1..20>

,**FROM** = ***SCOPE**(...)

***SCOPE**(...)

SCOPE = ***TASK** / ***CALL**ING-**PROC**EDURES

Operandenbeschreibung

VARIABLE-NAME =
Name einer Variablen außerhalb der Prozedur, die in die aktuelle Prozedur importiert werden soll.

VARIABLE-NAME = <structured-name 1..20 with-wild(40)>
Name einer Variablen außerhalb der Prozedur, die in die aktuelle Prozedur importiert werden soll.
Wenn der Name Musterzeichen enthält, werden alle Variablen importiert, deren Namen dem angegebenen Suchmuster entsprechen. Falls eine Musterzeichenfolge keinem Variablennamen entspricht, wird die Meldung SPD0519 ausgegeben.

VARIABLE-NAME = list-poss(2000): <structured-name 1..20>
Ein oder mehrere Namen von Variablen, die in die aktuelle Prozedur importiert werden sollen. Bei Listenangabe werden die Variablen in der angegebenen Reihenfolge importiert.

FROM = *SCOPE(...)
Bezeichnet den Geltungsbereich der zu importierenden Variablen..

SCOPE = *TASK
Die Variable wird im Geltungsbereich Task gesucht und importiert.

SCOPE = *CALLING-PROCEDURES
Die Variable wird in der gerufenen Prozedur gesucht. Sie muss dort mit IMPORT-ALLOWED=*YES deklariert sein. Der Suchvorgang verläuft in aufsteigender Richtung von der Aufruf-Prozedur bis zur ersten Prozedur (bei einer Hintergrund-Prozedur) oder bis zur Dialogebene (bei einer Vordergrund-Prozedur) gesucht.
(Wenn die zu importierende Variable schon in der Prozedur vorhanden (d.h. sichtbar) ist – egal ob mit IMPORT-ALLOWED = *YES oder *NO deklariert –, passiert nichts).

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; Element schon deklariert
2	0	SDP2000	Warnung: Nicht alle Elemente der Eingabeliste konnten erfolgreich-abgearbeitet werden. garantierte Meldung: SDP2000
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler garantierte Meldungen: SDP1008, SDP1018
	64	SDP2001	Keines der Elemente konnte importiert werden
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiele

```
/DECLARE-VARIABLE VERBOSE-MODE(TYPE=*BOOLEAN, INITIAL-VALUE= YES),-
/                  SCOPE=*CURRENT(IMPORT-ALLOWED=*YES)
/LEVEL=1
/CALL-PROCEDURE MY-PROCEDURE
/           ---> IMPORT-VARIABLE VERBOSE-MODE, -
/                FROM=*SCOPE(*CALLING-PROCEDURES)
/
/                LEVEL=2
/                IF (VERBOSE-MODE)
/                   WRITE-TEXT 'LAUFENDE PROZEDUR BEI LEVEL &(LEVEL).'
/                END-IF
/            ---> EXIT-PROCEDURE
/IF (VERBOSE-MODE)
/    WRITE-TEXT 'LAUFENDE PROZEDUR BEI LEVEL &(LEVEL).'
/END-IF
```

Die aufgerufene Prozedur „MY-PROCEDURE“ hat Zugriff zur Variablen „VERBOSE-MODE“ durch die rufende Prozedur auf Grund der Benutzung von IMPORT-VARIABLE. Jede Prozedur hat dabei eine lokale Variable mit Namen „LEVEL“. Siehe Abschnitt „Geltungsbereich von Variablen“ auf Seite 162.

```
/DECLARE-VARIABLE TVAR-1 (TYPE= *STRING, INIT-VAL='TV1')
/DECLARE-VARIABLE TVAR-2 (TYPE= *STRING, INIT-VAL='TV2')
/DECLARE-VARIABLE TVAR-3 (TYPE= *STRING, INIT-VAL='TV3')
/CALL-PROCEDURE IMPORT-TV-LIST ------------------------------------------  (1)
      --> /SET-PROCEDURE-OPTIONS
          /IMPORT-VARIABLE (TVAR-1, TVAR-2) -
          /      ,FROM=*SCOPE(SCOPE=*CALLING-PROCEDURE)
          /SHOW-VARIABLE
          /EXIT-PROCEDURE
TVAR-1 = TV1
TVAR-2 = TV2
*END-OF-CMD
/CALL-PROCEDURE IMPORT-TV-WILDCARD --------------------------------------  (2)
      --> /SET-PROCEDURE-OPTIONS
          /IMPORT-VARIABLE TVAR-<1,3> -
          /      ,FROM=*SCOPE(SCOPE=*CALLING-PROCEDURE)
          /SHOW-VARIABLE
          /EXIT-PROCEDURE
TVAR-1 = TV1
TVAR-3 = TV3
*END-OF-CMD
```

(1) Die Prozedur IMPORT-TV-LIST verwendet die Listenangabe um die Variablen TVAR-1 und TVAR-2 zu importieren.

(2) Die Prozedur IMPORT-TV-WILDCARD verwendet die Angabe von Musterzeichen um die Variablen TVAR-1 und TVAR-3 zu importieren.

INCLUDE-BLOCK
BEGIN-Block als Unterprozedur ausführen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando INCLUDE-BLOCK verzweigt zu dem Kommando BEGIN-BLOCK mit der angegebenen Marke (Namen). Nach Ausführung des (nächsten) Kommandos END-BLOCK oder EXIT-BLOCK erfolgt der Rücksprung zu dem Kommando, das dem Kommando INCLUDE-BLOCK folgt. (Zu BEGIN-Block siehe Abschnitt „Blockanfang als Sprungziel“ auf Seite 102).

Das Kommando INCLUDE-BLOCK ermöglicht die komfortable Ausführung einer Unterprozedur, die zwischen den Kommandos BEGIN-BLOCK und END-BLOCK definiert ist. In diesem Fall sind folgende Regeln zu beachten:

- /INCLUDE-BLOCK verzweigt nur zu einem /BEGIN-BLOCK mit einer Marke (Namen). Um zu anderen Kommandos mit einer Marke zu verzweigen, muss das Kommando GOTO benutzt werden.
- Der BEGIN-Block mit der Unterprozedur ist sinnvollerweise nach dem Kommando EXIT-PROCEDURE anzuordnen. Das vermeidet die ungewollte Ausführung während des Prozedurablaufs.
- Die Unterprozedur läuft in der gleichen Prozedurumgebung ab, wie die übergeordnete Prozedur (d.h. gleicher Variablenkontext, gleiche SYSFILE-Umgebung, usw.).
- Der rekursive Aufruf der Unterprozedur ist nicht erlaubt.
- Prozedurschachtelung ist erlaubt, sollte aber wegen Unübersichtlichkeit und unbeabsichtigtem Durchlaufen von BEGIN-Blöcken vermieden werden. Zu den Unterprozeduren kann direkt – in beliebiger Tiefe – verzweigt werden.
- Das Kommando GOTO kann in der Unterprozedur verwendet werden, wobei es aber nur zu Marken innerhalb der Unterprozedur verzweigen sollte, da sonst die Unterprozedur verlassen wird.
- Die bei dem Kommando BEGIN-BLOCK stehende Marke muss in der gesamten Prozedur eindeutig sein.

Das Kommando INCLUDE-BLOCK wird im Dialog-Modus abgewiesen.

Format

INCLUDE-BLOCK
BLOCK = <structured-name 1..255>

Operandenbeschreibung

BLOCK = <structured-name 1..255>
Name des BEGIN-Blocks, der als Unterprozedur ausgeführt werden soll.
Der Name ist ohne den abschließenden Doppelpunkt einzugeben.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)

Beispiel

```
/ASSIGN-SYSLST TO=PROT.EINGABE,SYSLST-NUMBER=1
/...
/IF (EING='*START')
/ INCLUDE-BLOCK INFO-1
/END-IF     &* Hier geht es nach Ausführung der Unterprozedur INFO-1 weiter
/...
/IF (EING='*END')
/ INCLUDE-BLOCK INFO-1
/END-IF     &* Hier geht es nach Ausführung der Unterprozedur INFO-1 weiter
/...
/...
/...
/INFO-1: BEGIN-BLOCK    &* Beginn der Unterprozedur INFO-1
/      WRITE-TEXT '&(TIME()): Es wurde &(EING) eingegeben',OUTPUT=*SYSLST(1)
/END-BLOCK &*Ende + Rücksprung zur Kommandozeile, die auf INCLUDE-BLOCK folgt
```

INCLUDE-CMD
Kommandofolgen aus Programm aufrufen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando INCLUDE-CMD können Kommandos und Kommandofolgen aus einem Programm heraus zur Ausführung übergeben werden. Das Kommando kann nur im Makro CMD (TU-Programm) und in EXECUTE-SYSTEM-CMD-Anweisungen ausgeführt werden.

Dabei ist zu beachten: Während der Ausführung von INCLUDE-CMD werden vom System einige Operationen wie Programm-Start, -Wiederaufnahme, und -Beendigung etc. zurückgewiesen, um mögliche Inkonsistenzen auszuschalten, da das Kommando im Programm-Modus aufgerufen wird (siehe auch unter Hinweise).

Die Ausgabe des Kommandos in SYSOUT ist in zwei Teile geteilt:

– Die Ausgabe als Ergebnis der Analyse des INCLUDE-CMD-Kommandos: Sie kann im SYSOUT-Puffer des CMD-Makros gesichert werden (Fehler-Report oder andere Informationsart)

– Die Ausgabe der Kommandoausführung, die in SYSOUT aktuell enthalten ist: Sie ist unabhängig vom Parameter des CMD-Aufrufs.
  Die Ausgabe dieser Kommandos nach SYSOUT kann aber durch das Kommando ASSIGN-SYSOUT beeinflusst werden, wenn sich dieses in der bei CMD=... angegebenen Kommandoliste befindet.

Format

INCLUDE-CMD	**Kurzname: INCMD**
CMD = <text 0..1800 with-low>	

Operandenbeschreibung

CMD = <text 0...1800 with-low>
Auszuführendes Kommando oder auszuführende Kommandofolge, getrennt durch Semikolons und insgesamt eingeschlossen in runden Klammern. Ein führendes Zeichen „/“ ist verboten.
Die Ausführung der Kommandoliste, die im Operanden CMD angegeben wird, ist mit der Ausführung einer aus diesen Kommandos bestehenden Include-Prozedur identisch, allerdings mit einigen Einschränkungen:

- Es ist nicht erlaubt, im Operanden CMD Prozedurkopf-Kommandos anzugeben. Die Kommandos SET-PROCEDURE-OPTIONS, BEGIN-PARAMETER-DECLARATION, END-PARAMETER-DECLARATION und DECLARE-PARAMETER werden deshalb zurückgewiesen.
- Für die Logging-Optionen gelten die Default-Einstellungen. Sie können mit MODIFY-PROCEDURE-OPTIONS geändert werden.
- Wenn in der Kommandoliste bei CMD (auf oberster Ebene) das Kommando EXIT-PROCEDURE angegeben wurde, dann endet INCLUDE-CMD mit diesem Kommando. Nachfolgende Kommandos werden ignoriert.
- Die Kommandos DO, ENDP, END-PROCEDURE, ENDP-RESUME können in der Kommandofolge im CMD-Operanden nicht angegeben werden.
- Wenn INCLUDE-CMD in einer Liste von Kommandoeingaben im Puffer des CMD-Makros angegeben ist, werden nachfolgende Kommandos ignoriert.
  Beispiel: CMD 'cmd1;cmd2;INCLUDE-CMD CMD=(PRINT-DOCUMENT X);cmd3'
  Hier wird die Eingabe „cmd3“ ignoriert.

Hinweise

- Die Kommandos, die im Operanden CMD angegeben werden, werden wie Eingaben in einer S-Prozedur ausgeführt, die mit INCLUDE-PROCEDURE aufgerufen wurde. Sie haben impliziten Zugriff zu den Variablen der gegenwärtigen Prozedurebene (genauso als wären sie mit INCLUDE-PROCEDURE aufgerufen worden und als würden sie die Systemdateien der gegenwärtigen Prozedurebene beerben (SYSTEM-FILE-CONTEXT=*STD).
- Im Gegensatz zu INCLUDE-PROCEDURE beendet INCLUDE-CMD ein Programm nicht, wenn es durch den Makro CMD aufgerufen wurde. Das Programm wird auch nicht beendet, wenn eine Prozedur in der Kommandofolge aufgrufen wird, die im Operanden CMD angegeben ist (siehe Beispiel).
- INCLUDE-CMD darf im CMD-Makro (TU-Programm) und in EXECUTE-SYTEM-CMD-Anweisungen ausgeführt werden.

– Während der Ausführung von INCLUDE-CMD werden vom System folgende Operationen zurückgewiesen, um mögliche Inkonsistenzen beim Aufruf im Programm-Modus auszuschalten:
  – Programm-Start und Programm-Beendigung: START-Hilfsprogramm, LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM), RESTART-PROGRAM (vgl. CALL-PROCEDURE ...UNLOAD-ALLOWED=*NO).
  – Programm-Wiederaufnahme: AID-Kommandos, RESUME-PROGRAM, EXIT-PROCEDURE mit RESUME-PROGRAM=*YES, ENDP-RESUME, INFORM-PROGRAM, SEND-MSG mit TO=*PROGRAM
  – Prozedur-Abbruch: CANCEL-PROCEDURE, K2 beim Prompting von Parametern.
  – Rekursiver Aufruf von INCLUDE-CMD.
  – BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD
  – SET-JOB-STEP, wenn ein Programm geladen ist.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0138	Fehler während der Prozedur-Voranalyse
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
CMD 'INCLUDE-CMD CMD=(DECLARE-VARIABLE A;
                      CALL-PROCEDURE MYPROC,(RET=A);
                      IF(A = ''OKAY'');
                      WRITE-TEXT ''SUCCESSFULL''
                      ELSE
                      WRITE-TEXT ''ERROR''
                      END-IF)'
"ZURÜCK IN DEN PROGRAMM-MODUS"
```

INCLUDE-PROCEDURE
Kommandofolge als Include-Prozedur starten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Eine Include-Prozedur ist eine Prozedur, in der die Datenumgebung der aufrufenden Prozedur sichtbar ist. Das heißt, alle Variablen sind sichtbar, die in der aufrufenden Prozedur sichtbar sind. Allerdings kann eine Variable der aufrufenden Prozedur durch eine neue Deklaration in der Include-Prozedur überdeckt werden (siehe Beispiel).

Das Kommando INCLUDE-PROCEDURE startet eine gespeicherte Kommandofolge (Prozedur). Bei Abarbeitung werden darin enthaltene symbolische Parameter durch die im Kommandoaufruf angegebenen Werte (Operand PROCEDURE-PARAMETERS) ersetzt:

– Als Aktualparameter können Variablen übergeben werden, über die die Prozedur auch Ausgabewerte zurückliefert.
– Die Aktualparameter können als Stellungs- oder als Schlüsselwortparameter übergeben werden. Die Reihenfolge der Stellungsparameter entspricht dabei der dynamischen Reihenfolge der DECLARE-PARAMETER-Kommandos; die Namen der Schlüsselwörter entsprechen den Namen der formalen Prozedurparameter. Die Schlüsselwörter dürfen bis zur Eindeutigkeit abgekürzt werden.
– Die Protokolierung wird im Kommandoaufruf eingestellt; ebenso, ob ein bereits geladenes Programm entladen werden darf.

Prozeduren können gespeichert werden als:

– katalogisierte (auch temporäre) SAM- oder ISAM-Datei mit Sätzen variabler Länge
– Element vom Typ J oder SYSJ einer PLAM-Bibliothek
– S-Variable vom Typ Liste

Prozedur-Formate:

– Text-Prozedur
  Die S-Prozedur liegt im ursprünglichen Textformat vor. Volle SDF-P-Funktionalität kann nur verwendet werden, wenn bei Aufruf der Prozedur das kostenpflichtige Subsystem SDF-P geladen ist. In Bibliotheken sollte für Text-Prozeduren der Elementtyp J verwendet werden.
– Objekt-Prozedur
  Die S-Prozedur im Textformat wurde mit dem Kommando COMPILE-PROCEDURE in ein Zwischenformat kompiliert. Diese Objekt-Prozedur kann, unabhängig von der Verfügbarkeit des Subsystems SDF-P, die volle SDF-P-Funktionalität nutzen (mit Ausnahme des Kommandos COMPILE-PROCEDURE). In Bibliotheken sollte für Objekt-Prozeduren der Elementtyp SYSJ verwendet werden (Default-Wert bei COMPILE-PROCEDURE).

Format

INCLUDE-**PROC**EDURE	Kurzname: **INP**

```
FROM-FILE = <filename 1..54 without-gen> / *LIBRARY-ELEMENT(...) / *VARIABLE(...)
   *LIBRARY-ELEMENT(...)
      |  LIBRARY = <filename 1..54 without-gen>
      |  ,ELEMENT = <composed-name 1..64>(...)
      |     <composed-name 1..64>(...)
      |        |  VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
      |  ,TYPE = *STD / *BY-LATEST-MODIFICATION / <alphanum-name 1..8>
   *VARIABLE(...)
      |  VARIABLE-NAME = <composed-name 1..255>
,PROCEDURE-PARAMETERS = *NO / <text 0..1800 with-low expr>
,LOGGING = *PARAMETERS(...) / YES / *NO /
   *PARAMETERS(...)
      |  CMD = *BY-PROC-TEST-OPTION / *YES / *NO
      |  ,DATA = *BY-PROC-TEST-OPTION / *YES / *NO
,UNLOAD-ALLOWED = *YES / *NO
,EXECUTION = *YES / *NO
```

Operandenbeschreibung

FROM-FILE = <filename 1..54 without-gen> / *LIBRARY-ELEMENT(...) / *VARIABLE(...)
Name der Prozedurdatei.

FROM-FILE = *LIBRARY-ELEMENT(...)
Die Prozedur ist in einem PLAM-Bibliothekselement abgelegt.

LIBRARY = <filename 1..54 without-gen>
Name der PLAM-Bibliothek, die die Prozedurdatei als Element (Typ J oder SYSJ; siehe Operand TYPE) enthält.

ELEMENT = <composed-name 1..64>(...)
Name des Elements.

VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
Version des Bibliothekelements. Voreingestellt ist *HIGHEST-EXISTING, d.h. die Prozedur wird dem Element mit der höchsten Version entnommen.

TYPE = *STD / *BY-LATEST-MODIFICATION / <alphanum-name 1..8>
Bestimmt den Elementtyp, unter dem die Prozedurdatei in der PLAM-Bibliothek abgelegt ist.

TYPE = *STD
Die Prozedurdatei kann als Element des Typs SYSJ oder J abgelegt sein.
Das angegebene Element wird zuerst unter den Elementen vom Typ SYSJ gesucht. Falls es dort nicht existiert, wird unter den Elementen vom Typ J weitergesucht.

Eine Nicht-S-Prozedur kann nur als Element vom Typ J vorliegen.
Eine S-Prozedur kann sowohl als Text-Prozedur (ursprüngliches Textformat) als auch als Objekt-Prozedur (kompiliertes Zwischenformat) vorliegen. Zur Vereinfachung der Verwaltung beider Formate in einer Bibliothek sollten Text-Prozeduren als Element vom Typ J und Objekt-Prozeduren als Element vom Typ SYSJ ablegt sein. Mit dem Kommando COMPILE-PROCEDURE wird aus einer Text-Prozedur vom Typ J standardmäßig eine Objekt-Prozedur vom Typ SYSJ (Default-Wert) erzeugt.
Die Angabe von TYPE=*STD (Default-Wert) stellt sicher, dass bei Einhaltung dieser Konvention Objekt-Prozeduren bevorzugt aufgerufen werden.

TYPE = *BY-LATEST-MODIFICATION
Die Prozedurdatei kann als Element des Typs SYSJ oder J abgelegt sein.
Existiert das angegebene Element sowohl als Typ SYSJ als auch J, wird das zuletzt geänderte Element aufgerufen. Bei gleichem Zeitstempel wird des Element vom Typ SYSJ aufgerufen.
Die Angabe von TYPE=*BY-LATEST-MODIFICATION stellt sicher, dass z.B. während der Testphase bei der Erstellung bzw. Erweiterung einer Prozedur das aktuellste Element aufgerufen wird.

TYPE = <alphanum-name 1..8>
Die Prozedurdatei wird ausschließlich unter den Elementen des angegebenen Typs gesucht.

FROM-FILE = *VARIABLE(...)
Die Prozedur ist in einer S-Variablen vom Typ Liste abgelegt.

VARIABLE-NAME = <composed-name 1..255>
Name der S-Variablen.

PROCEDURE-PARAMETERS = *NO / <text 0..1800 with-low *expr*>
Definiert die aktuellen Prozedurparameter; die Parameter müssen in Klammern eingeschlossen werden.
Zu Prozedurparametern siehe auch Abschnitt „Prozedurparameter übergeben“ auf Seite 108ff.

LOGGING = *PARAMETERS(...) / *YES / *NO
Steuert die Protokollierung des Prozedurablaufs.
Der Operand LOGGING wird bei Aufruf von *Nicht-S-Prozeduren* ignoriert, da die Protokollierung dort nur im Prozedurkopf vereinbart werden kann (siehe BEGIN-PROCEDURE, Operand LOGGING).
Bei Protokollierung einer S-Prozedur wird jede abgearbeitete Prozedurzeile mit vorangestellter Zeilennummer und Prozedurstufe ausgegeben.

Zu Protokollierung siehe auch Abschnitt „Protokollierung einstellen“ auf Seite 84ff.

LOGGING = *PARAMETERS(...)
Die Protokollierung kann getrennt eingestellt werden für Kommando-/Anweisungszeilen und Datenzeilen.

CMD = *BY-PROC-TEST-OPTION / *YES / *NO
Gibt an, ob Kommandos protokolliert werden sollen. Voreingestellt ist BY-PROC-TEST-OPTION, d.h. keine Protokollierung (entspricht *NO) bzw. der Wert, den der Benutzer mit dem Kommando MODIFY-PROC-TEST-OPTIONS als Voreinstellung gewählt hat.

DATA = *BY-PROC-TEST-OPTION / *YES / *NO
Gibt an, ob Datenzeilen protokolliert werden sollen. Voreingestellt ist BY-PROC-TEST-OPTION, d.h. keine Protokollierung (entspricht NO) bzw. der Wert, den der Benutzer mit dem Kommando MODIFY-PROC-TEST-OPTIONS als Voreinstellung gewählt hat

UNLOAD-ALLOWED = *YES / *NO
Legt fest, ob ein Programm, das zum Zeitpunkt des Prozeduraufrufs geladen ist, entladen werden darf.
Der Schutz vor Programmentladung ist nur für das Entladen mit den Kommandos START-EXECUTABLE-PROGRAM, LOAD-EXECUTABLE-PROGRAM und CANCEL-PROGRAM gewährleistet.
Die Angabe YES wird ignoriert, wenn der Prozeduraufruf aus einer Prozedur erfolgt, für die UNLOAD-ALLOWED=*NO vereinbart wurde.

EXECUTION = *YES / *NO
Legt fest, ob die Prozedur nur zu Testzwecken analysiert werden oder auch ausgeführt werden soll.
Der Test ist mit MODIFY-SDF-OPTIONS (Operand MODE) möglich.

Kommando-Returncode

Die nachfolgenden Kommando-Returncodes können nur zurückgegeben werden, wenn die aufgerufene Prozedur selbst keinen Kommando-Returncode liefert (z.B. EXIT-PROCEDURE wegen Fehlers nicht ausgeführt).

Kommando-Returncodes, deren Maincode mit „SSM“ beginnt, können nur bei Aufruf einer Nicht-S-Prozedur zurückgegeben werden.

Kommando-Returncodes, deren Maincode mit „SDP“ beginnt, können nur bei Aufruf einer S-Prozedur zurückgegeben werden.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
2	0	SSM2058	Protokoll-Typ-Fehler
2	0	SSM2065	EOF auf Prozedurdatei, /END-PROC simuliert
	1	SSM2036	Unvollständiger Operand
	1	SSM2054	Symbolischer Operandenfehler
	1	SSM2055	Symbolischer Operandenfehler in /BEGIN-PROC
	1	SDP0138	Fehler bei Voranalyse der Text-Prozedur oder Objekt-Prozedur fehlerhaft
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0093	Nicht-S-Prozedur kann nur Element vom Typ J sein
	64	SDP0144	Fehler bei Parameterübertragung
	64	SSM2052	DVS-Fehler (Open-Fehler)
	64	SSM2053	keine SAM/ISAM-Datei oder Datei beginnt nicht mit /BEGIN-PROC bzw. /PROC
	64	SSM2056	Parameter von /INCL-PROC und /BEGIN-PROC passen nicht zusammen
	64	SSM2061	Fehler bei Zugriff auf Bibilothekselement
	64	SSM2064	Prozedurdatei kann nicht von entferntem Rechner geholt werden
	130	SDP0099	Kein Adressraum mehr verfügbar
xx	xx	xxxxxxx	sonstige Returncodes der aufgerufenen Prozedur

Beispiel

Prozedur 1:

```
/DECLARE-VARIABLE A(TYPE = *STRING)
/DECLARE-VARIABLE B(TYPE = *STRING)
/DECLARE-VARIABLE C(TYPE = *STRING)
/DECLARE-VARIABLE D
/INCLUDE-PROCEDURE PROC2
```

Prozedur PROC2:

```
/DECLARE-VARIABLE NAME=A(INIT-VALUE=1,TYPE=*INTEGER),SCOPE =*CURRENT  ---  (1)
/DECLARE-VARIABLE NAME=C(TYPE=*STRING),SCOPE = *PROCEDURE  ------------  (2)
/DECLARE-VARIABLE NAME=B(TYPE=*INTEGER),SCOPE = *PROCEDURE  ------------  (3)
```

(1) In PROC2 wird eine Variable A deklariert, die nur in PROC2 gültig ist. Die Variable A der aufrufenden Prozedur (Prozedur 1) ist daher in PROC2 nicht sichtbar, sondern erst wieder nach Beendigung von PROC2.

(2) Dies ist eine zulässige Mehrfachdeklaration. Sie ist ohne Wirkung. Die Prozedurvariablen C und D aus Prozedur 1 sind in PROC2 sichtbar. Die Prozedurvariable A aus Prozedur 1 wird durch die Variable A aus PROC2 überdeckt.

(3) FEHLER: Die dritte Deklaration führt zu einem Fehler, weil es bereits eine Prozedurvariable B mit anderem Datentyp gibt. In einer Mehrfachdeklaration müssen alle Attribute mit der ursprünglichen Deklaration übereinstimmen.

MODIFY-PROCEDURE-OPTIONS
Prozedureigenschaften während des Prozedurlaufs ändern

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit MODIFY-PROCEDURE-OPTIONS können die meisten Prozedureigenschaften, die zu Prozedurbeginn mit dem Kommando SET-PROCEDURE-OPTIONS festgelegt wurden, während des Prozedurlaufs geändert werden.

MODIFY-PROCEDURE-OPTIONS darf nicht aufgerufen werden, wenn der Prozedurablauf unterbrochen ist.

Wird MODIFY-PROCEDURE-OPTIONS innerhalb einer Include-Prozedur aufgerufen, wirkt es sich nur auf diese Include-Prozedur aus, das heißt, Änderungen werden nicht in die aufrufende Prozedur übernommen.

Format

MODIFY-**PROC**EDURE-**OPT**IONS

IMPLICIT-**DECL**ARATION = ***UNCHA**NGED / ***Y**ES / ***NO**

,**LOG**GING-**ALLOW**ED = ***PAR**AMETERS(...) / ***NO** / ***Y**ES

***PAR**AMETERS(...)

CMD = ***UNCHA**NGED / ***Y**ES / ***NO**

,**DATA** = ***UNCHA**NGED / ***Y**ES / ***NO**

,**INTERRUPT-ALLOW**ED = ***UNCHA**NGED / ***Y**ES / ***NO**

,**DATA-ESC**APE-**CHAR** = ***UNCHA**NGED / ***NONE** / '&&' / '#' / '*' / '@' / '$' / ***STD**

,**DATA-ERROR**-HANDLING = ***UNCHA**NGED / ***Y**ES / ***NO**

,**JV-REPL**ACEMENT = ***UNCHA**NGED / ***NONE** / ***AFTER-BUILTIN-FUNCTION**

,**ERROR-MECH**ANISM = ***UNCHA**NGED / ***SPIN-OFF-COMPATIBLE** / ***BY-RETURNCODE**

,**SUP**PRESS-**SDP-MSG** = ***UNCHA**NGED / ***NONE** / ***ADD(**...) / ***REM**OVE**(**...)

***ADD**(...)

MSG-ID = list-poss(2000): <alphanum-name 7..7>)

***REM**OVE(...)

MSG-ID = list-poss(2000): <alphanum-name 7..7>)

Operandenbeschreibung

IMPLICIT-DECLARATION = *UNCHANGED / *YES / *NO
Gibt an, ob implizite Deklarationen erlaubt sind.
Bei UNCHANGED wird die bisherige Vereinbarung unverändert übernommen.
IMPLICIT-DECLARATION kann im Dialog angegeben werden.

LOGGING-ALLOWED =
Legt fest, ob die Protokollierung der Prozedur erlaubt ist.

LOGGING-ALLOWED = *PARAMETERS(...)
Legt in den nachfolgenden Angaben fest, was protokolliert werden darf.

CMD = *UNCHANGED / *YES / *NO
Gibt an, ob Kommandos protokolliert werden dürfen.
Bei *UNCHANGED wird die bisherige Vereinbarung unverändert übernommen.

DATA = *UNCHANGED / *YES / *NO
Gibt an, ob Daten protokolliert werden dürfen.
Bei *UNCHANGED wird die bisherige Vereinbarung unverändert übernommen.

LOGGING-ALLOWED = *YES
Kommandos und Daten dürfen protokolliert werden.

LOGGING-ALLOWED = *NO
Protokollierung ist nicht erlaubt.

INTERRUPT-ALLOWED = *UNCHANGED
Die bisherige Vereinbarung soll unverändert übernommen werden.

INTERRUPT-ALLOWED = *YES
Vereinbart, dass die Prozedur mit der Funktionstaste K2 unterbrochen und mit dem Kommando RESUME-PROCEDURE fortgesetzt werden darf.

INTERRUPT-ALLOWED = *NO
Vereinbart, dass die Prozedur nicht mit der Funktionstaste K2 unterbrochen werden kann.

DATA-ESCAPE-CHAR =
Legt fest, welches Zeichen als Escape-Zeichen gilt. Das Escape-Zeichen ist das Zeichen, das die zu ersetzenden Ausdrücke in Datensätzen kennzeichnet.

DATA-ESCAPE-CHAR = *UNCHANGED
Die bisherige Vereinbarung soll unverändert übernommen werden.

DATA-ESCAPE-CHAR = *NONE
In Datensätzen soll keine Ausdrucksersetzung stattfinden.

DATA-ESCAPE-CHAR = '&&' / '#' / '*' / '@' / '$'
Legt das Escape-Zeichen fest.

DATA-ESCAPE-CHAR = *STD
Als Escape-Zeichen soll das Zeichen '&' gelten.

DATA-ERROR-HANDLING = *UNCHANGED
Die bisherige Vereinbarung soll unverändert übernommen werden.

DATA-ERROR-HANDLING = *YES
Vereinbart, dass in folgenden Fällen die Fehlerbehandlung ausgelöst wird:

- wenn eine Prozedurzeile Daten enthält, wo Kommandos erwartet werden
- wenn in Datenzeilen eine geforderte Ausdrucksersetzung nicht durchgeführt werden kann
- wenn ein Prozedursatz ein einzelnes Escape-Zeichen enthält

DATA-ERROR-HANDLING = *NO
Es wird keine Fehlerbehandlung ausgelöst; &varname bleibt in Daten unverändert stehen, falls varname weder als Funktion noch als Variable bekannt ist.

JV-REPLACEMENT = *UNCHANGED / *NONE / *AFTER-BUILTIN-FUNCTION
Gibt an, ob bei der Ausdrucksersetzung auch eine Jobvariablenersetzung stattfinden kann.

JV-REPLACEMENT = *UNCHANGED
Die bisherige Vereinbarung soll unverändert übernommen werden.

JV-REPLACEMENT = *NONE
Bei der Ausdrucksersetzung werden Namen nicht als Jobvariablen-Namen interpretiert.

JV-REPLACEMENT = *AFTER-BUILTIN-FUNCTION
Bei einem Ausdruck der Form &(name) wird name als Jobvariablen-Name interpretiert, wenn es keine Variable oder Built-in-Funktion dieses Namens gibt. Dieser Operandenwert soll ein zu Nicht-S-Prozeduren kompatibles Verhalten bei der Ausdrucksersetzung ermöglichen.

ERROR-MECHANISM =
Gibt an, ob die Fehlerbehandlung kompatibel zum Spin-Off-Verhalten von Nicht-S-Prozeduren ausgelöst oder ob Subcode1 ungleich Null berücksichtigt wird. Die Einstellung ist für die Fehlerbehandlung von Anweisungen wirkungslos.

ERROR-MECHANISM = *UNCHANGED
Die bisherige Vereinbarung soll unverändert übernommen werden.

ERROR-MECHANISM = *SPIN-OFF-COMPATIBLE
Die Fehlerbehandlung wird kompatibel zum bisherigen Spin-Off-Verhalten ausgelöst. Der Subcode1 wird **nicht** berücksichtigt. (Damit wird sichergestellt, dass das Fehlerverhalten von S-Prozeduren, die bereits in BS2000 V10.0 erstellt wurden, kompatibel bleibt.)

ERROR-MECHANISM = *BY-RETURNCODE
Die Fehlerbehandlung wird ausgelöst, wenn der Subcode1 des letzten Kommando-Returncodes ungleich Null ist. Das Spin-Off-Verhalten wird nicht berücksichtigt. Bei *BY-RETURNCODE muss die Fehlerbehandlung in S-Prozeduren auf die Kommando-Return-codes der Kommandos abgestimmt werden.

Hinweis
- Um sich vor späteren Änderungen in der Anwender-Syntaxdatei bezüglich der Voreinstellung zu schützen, sollte der gewählte Wert explizit in der Prozedur angegeben werden.
- Die Operanden IMPLICIT-DECLARATION und JV-REPLACEMENT können auch im Dialog angegeben werden.
  Bei Beginn der Task gelten im Dialog folgende Einstellungen:
  IMPLICIT-DECLARATION = *YES
  JV-REPLACEMENT = *AFTER-BUILTIN-FUNCTION

SUPPRESS-SDP-MSG =
Bestimmt, ob die Ausgabe bestimmter SDF-P-Meldungen (Meldungsklasse SDP) unterdrückt werden soll. Die Option gilt nur in der aufrufenden Prozedur (wird nicht weitervererbt).

SUPPRESS-SDP-MSG = *UNCHANGED
Die vorhandene Einstellung (Kommando /SET-PROCEDURE-OPTIONS oder vorheriges Kommando /MODIFY-PROCEDURE-OPTIONS) wird nicht verändert.

SUPPRESS-SDP-MSG = *NONE
Alle SDF-P-Meldungen werden ausgegeben.

SUPPRESS-SDP-MSG = *ADD(...)
Menge der SDF-P-Meldungen, die nicht ausgegeben werden sollen (zusätzlich zu evtl. schon vorherigen Angaben).

MSG-ID = list-poss(2000): <alphanum-name 7..7>
Liste der Meldungsnummern (Meldungsklasse SDP).

SUPPRESS-SDP-MSG = *REMOVE(...)
Menge der (unterdrückten) SDF-P-Meldungen, die wieder ausgegeben werden sollen.

MSG-ID = list-poss(2000): <alphanum-name 7..7>
Liste der Meldungsnummern (Meldungsklasse SDP).

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/SET-PROCEDURE-OPTIONS, LOGGING-ALLOWED=*NO
...
/MODIFY-PROCEDURE-OPTIONS, LOGGING-ALLOWED=*YES
...
```

Ist am Anfang der Prozedur auf Grund der Einstellung in SET-PROCEDURE-OPTIONS Protokollierung nicht erlaubt, dürfen nach MODIFY-PROCEDURE-OPTIONS Kommandos und Daten protokolliert werden.

MODIFY-PROCEDURE-TEST-OPTIONS
Protokollierung ändern, Rückwärtssprünge begrenzen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando MODIFY-PROCEDURE-TEST-OPTIONS können für Testzwecke folgende Einstellungen zum Prozedurablauf geändert werden:

– Die Protokollierung von Kommandos und Daten. Das Einschalten ist nur möglich, wenn dies für die Prozedur erlaubt ist.
– Das Begrenzen von Rückwärtssprüngen. Dadurch werden z.B. Endlosschleifen verhindert.
– Das Simulieren von SDF-P-BASYS-Verhalten, falls die volle SDF-P-Funktionalität verfügbar ist. Diese Einstellung gilt auf Task-Ebene.

Bei Taskbeginn gelten folgende Einstellungen:

– CMD = *NO, DATA = *NO
– BACK-BRANCH-LIMIT = *NONE
– FUNCTIONALITY = *FULL

In Nicht-S-Prozeduren hat MODIFY-PROCEDURE-TEST-OPTIONS keine Wirkung; dort wird die Protokollierung mit dem Kommando BEGIN-PROCEDURE gesteuert.

Zu Protokollierung siehe Abschnitt „Protokollierung veranlassen“ auf Seite 113.

Format

MODIFY**-PROC**EDURE**-TEST-OPT**IONS

LOGGING = ***PAR**AMETERS(...) / ***Y**ES / ***NO** /

***PAR**AMETERS(...)

CMD = ***UNCHA**NGED / ***Y**ES / ***NO**

,**DATA** = ***UNCHA**NGED / ***Y**ES / ***NO**

,**BACK-BRANCH-LIMIT** = ***UNCHA**NGED / ***NONE** / <text 0..1800 with-low *arith-expr*>

,**FUNCTIONALITY** = ***UNCHA**NGED / ***FULL** / ***BASIC**

Operandenbeschreibung

LOGGING = *PARAMETERS(...)
Steuert die Protokollierung. Dieser Operand hat keine Wirkung auf Prozeduren, die mit Lesekennwort geschützt sind (siehe Abschnitt „Zulässigkeit des Protokollierens“ auf Seite 114).

CMD = *UNCHANGED / *YES / *NO
Gibt an, ob Kommandos protokolliert werden.
Bei *UNCHANGED wird die bisherige Vereinbarung unverändert übernommen.
Die Angabe *YES ist nur wirksam, wenn Kommando-Protokollierung generell erlaubt ist.

DATA = *UNCHANGED / *YES / *NO
Gibt an, ob Daten protokolliert werden.
Bei *UNCHANGED wird die bisherige Vereinbarung unverändert übernommen. Die Angabe *YES ist nur wirksam, wenn Daten-Protokollierung generell erlaubt ist.

LOGGING = *YES
Schaltet die vereinbarte Protokollierung ein.
Die Angabe *YES ist nur wirksam, wenn Kommando- bzw. Daten-Protokollierung generell erlaubt ist.

LOGGING = *NO
Schaltet die Protokollierung aus.

Hinweis
Die Angaben LOGGING = *YES und LOGGING = *PARAMETERS(CMD= *YES, DATA=*YES) sind äquivalent, genauso wie LOGGING = *NO und LOGGING = *PARAMETERS (CMD=*NO,DATA=*NO).

BACK-BRANCH-LIMIT =
Bestimmt die maximal erlaubte Zahl an Rückwärtssprüngen innerhalb der Prozedur. Dabei werden Rückwärtsprünge, die das Kommando SKIP-COMMANDS verursacht, nicht mitgerechnet.

BACK-BRANCH-LIMIT = *UNCHANGED
Die bisherige Vereinbarung gilt unverändert.

BACK-BRANCH-LIMIT = *NONE
Die Zahl der Rückwärtssprünge wird nicht eingeschränkt.

BACK-BRANCH-LIMIT = <text 0..1800 with-low *arith-expr*>
Integer-Ausdruck, der die maximale Zahl erlaubter Rückwärtssprünge angibt.
Falls diese Grenze erreicht wird, wird die Fehlerbehandlung aktiviert.

FUNCTIONALITY =
Bestimmt, welche SDF-P-Funktionalität aktiviert bzw. simuliert werden soll. Die Einstellung gilt auf Task-Ebene, d.h. sie bleibt bis zur Beendigung der Task bzw. bis zur nächsten Modifikation bestehen.

FUNCTIONALITY = *UNCHANGED / *FULL / *BASIC
Die bisherige Vereinbarung gilt unverändert.

FUNCTIONALITY = *FULL
Falls das kostenpflichte Subsystem SDF-P im System geladen ist, wird die volle SDF-P-Funktionalität aktiviert.

FUNCTIONALITY = *BASIC
Es wird die SDF-P-BASYS-Funktionalität simuliert. Falls das kostenpflichte Produkt SDF-P im System verfügbar ist, wird das Kommando MODIFY-PROCEDURE-TEST-OPTIONS auch bei dieser Einstellung ausgeführt.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	64	SDP0133	Unvollständiger Datentyp beim Ausdruck
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Funktion LOGGING-MODE( ), Seite 449.

OPEN-VARIABLE-CONTAINER
Variablenbehälter öffnen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando OPEN-VARIABLE-CONTAINER werden Variablenbehälter, die als PLAM-Bibliothekselemente abgespeichert sind, geöffnet. Existiert ein solcher Variablenbehälter bzw. ein solches Element beim Aufruf dieses Kommandos noch nicht, wird er bzw. es automatisch erzeugt.

Damit ist es möglich, permanent vorhandene S-Variablen zu erzeugen, d.h. S-Variablen, die in ihrer Existenz nicht von der jeweiligen Task abhängig sind.

Format

OPEN-VARIABLE-**CONTAIN**ER

CONTAINER-**NAME** = <composed-name 1..64>
,**FROM-FILE** = ***LIB**RARY-**ELEM**ENT (...)
***LIB**RARY-**ELEM**ENT(...)
LIBRARY = <filename 1..54 without-vers>
,**ELEM**ENT = ***CONTAIN**ER-**NAME** / <composed-name 1..64>(...)
<composed-name 1..64>(...)
VERSION = ***HIGH**EST-**EXIST**ING / <composed-name 1..24>
,**LOCK-ELEMENT** = ***NO** / ***Y**ES
,**SCOPE** = ***CURR**ENT / ***PROC**EDURE / ***TASK**(...)
***TASK**(...)
SAVE-AT-TERMINATION = ***NO** / ***Y**ES
,**AUTOMATIC-DECLARE** = ***ALL** / ***NONE** / <structured-name 1..20 with-wild(40)> /
list-poss(2000): <structured-name 1..20>

Operandenbeschreibung

CONTAINER-NAME = <composed-name 1..64>
Name des Variablenbehälters.

FROM-FILE = *LIBRARY-ELEMENT(...)
Bibliothekselement, das den Variablenbehälter enthält.
Der Datentyp des Elements ist SYSVCONT.

LIBRARY = <filename 1..54 without-vers>
Name der Plam-Bibliothek.
Die Angabe einer Bibliotheksliste (S-Variable SYSPLAMALT-<name>) ist erlaubt

ELEMENT =
Name des Elements.

ELEMENT = *CONTAINER-NAME
Der Name des Elements ist identisch mit dem des Variablenbehälters.

ELEMENT = <composed-name 1..64>(...)
Der Name des Elements kann sich von dem des Variablenbehälters unterscheiden.

VERSION =
Bezeichnet die Versionsnummer des Elements.

VERSION = *HIGHEST-EXISTING
Wählt die höchste existierende Versionsnummer.

VERSION = <composed-name 1..24>
Wählt die angegebene Versionsnummer.

LOCK-ELEMENT =
Gibt an, ob das Element geschlossen ist oder nicht.

LOCK-ELEMENT = *NO
Das Element ist im Input-Modus geöffnet. Die Behälter-Variablen werden von diesem Element in den Variablenbehälter kopiert. Anschließend wird das Element geschlossen.

LOCK-ELEMENT = *YES
Das Element ist im Input- wie im Output-Modus geöffnet. Die Behälter-Variablen werden von diesem Element in den Variablenbehälter kopiert. Anschließend bleibt das Element geöffnet, bis das Kommando CLOSE-VARIABLE-CONTAINER angegeben wird. Nachfolgende Angaben von OPEN-VARIABLE-CONTAINER in derselben oder in anderen Tasks werden abgewiesen.

SCOPE =
Definiert den Geltungsbereich des Variablenbehälters. Er kontrolliert den Zugriff zu den im Variablenbehälter enthaltenen Variablen. Der Geltungsbereich der Behälter-Variablen darf nicht größer sein als der des Variablenbehälters.

SCOPE = *CURRENT
Der Geltungsbereich des Variablenbehälters ist prozedurlokal (siehe dazu Abschnitt „Geltungsbereich von Variablen“ auf Seite 162).
Der Variablenbehälter kann nur in der lokalen Prozedur und in allen tieferliegenden Include-Prozeduren benutzt werden, nicht aber in der aufrufenden Prozedur. Er wird implizit beim Ende der aktuellen Prozedur geschlossen.

SCOPE = *PROCEDURE
Der Geltungsbereich des Variablenbehälters ist prozedurlokal (siehe dazu Abschnitt „Geltungsbereich von Variablen“ auf Seite 162).
Der Variablenbehälter kann in der lokalen Prozedur und in allen tieferliegenden Include-Prozeduren benutzt werden. Er kann auch in den aufrufenden Prozeduren benutzt werden, wenn der Aufruf mit INCLUDE-PROCEDURE erfolgte. Er wird implizit beim Ende der zuerst aufgerufenen Prozedur geschlossen. D.h. er ist über alle Include-Prozeduren hinweg bis zum Ende der hierarchiehöchsten Aufruf-Prozedur geöffnet.

SCOPE = *TASK(...)
Der Geltungsbereich des Variablenbehälters ist taskglobal (siehe dazu Abschnitt „Geltungsbereich von Variablen“ auf Seite 162).
Der Variablenbehälter kann benutzt werden, solange er nicht geschlossen oder die Task beendet wird. Im Unterschied zum Geltungsbereich von Variablen muss der Name des Behälters nicht früher, als er benützt wird, importiert werden.

SAVE-AT-TERMINATION =
Gibt an, ob der Variablenbehälter bei EXIT-JOB bzw. LOGOFF gesichert werden muss.

SAVE-AT-TERMINATION = *NO
Der Variablenbehälter wird bei EXIT-JOB nicht gesichert.

SAVE-AT-TERMINATION = *YES
Der Variablenbehälter wird bei EXIT-JOB gesichert. Allerdings wird er nicht bei abnormaler Task-Beendigung gesichert, wie z.B. bei der Einstellung EXIT-JOB MODE = ABNORMAL.

AUTOMATIC-DECLARE =
Gibt an, ob die Behälter-Variablen automatisch deklariert werden sollen.

AUTOMATIC-DECLARE = *ALL
Die Behälter-Variablen werden automatisch mit dem Geltungsbereich des Variablenbehälters deklariert.

AUTOMATIC-DECLARE = *NONE
Behälter-Variablen werden nicht automatisch deklariert.

AUTOMATIC-DECLARE = list-poss(2000): <structured-name 1..20>
Die angegebenen Behälter-Variablen werden automatisch mit dem Geltungsbereich des Variablenbehälters deklariert.

AUTOMATIC-DECLARE = <structured-name 1..20 with-wild(40)>
Die Behälter-Variablen, die das angegebene Suchmuster erfüllen, werden automatisch mit dem Geltungsbereich des Variablenbehälters deklariert.

Hinweise

- Variablen in einem Variablenbehälter können mit dem Kommando DECLARE-VARIABLE und dem darin enthaltenen Operanden CONTAINER erzeugt werden.
- Variablen, die als statische Strukturlayouts deklariert wurden, werden mit dem Namen dieses Strukturlayouts gesichert.
- Eine Bezugnahme auf einen Variablenbehälter ist erst erlaubt, wenn er mit OPEN-VARIABLE-CONTAINER erzeugt wurde.
- Wenn Variablen mit OPEN-VARIABLE-CONTAINER automatisch erzeugt werden und die Variable mit anderen Attributen schon existiert, wird die Deklaration abgewiesen und die Fehlermeldung SDP1018 zur Warnung zurückgegeben. Der Prozess des Öffnens wird allerdings fortgesetzt.
  Der Benutzer kann die zurückgewiesene Variable mit dem S-Variablenstrom SYSMSG abfragen.
- Wenn Behälter-Variablen mit AUTOMATIC-DECLARE erzeugt werden und sich auf statische Strukturen beziehen, werden sie in Strukturen vom Typ „*BY-SYSCMD“ konvertiert.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
2	0	SDP00xx	Warnung, weil folgendes passiert ist: garantierte Meldungen: SDP1008, SDP1018
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommando SHOW-VARIABLE-CONTAINER-ATTR, Seite 788.

RAISE-ERROR
Returncode erzeugen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

RAISE-ERROR erzeugt einen Kommando-Returncode und stößt anschließend die Fehlerbehandlung an, wenn SUBCODE 1 nicht Null ist.

Format

RAISE-ERROR
SUBCODE1 = **64** / <integer 0..255> ,**SUBCODE2** = **0** / <integer 0..255> ,**MAIN**CODE = **SDP0018** / <alphanum-name 7..7>

Operandenbeschreibung

SUBCODE1 = 64 / <integer 0..255>
Zahl, die die Fehlerklasse angibt;
64 = Fehlerklasse „SEMANTIC-ERROR“.

SUBCODE2 = 0 / <integer 0..255>
Zusatzinformation zur Fehlerklasse.

MAINCODE = SDP0018 / <alphanum-name 7..7>
Fehlerschlüssel zur differenzierten Bestimmung der Fehlerursache.
Der voreingestellte Wert für MAINCODE ist SDP0018.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar
xx	xx	xxxxxxx	Returncode wie in Operanden übergeben

READ-VARIABLE
Variablen Werte zuweisen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando READ-VARIABLE liest Daten satzweise von einem Eingabemedium ein und speichert sie in einer Variablen ab. Der Vorgang wird beendet, wenn der String „*END-OF-CMD“ oder Dateiende (EOF) auftritt.

Eingabemedium kann sein:

– das Terminal
– eine katalogisierte Datei
– eine Variable
– ein Bibliothekselement
– die Systemdatei SYSDTA

Hinweise zum Einlesen von SYSDTA

– Daten werden von SYSDTA genauso eingelesen wie von einer Datei. Allerdings mit folgender Ausnahme: Wenn das Lesen vor SYSDTA-EOF beendet wird, wird beim nächsten Lesen von SYSDTA auf den nächsten SYSDTA-Satz zugegriffen. Das erlaubt gezieltes Lesen von SYSDTA, z.B. für den Fall: wenn nur eine einfache Variable gelesen werden soll.
– Die Eingabe endet mit dem Ende der Systemdatei SYSDTA, d.h. entweder bei Dateiende, wenn SYSDTA einer katalogisierten Datei zugewiesen ist, oder mit dem nächsten Kommando, wenn SYSDTA SYSCMD zugewiesen ist (Default-Voreinstellung von S-Prozeduren).
– Einlesen von SYSDTA kann mit HOLD-PROGRAM, K2 oder BEGIN-BLOCK PROGRAM-INPUT=*MIXED-WITH-CMD nicht unterbrochen werden (um in den Kommando-Modus umzuschalten). Diese Kommandos bzw. Maßnahmen verursachen nur, dass ein Einlesestop, d.h. SYSDTA-EOF (Dateiende), zum jeweiligen Kommando gemeldet wird. Nur das Kommando SEND-DATA beendet das READ-VARIABLE-Kommando nicht.

Hinweis zum Einlesen einer Datei

Wenn nicht der komplette Inhalt der Datei für die weitere Bearbeitung benötigt wird, kann in folgenden Fällen die Datenmenge bereits beim Einlesen beschränkt werden:

1. Es wird nur ein zusammenhängender Teil von Datensätzen benötigt. Für den einzulesenden Bereich von Datensätzen wird die Nummer des ersten und des letzten benötigten Datensatzes in den Operanden BEGIN-RECORD and END-RECORD angegeben.

2. Es werden nur Datensätze benötigt, die bestimmte Zeichenfolgen enthalten. Die einzulesenden Datensätze werden über die im Operanden PATTERN angegebene Suchzeichenfolge selektiert. Der Operand PATTERN-TYPE bestimmt, ob die Suchzeichenfolge als String oder als regulärer Ausdruck nach POSIX-Regeln auszuwerten ist.
   Eine einfache Suchzeichenfolge kann direkt als String angegeben werden. Ein regulärer Ausdruck ermöglicht die Angabe einer komplexen Suchzeichenfolge, die nicht direkt als String angegeben werden kann (siehe „POSIX-Platzhalter“ auf Seite 558; der „reguläre Ausdruck“ ist im Handbuch „POSIX Kommandos“ [18] beschrieben).

   Hinweis

   Eine Suchzeichenfolge wird immer innerhalb des gesamten Datensatz gesucht, auch wenn beim Einlesen nur ein Teil (Spaltenbereich) des Datensatzes übernommen werden soll (siehe Beispiel 5, Fall 3 auf Seite 724).

3. Es wird jeweils nur ein Teil aus jedem Datensatz benötigt. Der einzulesende Datensatzteil wird als Spaltenbereich mit der Spalte des ersten und des letzten benötigten Zeichens in den Operanden BEGIN-COLUMN and END-COLUMN angegeben.

Format

READ-VARIABLE Kurzname: **RDV**

VARIABLE**-NAME** = ***BY-INPUT**(...) / ***LIST**(...) / list-poss(2000): <composed-name 1..255>

***BY-INPUT**(...)

PREFIX = ***NONE** / <composed-name 1..255>

***LIST**(...)

LIST-NAME = <composed-name 1..255>

,**WRITE-MODE** = ***REPLACE** / ***EXTEND**

,**STRING-QUOTES** = ***OPT**IONAL / ***Y**ES / ***NO**

,**INPUT** = ***TER**MINAL(...) / <filename 1..54 without-gen-vers>(...) / ***VAR**IABLE(...) / ***LIB**RARY**-ELEM**ENT(...) / ***SYSDTA**(...)

***TER**MINAL(...)

PROMPT-STRING = '>>' / <text 0..1800 with-low>

,**SECRET-INPUT** = ***NO** / ***Y**ES

<filename 1..54 without-gen-vers>(...)

REMOVE**-KEY** = ***Y**ES / ***NO**

,**BEG**IN**-REC**ORD = ***FIRST** / <integer 1..2147483647>

,**END-REC**ORD = ***LAST** / <integer 1..2147483647>

,**BEG**IN**-COL**UMN = ***FIRST** / <integer 1..2147483647>

,**END-COL**UMN = ***LAST** / <integer 1..2147483647>

,**PATTERN** = ***NONE** / <c-string 0..1800 with-low>

,**PATTERN-TYPE** = ***STRING** / ***REGULAR-EXPRESSION**

***VAR**IABLE(...)

VARIABLE**-NAME** = <composed-name 1..255>

***LIB**RARY**-ELEM**ENT(...)

LIBRARY = <filename 1..54 without-vers>

,**ELEM**ENT = <composed-name 1..64>(...)

<composed-name 1..64>(...)

VERSION = ***HIGH**EST**-EXIST**ING / <composed-name 1..24>

,**TYPE** = **S** / <alphanum-name 1..8>

***SYSDTA**(...)

REMOVE**-KEY** = ***Y**ES / ***NO**

Operandenbeschreibung

VARIABLE-NAME =
Bezeichnet die Variablen, denen Werte zugewiesen werden sollen.

VARIABLE-NAME = *BY-INPUT(...)
Gibt an, dass die Variablen in dem Format vorliegen, wie es das Kommando SHOW-VARIABLE über die Operanden PREFIX und FORM erzeugt hat.
Werden die Variablen nicht vom Kommando SHOW-VARIABLE erzeugt, dann müssen sie im SHOW-VARIABLE-Ausgabeformat vorliegen (z.B. vor und nach dem '=' muss ein Leerzeichen stehen). Existieren die Variablen bereits, wird ihnen der Wert zugewiesen.

Existiert eine Variable noch nicht, so wird sie implizit angelegt, wenn implizite Deklaration erlaubt ist.
Existiert zu einem Variablenelement die übergeordnete zusammengesetzte Variable nicht, so wird diese mit SCOPE = *CURRENT angelegt.
Statische Strukturen können nur als dynamische Strukturen wiedererzeugt werden.
STRING, INTEGER und BOOLEAN werden auf ANY abgebildet, falls die Variablen nicht existieren.

Bezieht sich die Eingabe in *BY-INPUT(...) auf ein Listenelement (Variablenname mit (*LIST)), dann wird das Kommando mit Fehler beendet.
Wird einer Variablen der Wert *NO-INIT zugewiesen, wird ihr bisheriger Inhalt gelöscht (entspricht einem Kommandoaufruf FREE-VARIABLE).
Der Wert *END-OF-VAR wird ignoriert. Der Wert *END-OF-CMD beendet die Zuweisung.

PREFIX = *NONE
Dem Variablennamen wird kein Präfix vorangestellt.

PREFIX = <composed-name 1..255>
Bezeichnet den Namen, der als Präfix dem Variablennamen vorangestellt wird.

VARIABLE-NAME = *LIST(...)
Bezeichnet eine Listenvariable.

LIST-NAME = <composed-name 1..255>
Name der Liste.
Ist die Liste bereits vorhanden, so muss sie vom Typ ANY, STRING, BOOLEAN oder INTEGER sein.
Existiert die Liste nicht oder kann sie nicht gefunden werden, dann wird sie neu angelegt, mit dem Geltungsbereich SCOPE = *CURRENT und dem Datentyp TYPE = *STRING.

WRITE-MODE = *REPLACE
Die Liste wird überschrieben.

WRITE-MODE = *EXTEND
Neue Elemente werden an das Ende der Liste angehängt.

VARIABLE-NAME = list-poss(2000): <composed-name 1..255>
Namen der Variablen, denen neue Werte zuzuweisen sind.
Die eingelesenen Werte werden den Variablen(elementen) - angefangen beim ersten Element der Struktur - der Reihe nach zugewiesen.
Die Fehlerbehandlung wird gestartet, wenn wegen vorzeitigem EOF oder *END-OF-CMD nicht alle Variablen mit Werten versorgt wurden.

STRING-QUOTES =
Legt fest, wie der Eingabewert interpretiert werden soll.

STRING-QUOTES = *OPTIONAL
Bewirkt, dass Variablenwerte wie Prozedurparameter angegeben werden können.
Bei der Interpretation der Eingabe wird der Datentyp der Variablen, der der Wert zugewiesen werden soll, berücksichtigt:

– Ist die Variable vom Datentyp ANY oder STRING, so wird der Wert als String interpretiert. Hochkommas innerhalb des Strings müssen verdoppelt werden (allerdings nur wenn der ganze String in Hochkommata eingeschlossen ist).
– Ist die Variable vom Datentyp INTEGER, so wird der Wert als Integer-Wert interpretiert.
– Ist die Variable vom Datentyp BOOLEAN, so wird der Wert als boolescher Wert interpretiert.

STRING-QUOTES = *YES
Interpretiert Werte, die mit Hochkomma (') oder C-Hochkomma (C') beginnen, als String. Ist der Wert nicht mit Hochkomma abgeschlossen oder sind Hochkommas innerhalb des Wertes nicht verdoppelt, so gilt dies als Fehler.
Falls ein Wert, der nicht in Hochkomma eingeschlossen ist, einem der Schlüsselwörter für boolesche Werte entspricht, so wird er als boolescher Wert interpretiert. In allen anderen Fällen wird er als Integer-Wert interpretiert.

STRING-QUOTES = *NO
Interpretiert alle Werte als String.

INPUT =
Legt fest, von wo die Werte eingelesen werden sollen.

INPUT = *TERMINAL(...)
Prompting: Die Werte werden von der Datenstation eingelesen. (Dabei werden die Werte aber nicht wie bei DECLARE-PARAMETER INITIAL-VALUE =*PROMPT in Großbuchstaben konvertiert.)

PROMPT-STRING = '>>' / <text 0..1800 with-low>
String-Ausdruck. Definiert das Zeichen bzw. die Zeichenfolge, die als Eingabeaufforderung ausgegeben werden soll.

SECRET-INPUT = *NO / *YES
Gibt an, ob die Benutzereingabe geschützt werden soll.

INPUT = <filename 1..54 without-gen-vers>(...)
Die Werte sind aus der angegebenen Datei einzulesen.
Jeder Eingabesatz gilt als einzelner Wert.

REMOVE-KEY =
Gibt an, ob beim Lesen aus einer ISAM-Datei der Schlüssel weggelassen werden soll.

REMOVE-KEY = *YES
Bei einer ISAM-Datei werden die Schlüssel nicht mit eingelesen. Bei SAM-Datei wird nur diese Angabe akzeptiert.

REMOVE-KEY = *NO
Bei ISAM-Dateien mit den Standardattributen KEY-POSITION=5 und KEY-LENGTH=8 wird der Schlüssel mit eingelesen. Bei ISAM-Dateien mit anderen Attributen führt die Angabe zum Fehler. Bei SAM-Dateien führt diese Angabe in jedem Fall zum Fehler.

BEGIN-RECORD = *FIRST / <integer 1..2147483647>
Gibt an, mit welchem Datensatz das Einlesen beginnen soll. Mit *FIRST ist der erste Datensatz der Datei voreingestellt.
Falls der angegebene Wert größer als die Anzahl der vorhandenen Datensätze ist, wird eine leere Datei angenommen. Im Fall von VARIABLE-NAME=*LIST(...) wird eine leere Liste angelegt.

END-RECORD = *LAST / <integer 1..2147483647>
Gibt an, nach welchem Datensatz das Einlesen beendet werden soll. Mit *LAST ist der letzte Datensatz der Datei voreingestellt.
Falls der angegebene Wert größer als die Anzahl der vorhandenen Datensätze ist, wird *LAST angenommen.

BEGIN-COLUMN = *FIRST / <integer 1..2147483647>
Gibt an, ab welcher Spaltenposition der Inhalt eines Datensatzes eingelesen werden soll. Mit *FIRST ist der Beginn des Datensatzes (Spaltenposition 1) voreingestellt.
Falls der angegebene Wert größer als der Datensatz ist, werden keine Daten übernommen (keine Listenelemente angelegt).

END-COLUMN = *LAST / <integer 1..2147483647>
Gibt an, nach welcher Spaltenposition das Einlesen eines Datensatzes beendet werden soll. Mit *LAST das Ende des Datensatzes voreingestellt.
Falls der angegebene Wert größer als der Datensatz ist, wird *LAST angenommen.

PATTERN =
Gibt an, ob nur Datensätze, die eine bestimmte Suchzeichenfolge enthalten, eingelesen werden.

PATTERN = *NONE
Datensätze werden unabhängig von einer bestimmten Suchzeichenfolge eingelesen.

PATTERN = <c-string 0..1800 with-low>
Es werden nur Datensätze eingelesen, die die angegebene Suchzeichenfolge enthalten. Der Operand PATTERN-TYPE bestimmt, wie die Suchzeichenfolge auszuwerten ist.

PATTERN-TYPE =
Gibt an, wie die Suchzeichenfolge auszuwerten ist.

PATTERN-TYPE = *STRING
Es werden Datensätze selektiert, die den angegebenen String enthalten.

PATTERN-TYPE = *REGULAR-EXPRESSION
Es werden Datensätze selektiert, die den als regulären Ausdruck angegebenen String enthalten.

INPUT = *VARIABLE(...)
Die Werte sind aus einer Listenvariablen einzulesen.

VARIABLE-NAME = <composed-name 1..255>
Name der Listenvariablen.
Die Variable muss existieren und mit dem Datentyp STRING deklariert worden sein. (Sie kann auch mit dem Datentyp ANY deklariert sein. Jedoch darf sie dann auch nur Strings enthalten.)

INPUT = *LIBRARY-ELEMENT(...)
Die Werte sind aus dem Element einer PLAM-Bibliothek einzulesen. Die Angabe einer Bibliotheksliste (S-Variable SYSPLAMALT-<name>) ist erlaubt.

LIBRARY = <filename 1..54 without-vers>
Name der Bibliothek.

ELEMENT = <composed-name 1..64>(...)
Name des Bibliothekselements.

VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
Version des Bibliothekselements.

TYPE = S / <alphanum-name 1..8>
Typ des Bibliothekselements.

INPUT = *SYSDTA(...)
Die Werte werden von SYSDTA eingelesen.

REMOVE-KEY =
Falls SYSDTA einer Datei zugewiesen ist:
Gibt an, ob beim Lesen aus einer ISAM-Datei der Schlüssel weggelassen werden soll.

REMOVE-KEY = *YES
Bei einer ISAM-Datei wird der Schlüssel weggelassen. Bei einer SAM-Datei wird nur diese Angabe akzeptiert.

REMOVE-KEY = *NO
Nur bei einer ISAM-Datei mit den Standardattributen KEY-POSITION=5 und KEY-LENGTH=8 wird der Schlüssel mit eingelesen. Für eine ISAM-Datei mit anderen Attributen und für eine SAM-Datei führt diese Angabe zum Fehler.

Kommando-Returncode

Es ist möglich, dass ein Teil des Kommandos schon abgearbeitet und ausgeführt wurde, bevor der Fehler auftrat. In diesem Fall ist das Ergebnis des Kommandos nicht garantiert.

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
2	0	SDP2000	Warnung: Nicht alle Elemente der Eingabeliste konnten erfolgreich abgearbeitet werden. garantierte Meldung: SDP2000
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0089	INPUT-Fehler
	64	SDP0091	Semantikfehler garantierte Meldung: SDP1008
	64	SDP2001	Keines der Elemente konnte eingelesen werden
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel 1

Wenn eine bestimmte Menge von Werten in einer Prozedur benutzt werden muss, können diese Werte in SYSCMD aufgelistet werden; sie müssen nicht in eine andere Datei gebracht werden.

```
/DECLARE-VARIABLE FILES(TYPE=*STRING),MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(FILES),INPUT=*SYSDTA
FILE1
FILE2
FILE3
/FOR FILE=*LIST(FILES)
/   IF (NOT IS-CATALOGED-FILE(FILE))
/      WRITE-TEXT 'DATEI &FILE NICHT GEFUNDEN.'
/   END-IF
/END-FOR
```

Beispiel 2

Von einer Datei soll eine ganz bestimmte, konstante Anzahl von Zeilen gelesen werden: in diesem Beispiel sind es genau 10. Damit werden bei jedem Schleifendurchlauf 10 Zeilen der Datei „MY-FILE“ gelesen, und eine Liste „LINES“ wird deklariert mit der Grenze von 10 Listenelementen. Kurz: Dieses Beispiel zeigt den Inhalt der Datei „MY-FILE“ an.

```
/ASSIGN-SYSDTA TO=MY-FILE
/DECLARE-VARIABLE LINES(TYPE=*STRING),MULTIPLE-ELEMENTS=*LIST(LIMIT=10)
/REPEAT
/   READ-VARIABLE *LIST(LINES),INPUT=*SYSDTA
/   SHOW-VARIABLE LINES,INFORMATION=*PARAMETERS(NAME=*NONE)
/UNTIL (SIZE('LINES') LT 10)
```

Beispiel 3

Eine Prozedur hat folgenden Inhalt:

```
/DECLARE-VARIABLE VORNAME
/WRITE-TEXT 'Geben Sie einen Vornamen ein!'
/READ-VARIABLE VORNAME
/SHOW-VARIABLE VORNAME
```

Ausgabe

```
Geben Sie einen Vornamen ein!
```

Eingabe

```
Hans
```

Ausgabe

```
VORNAME = Hans
```

Beispiel 4

Eine Datei soll Zeile für Zeile bei Benutzung von SYSDTA gelesen werden. Ein Fehler wird gemeldet, wenn durch READ-VARIABLE das Dateiende „EOF“ von SYSDTA gefunden wird. So zeigt dieses Beispiel einfach den Inhalt der Datei „MY-FILE“ an.

```
/ASSIGN-SYSDTA TO=MY-FILE
/EOF=FALSE
/WHILE (NOT EOF)
/   READ-VARIABLE LINE,INPUT=*SYSDTA
/   IF-CMD-ERROR
/       EOF=TRUE
/   ELSE
/       SHOW-VARIABLE LINE,INFORMATION=*PARAMETERS(NAME=*NONE)
/   END-IF
/END-WHILE
```

Beispiel 5

Die Datei DATA-FILE enthält folgende Datensätze:

```
MUELLER       : 55900 : DE : Gladbecker Strasse 7
GRANDY        : 74663 : UK : Albert Street 12
DANFERTH      : 83092 : DE : Colmberger Strasse 2
GRABATER      : 01927 : FR : rue du Couedic 27
SMITH         : 54920 : UK : Elizabeth Street 54
DUPONT        : 45888 : FR : rue de la Maderie 78
VANDENMORSE   : 94958 : BE : Brusselssteenweg 101
```

1. Datensatzbereich einlesen

```
/DECLARE-VARIABLE FILE (TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(FILE), INPUT=SDF-P-FILE (BEGIN-RECORD = 4, -
/                                             END-RECORD = 7)
/SHOW-VARIABLE FILE
FILE(*LIST) = GRABATER      : 01927 : FR : rue du Couedic 27
FILE(*LIST) = SMITH         : 54920 : UK : Elizabeth Street 54
FILE(*LIST) = DUPONT        : 45888 : FR : rue de la Maderie 78
FILE(*LIST) = VANDENMORSE   : 94958 : BE : Brusselssteenweg 101
```

   In die Variable FILE wurden die Datensätze 4 bis 7 eingelesen.

2. Spaltenbereich aus Datensatzbereich einlesen

```
/DECLARE-VARIABLE FILE (TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(FILE), INPUT=SDF-P-FILE (BEGIN-RECORD=5, -
/                                             END-RECORD=8, -
/                                             BEGIN-COLUMN=17, -
/                                             END-COLUMN=21)
/SHOW-VARIABLE FILE
FILE(*LIST) = 54920
FILE(*LIST) = 45888
FILE(*LIST) = 94958
```

   In die Variable FILE wurden jeweils die Spalten 17 bis 21 aus den Datensätzen 5 bis 7 eingelesen (ein 8. Datensatz existiert nicht).

3. Datensätze nach String durchsuchen und Treffer einlesen

```
/DECLARE-VARIABLE FILE (TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(FILE), INPUT=SDF-P-FILE (PATTERN=': DE :', -
/                                             PATTERN-TYPE=*STRING)
/SHOW-VARIABLE FILE
FILE(*LIST) = MUELLER       : 55900 : DE : Gladbecker Strasse 7
FILE(*LIST) = DANFERTH      : 83092 : DE : Colmberger Strasse 2
```

   In die Variable FILE wurden alle Datensätze eingelesen, die den String „: DE :“ enthalten.

```
/READ-VARIABLE *LIST(FILE), INPUT=SDF-P-FILE (BEGIN-COLUMN=23, -
/                                             END-COLUMN=28, -
/                                             PATTERN=': DE :', -
/                                             PATTERN-TYPE=*STRING)
/SHOW-VARIABLE FILE
FILE(*LIST) = : DE :
FILE(*LIST) = : DE :
```

In die Variable FILE wurden aus allen Datensätzen, die den String „: DE :“ enthalten, die Spalten 23 bis 28 eingelesen.

```
/READ-VARIABLE *LIST(FILE), INPUT=SDF-P-FILE (BEGIN-COLUMN=1, -
/                                             END-COLUMN=22, -
/                                             PATTERN=': DE :', -
/                                             PATTERN-TYPE=*STRING)
/SHOW-VARIABLE FILE
FILE(*LIST) = MUELLER       : 55900
FILE(*LIST) = DANFERTH      : 83092
```

In die Variable FILE wurden aus allen Datensätzen, die den String „: DE :“ enthalten, die Spalten 1 bis 22 eingelesen.

4. Datensätze nach String oder regulären Ausdruck durchsuchen und Treffer einlesen

```
/DECLARE-VARIABLE FILE (TYPE=*STRING), MULTIPLE-ELEMENTS=*LIST
/READ-VARIABLE *LIST(FILE), INPUT=SDF-P-FILE (PATTERN=': DE :', -
/                                             PATTERN-TYPE=*STRING)
/SHOW-VARIABLE FILE
FILE(*LIST) = MUELLER       : 55900 : DE : Gladbecker Strasse 7
FILE(*LIST) = DANFERTH      : 83092 : DE : Colmberger Strasse 2
```

In die Variable FILE wurden alle Datensätzen eingelesen, die den String „: DE :“ enthalten.

```
/READ-VARIABLE *LIST(FILE) -
/             , INPUT=SDF-P-FILE (PATTERN=': [DB]E :', -
/                                  PATTERN-TYPE=*REGULAR-EXPRESSION)
/SHOW-VARIABLE FILE
FILE(*LIST) = MUELLER       : 55900 : DE : Gladbecker Strasse 7
FILE(*LIST) = DANFERTH      : 83092 : DE : Colmberger Strasse 2
FILE(*LIST) = VANDENMORSE   : 94958 : BE : Brusselssteenweg 101
```

In die Variable FILE wurden alle Datensätzen eingelesen, die den regulären Ausdruck „: [DB]E :“ (also „: DE :“ oder „: BE :“) enthalten.

REPEAT
REPEAT-Block einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

REPEAT leitet einen REPEAT-Block (REPEAT-Schleife) ein. Die Ausführung der Kommandofolge innerhalb des REPEAT-Blocks wird wiederholt, solange die Schleifenbedingung erfüllt ist.

Die Schleifenbedingung wird im UNTIL-Kommando geprüft, das den REPEAT-Block abschließt. Wenn die Bedingung nicht erfüllt ist, wird zum ersten Kommando innerhalb des REPEAT-Blockes zurückgekehrt; andernfalls wird die Kommandoausführung nach dem UNTIL-Kommando fortgesetzt (siehe auch Abschnitt „REPEAT-Block“ auf Seite 100).

Format

REPEAT

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Abschnitt „Prozedurrumpf erstellen“ auf Seite 93.

REPEAT-CMD
Kommandoausführung wiederholen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando REPEAT-CMD ermöglicht die wiederholte Ausführung eines Kommandos, wobei ein speziell gekennzeichneter Teil (Namensteil, Operand, Operandenwert) nacheinander durch die Elemente einer Eingabeliste substituiert wird. Diese Elemente können als Sätze in einer Datei, einem Bibliothekselement oder als Listenelemente einer S-Variablen spezifiziert oder direkt am Terminal eingegeben werden. Das angegebene Kommando wird für jedes Element der Eingabeliste aufgerufen.

Die Substitution kann prinzipiell an beliebiger Stelle des Kommandos erfolgen; sie kann auch an mehreren Stellen vereinbart werden. Die Wiederholung des Kommandos erfolgt aber immer nur in einfacher Schleife (keine Unterschleifen).

Es kann angegeben werden, ob in einem Selektionsmenü die Elemente der Eingabeliste nochmals zur Überprüfung ausgegeben werden sollen.

Das Kommando ist besonders zur Ausführung von Kommandos geeignet, die nicht die Angabe von mehreren Werten (list-possible-Option) unterstützen.

Format

REPEAT-CMD Kurzname: **REPCMD**

CMD = <text 0..1800 with-low>

,**SUBST**ITUTION**-LIST** = ***TERMINAL** / <filename 1..54 without-gen-vers> / ***VAR**IABLE(...) /
***LIB**RARY**-ELEM**ENT(...)

***VAR**IABLE(...)
VARIABLE**-NAME** = <composed-name 1..255>

***LIB**RARY**-ELEM**ENT(...)
LIBRARY = <filename 1..54 without-vers>
,**ELEM**ENT = <composed-name 1..64>(...)
<composed-name 1..64>(...)
VERSION = ***HIGH**EST**-EXIST**ING / <composed-name 1..24>
,**TYPE** = **S** / <alphanum-name 1..8>

,**DIALOG-SEL**ECTION = ***NO** / ***Y**ES

,**CONT**INUE**-AFTER-ERROR** = ***Y**ES / ***NO**

Operandenbeschreibung

CMD = <text 0..1800 with-low>

Angabe des BS2000-Kommandos, das wiederholt ausgeführt werden soll. Die Angabe ist in runde Klammern einzuschließen. Als Platzhalter für zu substituierende Teile sind die Zeichen %* anzugeben. Der Kommandoname ist ohne führenden Schrägstrich anzugeben.
Beispiel: `CMD=(WRITE-TEXT TEXT='%*')`

SUBSTITUTION-LIST = *TERMINAL / <filename 1..54 without-gen-vers> / VARIABLE(...) / *LIBRARY-ELEMENT(...)

Bezeichnet das Eingabemedium für die Elemente der Eingabeliste.

SUBSTITUTION-LIST = *TERMINAL

Die Elemente der Eingabeliste werden vom Terminal gelesen. Nach Absenden des Kommandos REPEAT-CMD wird als Eingabeaufforderung die Zeichenfolge „%>>:“ ausgegeben. Jede Eingabe für ein Element ist mit den Tasten EM DUE abzusenden. Danach erscheint erneut die Eingabeaufforderung. Das Einlesen wird beendet, wenn der String *END-OF-CMD eingegeben wird.

SUBSTITUTION-LIST = <filename 1..54 without-gen-vers>

Die Elemente der Eingabeliste werden aus der angegebenen Datei (ISAM-Datei mit Standardschlüsseln oder SAM-Datei) gelesen. Das Einlesen wird beendet, wenn Dateiende erkannt oder der String *END-OF-CMD gelesen wird.

SUBSTITUTION-LIST = VARIABLE(...)
Die Elemente der Eingabeliste der angegebenen Variablenliste werden gelesen. Das Einlesen wird beendet, wenn das Listenende erkannt oder der String *END-OF-CMD gelesen wird.

VARIABLE-NAME = <composed-name 1..255>
Name der Listenvariablen.
Die Variable muss existieren und mit dem Datentyp STRING deklariert worden sein. (Sie kann auch mit dem Datentyp ANY deklariert sein. Jedoch darf sie dann auch nur Strings enthalten.)

SUBSTITUTION-LIST = *LIBRARY-ELEMENT(...)
Die Elemente der Eingabeliste aus dem angegebenen Bibliothekselement werden gelesen. Das Einlesen wird beendet, wenn Dateiende erkannt oder der String *END-OF-CMD gelesen wird.

LIBRARY = <filename 1..54 without-vers>
Name der Bibliothek.

ELEMENT = <composed-name 1..64>(...)
Name des Bibliothekselements.

VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
Version des Bibliothekselements.

TYPE = S / <alphanum-name 1..8>
Typ des Bibliothekselements.

DIALOG-SELECTION = *NO / *YES
Bestimmt, ob am Terminal ein Selektionsmenü ausgegeben wird. In diesem Menü werden alle Elemente der Eingabeliste zur nochmaligen Verifikation aufgelistet; Markierung mit beliebigem Zeichen möglich.

CONTINUE-AFTER-ERROR = *YES / *NO
Bestimmt, ob nach fehlerhafter Kommandoausführung die Verarbeitung mit dem nächsten Element der Eingabeliste fortgesetzt oder abgebrochen wird.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
2	0	SDP2000	Warnung: Nicht alle Elemente der Eingabeliste konnten erfolgreich abgearbeitet werden. garantierte Meldung: SDP2000
	1	SDP2001	Keines der Elemente der Eingabeliste konnte erfolgreich abgearbeitet werden. garantierte Meldung: SDP2001

Wurde aufgrund der Angabe CONTINUE-AFTER-ERROR=*NO die Verarbeitung nach dem ersten Fehler abgebrochen (Meldung SDP2003), so wird der Returncode des fehlerhaften Kommandos geliefert.

Beispiel

Das Kommando ENTER-PROCEDURE soll mehrmals für dieselbe Prozedur, aber mit verschiedenen Auftragsnamen aufgerufen werden. Die Werte sollen am Terminal eingegeben und noch einmal kontrolliert werden:

```
/REPEAT-CMD (ENTER-PROC FROM=PROC.WAIT-600,JOB-CLASS=JCB00050,
             JOB-NAME=%*),SUBSTITUTION-LIST=*TERMINAL,DIALOG-SELECT=*YES
%>>: PROC01
%>>: PROC02
%>>: PROC03
%>>: TESTA
%>>: TESTB
%>>: PROC04
%>>: *END-OF-CMD
```

Die Eingabe der Werte wird mit *END-OF-CMD beendet. Die eingegebenen Auftragsnamen werden jetzt noch einmal in einem Selektionsmenü zur Auswahl angeboten:

```
            Please select list elements
------------------------------------------------------------------------------

 (x) PROC01
 (x) PROC02
 (x) PROC03
 ( ) TESTA
 ( ) TESTB
 (x) PROC04









------------------------------------------------------------------------------
NEXT = *EXECUTE
       *ALL or *NONE or *EXECUTE or *CANCEL
```

Das Kommando ENTER-PROCEDURE startet die vier Aufträge mit den ausgewählten Auftragsnamen (im Menü selektiert durch Markierung mit x):

```
%  JMS0066 JOB 'PROC01' ACCEPTED ON 07-03-30 AT 11:17, TSN = 050L
%  JMS0066 JOB 'PROC02' ACCEPTED ON 07-03-30 AT 11:17, TSN = 050P
%  JMS0066 JOB 'PROC03' ACCEPTED ON 07-03-30 AT 11:17, TSN = 050Q
%  JMS0066 JOB 'PROC04' ACCEPTED ON 07-03-30 AT 11:17, TSN = 050R
/
```

REPEAT-STMT
Anweisungsausführung wiederholen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando REPEAT-STMT ermöglicht die wiederholte Ausführung einer Anweisung (SDF-Format), wobei ein speziell gekennzeichneter Teil (Namensteil, Operand, Operandenwert) nacheinander durch die Elemente einer Eingabeliste substituiert wird. Diese Elemente können als Sätze in einer Datei, einem Bibliothekselement oder als Listenelemente einer S-Variablen spezifiziert oder direkt am Terminal eingegeben werden. Die angegebene Anweisung wird für jedes Element der Eingabeliste aufgerufen.

Die Substitution kann prinzipiell an beliebiger Stelle der Anweisung erfolgen; sie kann auch an mehreren Stellen vereinbart werden. Die Wiederholung der Anweisung erfolgt aber immer nur in einfacher Schleife (keine Unterschleifen).

Es kann angegeben werden, ob in einem Selektionsmenü die Elemente der Eingabeliste nochmals zur Überprüfung ausgegeben werden sollen.

Das Kommando ist besonders zur Ausführung von Anweisungen geeignet, die nicht die Angabe von mehreren Werten (list-possible-Option) unterstützen.

Format

REPEAT-STMT Kurzname: **REPSTMT**

STMT = <text 0..1800 with-low>

,**SUBST**ITUTION-**LIST** = ***TERMINAL** / <filename 1..54 without-gen-vers> / ***VAR**IABLE(...) / ***LIB**RARY-**ELEM**ENT(...)

***VAR**IABLE(...)
VARIABLE-**NAME** = <composed-name 1..255>

***LIB**RARY-**ELEM**ENT(...)
LIBRARY = <filename 1..54 without-vers>
,**ELEM**ENT = <composed-name 1..64>(...)
<composed-name 1..64>(...)
VERSION = ***HIGH**EST-**EXIST**ING / <composed-name 1..24>
,**TYPE** = **S** / <alphanum-name 1..8>

,**DIALOG-SEL**ECTION = ***NO** / ***Y**ES

,**CONT**INUE-**AFTER-ERROR** = ***Y**ES / ***NO**

Operandenbeschreibung

STMT= <text 0..1800 with-low>
Anweisung im SDF-Format, die wiederholt ausgeführt werden soll. Die Angabe ist in runde Klammern einzuschließen. Als Platzhalter für zu substituierende Teile sind die Zeichen %* anzugeben. Der Anweisungsname ist ohne führende Schrägstriche anzugeben.
Beispiel: `STMT=(WRITE-TEXT TEXT='%*')`

SUBSTITUTION-LIST = *TERMINAL / <filename 1..54 without-gen-vers> / VARIABLE(...) / *LIBRARY-ELEMENT(...)
Bezeichnet das Eingabemedium für die Elemente der Eingabeliste.

SUBSTITUTION-LIST = *TERMINAL
Die Elemente der Eingabeliste werden vom Terminal gelesen. Nach Absenden des Kommandos REPEAT-STMT wird als Eingabeaufforderung die Zeichenfolge „%>>:“ ausgegeben. Jede Eingabe für ein Element ist mit den Tasten EM DUE abzusenden. Danach erscheint erneut die Eingabeaufforderung. Das Einlesen wird beendet, wenn der String *END-OF-CMD eingegeben wird.

SUBSTITUTION-LIST = <filename 1..54 without-gen-vers>
Die Elemente der Eingabeliste werden aus der angegebenen Datei (ISAM-Datei mit Standardschlüsseln oder SAM-Datei) gelesen. Das Einlesen wird beendet, wenn Dateiende erkannt oder der String *END-OF-CMD gelesen wird.

SUBSTITUTION-LIST = VARIABLE(...)
Die Elemente der Eingabeliste der angegebenen Variablenliste werden gelesen. Das Einlesen wird beendet, wenn das Listenende erkannt oder der String *END-OF-CMD gelesen wird.

VARIABLE-NAME = <composed-name 1..255>
Name der Listenvariablen.
Die Variable muss existieren und mit dem Datentyp STRING deklariert worden sein. (Sie kann auch mit dem Datentyp ANY deklariert sein. Jedoch darf sie dann auch nur Strings enthalten.)

SUBSTITUTION-LIST = *LIBRARY-ELEMENT(...)
Die Elemente der Eingabeliste aus dem angegebenen Bibliothekselement werden gelesen. Das Einlesen wird beendet, wenn Dateiende erkannt oder der String *END-OF-CMD gelesen wird.

LIBRARY = <filename 1..54 without-vers>
Name der Bibliothek.

ELEMENT = <composed-name 1..64>(...)
Name des Bibliothekselements.

VERSION = *HIGHEST-EXISTING / <composed-name 1..24>
Version des Bibliothekselements.

TYPE = S / <alphanum-name 1..8>
Typ des Bibliothekselements.

DIALOG-SELECTION = *NO / *YES
Bestimmt, ob am Terminal ein Selektionsmenü ausgegeben wird. In diesem Menü werden alle Elemente der Eingabeliste zur nochmaligen Verifikation aufgelistet; Markierung mit beliebigem Zeichen möglich.

CONTINUE-AFTER-ERROR = *YES / *NO
Bestimmt, ob nach fehlerhafter Anweisungsausführung die Verarbeitung mit dem nächsten Element der Eingabeliste fortgesetzt oder abgebrochen wird.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
2	0	SDP2000	Warnung: Nicht alle Elemente der Eingabeliste konnten erfolgreich abgearbeitet werden. garantierte Meldung: SDP2000
	1	SDP2001	Keines der Elemente der Eingabeliste konnte erfolgreich abgearbeitet werden. garantierte Meldung: SDP2001

Wurde aufgrund der Angabe CONTINUE-AFTER-ERROR=*NO die Verarbeitung nach dem ersten Fehler abgebrochen (Meldung SDP2003), so wird der Returncode der fehlerhaften Anweisung geliefert.

SAVE-RETURNCODE
Aktuellen Kommando-Returncode sichern

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

SAVE-RETURNCODE sichert den aktuellen Kommando-Returncode, sodass er mit SDF-P-Funktionen ausgewertet werden kann. Zur Auswertung des Returncodes stehen die vordefinierten Funktionen SUBCODE1( ), SUBCODE2( ) und MAINCODE( ) zur Verfügung (siehe Kapitel „Vordefinierte Funktionen“ auf Seite 353).

SAVE-RETURNCODE ist erforderlich, wenn kein Fehler auftrat, der Returncode aber ausgewertet werden soll.

Der gesicherte Returncode ist bis zum nächsten aufgetretenen Fehler (Fehler in irgendeinem Kommando) oder bis zum SAVE-RETURNCODE-Kommando verfügbar.

Hinweis

Bei IF-CMD-ERROR führt SDF-P implizit ein SAVE-RETURNCODE-Kommando aus.

Format

SAVE-RETURNCODE

Kommando-Returncode

(SC2) SC1	Maincode	Bedeutung
0	CMD0001	Ohne Fehler
1	CMD0202	Syntaxfehler
3	CMD2203	Falsche Syntaxdatei
32	CMD0221	Systemfehler (interner Fehler)
130	SDP0099	Kein Adressraum mehr verfügbar

SAVE-VARIABLE-CONTAINER
Variablenbehälter sichern

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando SAVE-VARIABLE-CONTAINER werden Variablenbehälter gesichert.

Format

SAVE-VARIABLE-**CONTAIN**ER

CONTAINER-**NAME** = <composed-name 1..64 with-wild(80)>(...) /
list-poss(2000):<composed-name 1..64>(...)

<composed-name 1..64 with-wild(80)>(...)

ELEMENT-**VERSION** = ***SAME** / ***INCREMENT**

list-poss(2000):<composed-name 1..64>(...)

ELEMENT-**VERSION** = ***SAME** / ***INCREMENT**

Operandenbeschreibung

CONTAINER-NAME =
Namen des Variablenbehälters.

CONTAINER-NAME = <composed-name 1..64 with-wild(80)>(...)
Variablenbehälter, der das angegebene Suchmuster erfüllt.

ELEMENT-VERSION =
Bezeichnet die Versionsnummer des Bibliothekselements.

ELEMENT-VERSION = *SAME
Die Versionsnummer des Elements bleibt unverändert.

ELEMENT-VERSION = *INCREMENT
Die Versionsnummer des Elements wird erhöht.
Es muss in `OPEN-VARIABLE-CONTAINER LOCK-ELEMENT = *NO` angegeben sein.

CONTAINER-NAME = list-poss(2000): <composed-name 1..64>(...)
Namenliste der Variablenbehälter.

ELEMENT-VERSION =
Bezeichnet die Versionsnummer des Bibliothekselements.

ELEMENT-VERSION = *SAME
Die Versionsnummer des Elements bleibt unverändert.

ELEMENT-VERSION = *INCREMENT
Die Versionsnummer des Elements wird erhöht.
Es muss in OPEN-VARIABLE-CONTAINER LOCK-ELEMENT = *NO angegeben sein.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommando SHOW-VARIABLE-CONTAINER-ATTR, Seite 788.

SELECT-VARIABLE-ELEMENTS
Elemente einer Listenvariablen auswählen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando SELECT-VARIABLE-ELEMENTS gibt die Elemente einer S-Variablen (Listenvariablen) auf den Bildschirm aus und schreibt davon ausgewählte Elemente in eine andere S-Variable. Der Bildschirm kann zur Benutzerführung durch Überschriften strukturiert werden. Die Bildschimführung (scrollen) erfolgt ähnlich wie im SDF-geführten Dialog; die Funktionstasten können mit dem Kommando /MODIFY-SDF-OPTIONS vereinbart werden.

Layout der Bildschirmausgabe

Der Bildschirm beginnt mit einer unterstrichenen Überschriftszeile, gefolgt von einer Kopfzeile zur Erläuterung der ausgegebenen Elementwerte. Beide Textzeilen werden im Kommando vereinbart, Operanden HEADER-LINE und TITLE.
Die Werte eines Elementes werden jeweils in eine Zeile geschrieben. Die Zeile beginnt mit einem Markierungsfeld „( )“, gefolgt von einem Blank und maximal 75 Zeichen. Die Abstände zwischen den Elementwerten können mit dem Operanden LENGTH implizit festgelegt werden (Standard = ␣). Siehe Beispiel am Ende der Kommandobeschreibung.

Markieren einer Zeile

Ein Element (Zeile) wird durch Eintragen eines beliebigen Zeichens zwischen die Klammern des Markierungsfeldes markiert. Im Kommando kann vereinbart werden, dass ein Element mit diesem Zeichen als Inhalt automatisch der Ausgabevariablen hinzugefügt wird.

Format

SELECT-**VAR**IABLE-**ELEM**ENTS

FROM-VARIABLE = <composed-name 1..255>

,**TO-VAR**IABLE = <composed-name 1..255>(...)

<composed-name 1..255>(...)

SELECTION-**CODE** = ***NO** / ***Y**ES

,**HEADER-LINE** = ***NONE** / <c-string 1..80>

,**MESSAGE** = ***NONE** / <text 1..240 with-low>

,**DISP**LAYED-**ELEM**ENTS = ***STD** / list-poss(5): <composed-name 1..255>(...)

<composed-name 1..255>(...)

LENGTH = ***BY-VALUE** / <integer 1..75>

,**TITLE** = ***BY-ELEMENT-NAME** / <c-string 1..75>

Operandenbeschreibung

FROM-VARIABLE = <composed-name 1..255>
Name der S-Variablen (Input-Variable, Listenvariable), deren Elemente angezeigt werden sollen.

TO-VARIABLE = <composed-name 1..255>(...)
Name einer S-Variablen (Output-Variable), in welche die markierten Elemente geschrieben werden. Eine bereits vorhandene Variable gleichen Namens wird überschrieben.

SELECTION-CODE =
Bestimmt, ob das zur Markierung benutzte Zeichen in die Output-Variable übertragen wird.

SELECTION-CODE = *NO
Das zur Markierung benutzte Zeichen wird nicht in die Output-Variable übertragen

SELECTION-CODE = *YES
Das Variablenelement SELECTION-CODE wird der Output-Variablen automatisch hinzugefügt. Ist die Input-Variable eine einfache Liste (integer, boolean oder string), wird der Output-Variablen außerdem das Element VALUE hinzugefügt, in welches das markierte Listenelement übertragen wird.
Die Output-Variable muss als dynamische Struktur deklariert sein, anderenfalls wird das Kommando abgewiesen.

HEADER-LINE = *NONE / <c-string 1..80>
Textzeile (Kopfzeile) für die Bildschirmmausgabe.

MESSAGE = *NONE / <text 1..240 with-low>
Nachricht, die im ersten Auswahlbildschirm unten (in den letzten 3 Zeilen) angezeigt wird. Die Nachricht wird nach Absenden des Bildschirms nicht mehr angezeigt.

Hinweis: Die Nachricht wird nur angezeigt mit SDF ≥ V2.6B.

DISPLAYED-ELEMENTS =
Bezeichnet ein oder mehrere Elemente der Input-Variablen und die Art der Darstellung auf dem Bildschirm.
Bei Strukturelementen werden nur die Werte auf der ersten Stufe ausgegeben.

DISPLAYED-ELEMENTS = *STD
Es ist zu unterscheiden, ob die Elemente wiederum einfache oder zusammengesetzte Variablen sind.
Einfache Variable:
Alle Elementwerte werden ausgegeben; pro Zeile ein Elementwert.
Zusammengesetzte Variable (Struktur):
Nur der Elementwert des jeweils ersten Unterelementes (Strukturelement) wird ausgegeben; pro Zeile ein Elementwert.

DISPLAYED-ELEMENTS = <composed-name 1..255>(...)
Namen von ein oder mehreren Variablenelementen, deren Werte pro Element in eine Zeile geschrieben werden. Die Elementwerte werden in der Reihenfolge ausgegeben, wie in der Operandenliste spezifiziert. Die Elemente müssen immer vom Datentyp „integer“, „boolean“ oder „string“ sein.

LENGTH = *BY-VALUE / <integer 1..76>
Länge, in der der Elementwert auszugeben ist. Die Summe der Elementwerte einer Zeile darf 75 Zeichen nicht überschreiten, einschließlich der Zwischenräume zwischen den Werten. Längere Zeilen werden abgeschnitten. Bei Angabe von *BY-VALUE wird die (implizite) Länge des Elementwertes genommen; maximal 75 Zeichen.

TITLE = *BY-ELEMENT-NAME / <c-string 1..76>
Text zur Erläuterung des Elementwertes. Maximale Länge wie bei LENGTH=.... angegeben. Ist LENGTH=*BY-VALUE vereinbart, wird der angegebene Text abgeschnitten, wenn er die maximale Länge überschreitet. Bei Angabe von *BY-ELEMENT-NAME wird der Elementname als Text geschrieben.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler garantierte Meldungen: SDP1120, SPD1121, SPD1122
	64	SPD0259	Ausführung abgebrochen; Selektion ignoriert.
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Ausschnitt eines Prozedurablaufs (Löschen bzw. Anzeigen von Dateien):

Die S-Variable OPS enthält die Ausgabe des Kommandos /SHOW-FILE-ATTRIBUTES. Ausgewählte Unterelemente dieser Variablen (F-NAME, F-SIZE, CRE-DATE) werden mit dem Kommando /SELECT-VARIABLE-ELEMENTS auf den Bildschirm ausgegeben. Bestimmte Elemente werden mit „d“ und „s“ markiert und in die S-Variable SELECTED übertragen, einschließlich dem Markierungszeichen. Die unter der Abbildung stehende (kleine) Prozedur löscht die mit „d“ markierten Dateien und zeigt die Inhalte der mit „s“ markierten Dateien an.

```
/declare-variable OPS(type=*structure),multiple-elements=*list
/declare-variable SELECTED(type=*structure),multiple-elements=*list
/declare-variable STRUC(type=*structure)

/execute-cmd (show-file-attributes *all,select=*by-attributes(-
/             size=*interval(from=500),information=*all)),-
/             structure-output=OPS,text-output=*NONE

/select-variable-elements from-variable=OPS,-
/          ,to-variable=SELECTED(selection-code=*yes),-
/          header-line='Please select files to be deleted (d) or shown (s)',-
/          displayed-elements=(F-NAME(LENGTH=54,TITLE='File name'),-
/          F-SIZE(TITLE='Size'),CRE-DATE)
```

```
Please select files to be deleted (d) or shown (s)
--------------------------------------------------------------------------------
    File name                       Size    CRE-DATE
(d) A1                              501     12-04-2007
(d) A2                              502     12-04-2007
( ) A3                              503     12-04-2007
( ) A4                              503     12-04-2007
(s) A5                              504     12-04-2007
( ) A6                              505     12-04-2007
(s) A7                              506     12-04-2007
( ) A8                              507     12-04-2007
( ) A9                              508     12-04-2007
( ) B1                              509     12-04-2007
(d) B2                              510     12-04-2007
--------------------------------------------------------------------------------
NEXT= +
      *EXECUTE or *CANCEL or *NONE or *ALL or +

LTG                                                                    TAST
```

```
/for STRUC=*list(SELECTED)
/  struc.selection-code = upper-case(struc.selection-code)
/  if struc.selection-code == 'D'
/     delete-file &(STRUC.F-NAME)
/  end-if'
/  if struc.selection-code == 'S'
/     show-file &(STRUC.F-NAME)
/  end-if'
/end-for
```

SEND-DATA
Datensatz an ein Programm übergeben

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

SEND-DATA sollte immer dann verwendet werden, wenn Datensätze und Kommandos gemischt werden sollen. Darüber hinaus bietet SEND-DATA folgende Vorteile:

– Datensätze können mit einer Marke versehen werden.
– Datensätze können Kommentare enthalten.
– Datensätze können mehrzeilig angegeben werden (Fortsetzungsbehandlung).
– Datensätze können mit einem Schrägstrich beginnen.
– Über eine einheitliche Oberfläche können im Kommando Daten und EOF-Bedingung erzeugt werden.

Das Auftreten eines SEND-DATA-Kommandos im „Datenstrom“ löst nicht wie bei anderen Kommandos implizit eine EOF-Bedingung aus, sondern steuert dies über den Operanden RECORD.

Format

SEND-DATA
RECORD = ***EOF** / <text 0..1800 with-low *string-expr*>

Operandenbeschreibung

RECORD = *EOF
Setzt die EOF-Bedingung.

RECORD = <text 0..1800 with-low *string-expr*>
String-Ausdruck. Die Auswertung des Ausdrucks ergibt den Datensatz.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

SEND-STMT
Anweisungssatz an ein Programm übergeben

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

SEND-STMT sollte immer dann verwendet werden, wenn Kommandos und Anweisungen gemischt werden. Ein SEND-STMT-Kommando im Datenstrom löst analog zu SEND-DATA keine implizite EOF-Bedingung aus.

Format

SEND-STMT
RECORD = ***EOF** / <text 0..1800 with-low *string-expr*>

Operandenbeschreibung

RECORD = *EOF
Setzt die EOF-Bedingung, das heißt: Ende der Anweisungseingabe.

RECORD = <text 0..1800 with-low *string-expr*>
String-Ausdruck. Die Auswertung des Ausdrucks ergibt die Anweisung.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

SET-PROCEDURE-OPTIONS
Prozedureigenschaften festlegen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando SET-PROCEDURE-OPTIONS kann der Benutzer die Eigenschaften einer *S-Prozedur* festlegen. Das Kommando ist optional. Wird es verwendet, so muss es das *erste* Kommando des Prozedurkopfs sein. Wird es nicht verwendet, sind die Eigenschaften gemäß den Voreinstellungen von SDF-P vereinbart. Folgende Einstellungen können im Kommando SET-PROCEDURE-OPTIONS getroffen werden (in Klammern die Voreinstellung von SDF-P):

- Zugelassener Prozeduraufruf (CALLER=*ANY)
- Implizite Deklaration von S-Variablen (IMPLICIT-DECLARATION=*YES)
- Umfang der Protokollierung (LOGGING=*YES)
- Unterbrechung der Prozedur (INTERRUPT-ALLOWED=*YES)
- Format der Prozedur (INPUT-FORMAT=*FREE-RECORD-LENGTH)
- Variablenersetzung innerhalb von Datensätzen (DATA-ESCAPE-CHAR=*NONE)
- SYSFILE-Umgebung der aktuellen Prozedurstufe (SYSTEM-FILE-CONTEXT=*STD)
- Fehlerbehandlung, wenn Eingabedaten und Kommandos gemischt werden (DATA-ERROR-HANDLING=*YES)
- Einstellung für die Jobvarialen-Ersetzung (Voreinstellung im Dialog JV-REPLACEMENT=*AFTER-BUILTIN-FUNCTION; in S-Prozeduren JV-REPLACEMENT=*NO)
- Einstellung der Fehlerbehandlung (ERROR-MECHANISM=*SPIN-OFF-COMPATIBLE)
- Unterdrücken ausgewählter SDF-P-Meldungen (SUPPRESS-SDP-MSG=...)

Hinweis

- Voreinstellungen des Kommandos, die in der aktivierten Syntaxdatei modifiziert wurden, gelten für die Prozedur nur bei expliziter Angabe des SET-PROCEDURE-OPTIONS-Kommandos.
- SET-PROCEDURE-OPTIONS darf nur einmal und dann nur als erstes Kommando der Prozedur aufgerufen werden (siehe auch Abschnitt „Prozedurkopf erstellen“ auf Seite 81).

Format

SET-PROCEDURE-**OPT**IONS

CALLER = ***ANY** / ***CALL** / ***INCL**UDE

,**IMPLI**CIT-**DECL**ARATION = ***YES** / ***NO**

,**LOG**GING-**ALLOW**ED = ***PAR**AMETERS(...) / ***Y**ES / ***NO**

***PAR**AMETERS(...)

CMD = *YES / ***NO**

,**DATA** = *YES / ***NO**

,**INTERRUPT-ALLOW**ED = ***YES** / ***NO**

,**INP**UT-**FORM**AT = ***FREE-REC**ORD-**LENGTH** / ***BY-SDF-OPT**ION

,**DATA-ESC**APE-**CHAR** = ***NONE** / '&&' / '#' / '*' / '@' / '$' / ***STD**

,**SYS**TEM-**FILE-CONTEXT** = ***STD** / ***SAME-AS-CALL**ER / ***OWN**

,**DATA-ERROR**-HANDLING = ***YES** / ***NO**

,**JV-REPLACEMENT** = ***NONE** / ***AFTER-BUILTIN-FUNCTION**

,**ERROR-MECHANISM** = ***SPIN-OFF-COMPATIBLE** / ***BY-RETURNCODE**

,**SUP**PRESS-**SDP-MSG** = ***NONE** / list-poss(2000): <alphanum-name 7..7>

Operandenbeschreibung

CALLER =
Vereinbart, mit welchen Kommandos die Prozedur aufgerufen werden darf.

CALLER = *ANY
Die Prozedur darf sowohl mit CALL-PROCEDURE als auch mit INCLUDE-PROCEDURE aufgerufen werden (ENTER-PROCEDURE setzt intern ein CALL-PROCEDURE-Kommando ab).

CALLER = *CALL
Die Prozedur darf nur mit CALL-PROCEDURE (oder ENTER-PROCEDURE) aufgerufen werden.

CALLER = *INCLUDE
Die Prozedur darf nur mit INCLUDE-PROCEDURE aufgerufen werden.

IMPLICIT-DECLARATION = *YES / *NO
Gibt an, ob implizite Deklaration von Variablen erlaubt ist.

LOGGING-ALLOWED =
Gibt an, ob Protokollierung für die Prozedur erlaubt ist und was protokolliert werden darf.

LOGGING-ALLOWED = *PARAMETERS(...)
Legt in den nachfolgenden Angaben fest, was protokolliert werden darf.

CMD = *YES / *NO
Gibt an, ob Kommandos protokolliert werden dürfen.

DATA = *YES / *NO
Gibt an, ob Daten protokolliert werden dürfen.

LOGGING-ALLOWED = *YES
Protokollierung ist erlaubt, das heißt, sowohl Kommandos als auch Daten dürfen protokolliert werden.

LOGGING-ALLOWED = *NO
Protokollierung ist nicht erlaubt.

INTERRUPT-ALLOWED = *YES
Gibt an, dass die Prozedur mit der Funktionstaste K2 unterbrochen und mit dem Kommando RESUME-PROCEDURE fortgesetzt werden darf.

INTERRUPT-ALLOWED = *NO
Gibt an, dass die Prozedur nicht mit der Funktionstaste K2 unterbrochen werden kann.

INPUT-FORMAT =
Bezeichnet das Eingabeformat der Prozedursätze.

INPUT-FORMAT = *FREE-RECORD-LENGTH
Sätze werden in voller Länge interpretiert. Auf das Fortsetzungszeichen dürfen bis zum Satzende nur noch Leerzeichen folgen.

INPUT-FORMAT = *BY-SDF-OPTION
Das Eingabeformat der Prozedursätze ist über das Kommando MODIFY-SDF-OPTIONS, Operand CONTINUATION festgelegt (siehe Handbuch „Kommandos, Bd. 1-5“ [3]).
Der Prozedursatz, der das Kommando SET-PROCEDURE-OPTIONS enthält, wird grundsätzlich in voller Länge interpretiert (entspricht der Angabe FREE-RECORD-LENGTH).

DATA-ESCAPE-CHAR =
Legt das Escape-Zeichen in Datensätzen fest. Das Escape-Zeichen ist das Zeichen, das die Ausdrucksersetzung einleitet.

DATA-ESCAPE-CHAR = *NONE
In Datensätzen soll keine Ausdrucksersetzung erfolgen.

DATA-ESCAPE-CHAR = '&&' / '#' / '*' / '@' / '$'
Legt ein Escape-Zeichen fest.

DATA-ESCAPE-CHAR = *STD
Als Escape-Zeichen gilt das Zeichen &.

SYSTEM-FILE-CONTEXT =
Gibt an, mit welcher Systemdatei-Umgebung die Prozedur ablaufen soll.

SYSTEM-FILE-CONTEXT = *STD
Es wird eine eigene Systemdatei-Umgebung eingerichtet. Die Systemdatei SYSDTA wird automatisch der Systemdatei SYSCMD (also der Prozedurdatei) zugewiesen. Für die anderen Systemdateien werden die Zuweisungen des Aufrufers übernommen. Die Änderung von Zuweisungen gilt nur innerhalb der aktuellen Prozedurstufe. Bei Prozedurende erhalten die Systemdateien wieder die Zuweisungen des Aufrufers.

SYSTEM-FILE-CONTEXT = *SAME-AS-CALLER
Die Prozedur läuft in der Systemdatei-Umgebung des Aufrufers ab. Die Änderung von Zuweisungen innerhalb der aktuellen Prozedurstufe wirkt sich deshalb *immer* auf die Systemdatei-Umgebung des Aufrufers aus.

SYSTEM-FILE-CONTEXT = *OWN
Es wird eine eigene Systemdatei-Umgebung eingerichtet. Für *alle* Systemdateien werden die Zuweisungen des Aufrufers übernommen (auch für SYSDTA!). Die Änderung von Zuweisungen gilt nur innerhalb der aktuellen Prozedurstufe. Bei Prozedurende erhalten die Systemdateien wieder die Zuweisungen des Aufrufers.
Die Einstellung *OWN entspricht auch dem bisherigen Verhalten bei *Nicht-S-Prozeduren*.

DATA-ERROR-HANDLING =
Vereinbart, dass in bestimmten Fällen die Fehlerbehandlung ausgelöst wird.

DATA-ERROR-HANDLING = *YES
Vereinbart, dass in folgenden Fällen die Fehlerbehandlung ausgelöst wird:
– wenn eine Prozedurzeile Daten enthält, wo Kommandos erwartet werden
– wenn in Datenzeilen eine geforderte Ausdrucksersetzung nicht durchgeführt werden kann
– wenn ein Datensatz ein einzelnes Escape-Zeichen enthält

DATA-ERROR-HANDLING = *NO
Es wird keine Fehlerbehandlung in den oben beschriebenen Fällen ausgelöst; &varname bleibt in den Daten unverändert stehen, falls varname weder als Funktion noch als Variable bekannt ist.

JV-REPLACEMENT =
Gibt an, ob bei der Ausdrucksersetzung auch eine Jobvariablenersetzung stattfinden kann.

JV-REPLACEMENT = *NONE
Bei der Ausdrucksersetzung werden Namen nicht als Jobvariablen-Namen interpretiert.

JV-REPLACEMENT = *AFTER-BUILTIN-FUNCTION
Bei einem Ausdruck der Form &(name) wird name als Jobvariablen-Name interpretiert, wenn es keine Variable oder vordefinierte Funktion dieses Namens gibt. Dieser Operandenwert soll ein zu Nicht-S-Prozeduren kompatibles Verhalten bei der Ausdrucksersetzung ermöglichen. Da der Jobvariablenname jederzeit durch neue Variablendeklarationen oder vordefinierte Funktionen überlagert werden kann, wird dem Anwender dringend empfohlen, diesen Operandenwert nicht einzustellen, sondern die Jobvariablenersetzung durch Verwendung der vordefinierte Funktion JV() durchzuführen. Also: &(JV('name')).

ERROR-MECHANISM =
Gibt an, ob die Fehlerbehandlung kompatibel zum Spin-Off-Verhalten von Nicht-S-Prozeduren ausgelöst oder ob Subcode1 ungleich Null berücksichtigt wird. Die Einstellung ist für die Fehlerbehandlung von Anweisungen wirkungslos.

ERROR-MECHANISM = *SPIN-OFF-COMPATIBLE
Die Fehlerbehandlung wird kompatibel zum bisherigen Spin-Off-Verhalten ausgelöst. Der Subcode1 wird **nicht** berücksichtigt. (Damit wird sichergestellt, dass das Fehlerverhalten von S-Prozeduren, die bereits in BS2000 V10.0 erstellt wurden, kompatibel bleibt).

ERROR-MECHANISM = *BY-RETURNCODE
Die Fehlerbehandlung wird ausgelöst, wenn der Subcode1 des letzten Kommando-Returncodes ungleich Null ist. Das Spin-Off-Verhalten wird nicht berücksichtigt. Bei der Einstellung *BY-RETURNCODE muss die Fehlerbehandlung in S-Prozeduren auf die Kommando-Return-codes der Kommandos abgestimmt werden.

Hinweis

Um sich vor späteren Änderungen in der Anwender-Syntaxdatei bezüglich der Voreinstellung zu schützen, sollte der gewählte Wert explizit in der Prozedur angegeben werden.

SUPPRESS-SDP-MSG =
Bestimmt, ob die Ausgabe bestimmter SDF-P-Meldungen (Meldungsklasse SDP) unterdrückt werden soll. Die Option gilt nur in der aufrufenden Prozedur (wird nicht weitervererbt).

SUPPRESS-SDP-MSG = *NONE
Die Meldungsausgabe wird nicht unterdrückt; alle SDF-P-Meldungen werden ausgegeben.

SUPPRESS-SDP-MSG = list-poss(2000): <alphanum-name 7..7>
Menge der SDF-P-Meldungen, die nicht ausgegeben werden sollen.

Kommando-Returncode

Das Kommando SET-PROCEDURE-OPTIONS kann nur als erstes Kommando des Prozedurkopfes einer S-Prozedur verwendet werden. Fehler im Prozedurkopf erkennt SDF-P bei der Voranalyse und beendet den Prozeduraufruf.

Die Kommando-Returncodes können nur auftreten, wenn das Kommando außerhalb des Prozedurkopf verwendet wird.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Kommando MODIFY-PROCEDURE-OPTIONS, Seite 701.

SET-VARIABLE
Variablen einen Wert zuweisen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Mit dem Kommando SET-VARIABLE werden einfachen oder zusammengesetzten Variablen Werte zugewiesen.

Bei Zuweisung von einfachen Variablen oder Variablenelementen müssen die Datentypen sich entsprechen. Bei globaler Zuweisung von zusammengesetzen Variablen muss auch die Struktur der Variablen übereinstimmen.

Stehen auf beiden Seiten der Zuweisung zusammengesetzte Variablen, ist auf Folgendes zu achten:
– Werden Variablenelemente überschrieben?
– Wird die zusammengesetzte Variable erweitert?
– Werden Variablenelemente der rechts stehenden Variablen ignoriert?

Bei der Zuweisung zusammengesetzter Variablen vom Typ Array ist zu beachten, dass die Reihenfolge, in der Inhalte von Array-Elementen anderen Elementen zugewiesen werden, bestimmt wird durch die Reihenfolge der Array-Elemente auf der rechten Seite der Zuweisung. Bei Listenvariablen kann Anfang und Ende eines Bereichs von Listenelementen vereinbart werden.

Bei der Kommandoeingabe reicht es aus, wenn statt

`/SET-VARIABLE <variable`$_1$`> = <variable`$_2$`> / <text>`

geschrieben wird

`/<variable`$_1$`> = <variable`$_2$`> / <text>`

Die Schreibweise ohne den Kommandonamen ist auch aus Performance-Gründen zu empfehlen (siehe Seite 278).

Hinweis

Die Operanden des Kommandos SET-VARIABLE werden nur von SDF-P ausgewertet und sind, wie nachfolgend dargestellt, einzugeben. Für die Operanden gelten die SDF-Abkürzungsregeln. SDF-Funktionen, wie Auskunft über mögliche Operandenwerte oder Korrekturdialog, sind auf Operandenebene nicht verfügbar. Im geführten Dialog stellt SDF nur ein Eingabefeld mit „# =“ zur Verfügung.

Format

SET-VARIABLE Kurzname: **STV**

<composed-name$_1$ 1..255> = <text 0..1800 with-low *expr*> / <composed-name$_2$ 1..255> **/**
***STRING-TO-VAR**IABLE(...) / ***LIST**(...)

***STRING-TO-VAR**IABLE(...)**)**
STRING = <text 0..1800 with-low *expr*>
,**VALUE-TYPE** = ***STD** / ***STRING**

***LIST(...)**
LIST-NAME = <composed-name 1..255>
,**FROM-INDEX** = ***FIRST** / ***LAST** / <integer 1..2147483647>
,**NUMBER-OF-ELEM**ENTS = **1** / ***REST** / <integer 1..2147483647>

,WRITE-MODE = ***REPLAC**E / ***MERGE** / ***EXTEND** / ***PREFIX**

Operandenbeschreibung

<composed-name$_1$ 1..255> =
Bezeichnet die Variable, der ein Wert oder Inhalt einer anderen Variablen zugewiesen wird. Die Variable <composed-name$_1$> kann eine einfache oder zusammengesetzte Variable sein.
Einer einfachen Variablen wird ein Wert zugewiesen, der durch einen Ausdruck bestimmt wird. Der Ausdruck muss ein Ergebnis liefern, das dem Datentyp entspricht, mit dem die Variable deklariert wurde.
Die Regeln für die Zuweisung an zusammengesetzte Variablen sind in der Tabelle am Ende dieser Kommandobeschreibung zusammengefasst.
Ist die Variable über den Behälter-Mechanismus mit einer Jobvariablen verknüpft, muss das Ergebnis des Ausdrucks den entsprechenden Bedingungen genügen: Datentyp STRING, maximale Länge 255 Byte.

<composed-name$_1$ 1..255> = <text 0..1800 with-low *expr*>
Der Variablen composed-name$_1$ wird ein Wert zugewiesen, der durch einen Ausdruck bestimmt wird.

<composed-name$_1$ 1..255> = <composed-name$_2$ 1..255>
Der Variablen composed-name$_1$ wird der Inhalt der Variablen composed-name$_2$ zugewiesen; composed-name$_2$ ist eine einfache oder eine zusammengesetzte Variable.
Ist composed-name$_1$ ein Listenelement, muss dieses Element bereits vorhanden sein, es kann nicht implizit angelegt werden, auch dann nicht, wenn implizite Deklaration erlaubt ist. (Ausnahme: Mit /liste# = <wert> kann der Listenkopf angelegt werden.) Die Regeln für die Zuweisung von zusammengesetzten Variablen sind in der Tabelle am Ende dieser Operandenbeschreibung dargestellt.

<composed-name$_1$ 1..255> = *STRING-TO-VARIABLE(...)
Legt fest, dass ein String gemäß der Konvertierungsregeln (siehe Abschnitt „SDF-Kommando-Strings zu S-Variablen konvertieren und umgekehrt“ auf Seite 186) in eine S-Variable vom Typ Struktur konvertiert wird.

STRING = <text 0..1800 with-low *expr*>
Eingabe-String, der in eine S-Variable konvertiert werden soll.

VALUE-TYPE =
Legt fest, ob der Eingabestring wertabhängig oder generell in den Typ String konvertiert werden soll. Siehe hierzu Konvertierungsregel 4 auf Seite 186.

VALUE-TYPE = *STD
Der Eingabe-String wird wertabhängig (String/Integer/Boolean) konvertiert. Für leere oder aus Leerzeichen bestehende Listenelemente werden keine entsprechenden Elemente von Listenvariablen angelegt.

VALUE-TYPE = *STRING
Der Eingabe-String wird generell in Variable des Typs String konvertiert. Für leere oder aus Leerzeichen bestehende Listenelemente werden entsprechende Elemente von Listenvariablen angelegt.

<composed-name$_1$ 1..255> = *LIST(...)
Der Variablen <composed-name$_1$> werden Elemente einer Listenvariablen zugewiesen. Je nach Anzahl zugewiesener Listenelemente muss <composed-name$_1$> eine einfache oder zusammengesetzte Variable sein.

LIST-NAME = <composed-name 1..255>
Name der Listenvariablen.

FROM-INDEX = *FIRST / *LAST / <integer 1..2147483647>
Index des Elementes der Listenvariablen, mit dem beginnend eine spezifizierte Anzahl von Listenelementen der Variablen <composed-name$_1$> zugewiesen werden.
*FIRST: Die Zuweisung beginnt mit dem ersten Element der Liste; Voreinstellung.
Die Angabe *LAST weist genau das letzte Element der Liste zu. Der Operand NUMBER-OF-ELEMENTS wird in diesem Fall ignoriert.

NUMBER-OF-ELEMENTS = 1 / *REST / <integer 1..2147483647>
Anzahl der Listenelemente, die zugewiesen werden. Voreinstellung: Ein Element wird zugewiesen.
Die Angabe *REST weist alle Elemente vom angegebenen Startelement (Operand FROM-INDEX) bis zum letzten Element der Liste zu.

WRITE-MODE =
Legt fest, wohin der neue Variableninhalt bei der Zuweisung gebracht wird.
Die Tabelle am Ende dieser Operandenbeschreibung zeigt, welche Variablen und Werte bei der Zuweisung mit den Operandenwerten von WRITE-MODE = kombiniert werden können.

WRITE-MODE = *REPLACE

Der Variablenname auf der linken Seite der Zuweisung muss eine einfache oder zusammengesetzte Variable bezeichnen.
Die Variable bzw. die Elemente der zusammengesetzten Variablen müssen überschreibbar sein. (Eine Variable ist in der Regel nur dann nicht überschreibbar, wenn sie als Behälter eine Jobvariable mit Schreibschutzkennwort besitzt und das Kennwort nicht in der aktuellen Kennwortliste enthalten ist.)
Der Inhalt der Variablen, die links vom Gleichheitszeichen steht, wird implizit mit FREE-VARIABLE gelöscht; anschließend wird die Variable mit dem Wert, der sich aus der Angabe rechts vom Gleichheitszeichen ergibt, überschrieben.

Arrays

Auf beiden Seiten des Gleichheitszeichens muss ein Array stehen.
Die Elemente des links stehenden Arrays werden von den Elementen des rechts stehenden der Reihe nach überschrieben. Enthält der rechts stehende Array mehr Elemente, wird der links stehende erweitert. Enthält der rechts stehende Array weniger Elemente, werden die überzähligen Elemente des links stehenden Arrays gelöscht (implizites FREE-VARIABLE).

Listen

Auf beiden Seiten des Gleichheitszeichens muss eine Liste stehen.
Die Elemente der links stehenden Listen werden von den Elementen der rechts stehenden der Reihe nach überschrieben. Enthält die rechts stehende Liste mehr Elemente, wird die links stehende erweitert. Enthält die rechts stehende Liste weniger Elemente, werden die überzähligen Elemente der links stehenden Liste gelöscht (implizites FREE-VARIABLE).

Strukturen

Auf beiden Seiten des Gleichheitszeichens muss eine Struktur angegeben sein.

– Zuweisung an eine statische Struktur: Elemente der links stehenden Struktur werden mit den Inhalten der Elemente auf der rechten Seite des Gleichheitszeichens überschrieben, wenn die Elemente den gleichen Elementnamen haben. Enthält die rechts stehende Struktur Elemente, für die es kein „Pendant“ in der links stehenden Struktur gibt, werden diese Elemente ignoriert.
– Zuweisung an eine dynamische Struktur: Implizit werden alle Elemente der rechts stehenden Struktur als Elemente der links stehenden Struktur deklariert.

WRITE-MODE = *MERGE

Der Variablenname auf der linken Seite der Zuweisung muss eine zusammengesetzte Variable vom Typ Array oder Struktur bezeichnen.
Der zusammengesetzten Variablen links vom Gleichheitszeichen wird der Inhalt der rechts stehenden zusammengesetzten Variablen zugewiesen.
Falls die zusammengesetzten Variablen links und rechts vom Gleichheitszeichen identisch sind, ist die Wirkung von WRITE-MODE = *MERGE gleich der von WRITE-MODE = *REPLACE.

Arrays
Auf beiden Seiten des Gleichheitszeichens muss ein Array stehen. Wenn die beteiligten Arrays Elemente mit dem gleichen Arrayindex enthalten, gilt:
Dem Element des links stehenden Arrays wird der Inhalt des rechts stehenden Arrayelements zugewiesen, das den gleichen Arrayindex hat. Arrayelemente, für die es auf der linken Seite der Zuweisung kein „Pendant“ mit gleichem Arrayindex gibt, werden angelegt.

Strukturen
Auf beiden Seiten des Gleichheitszeichens muss eine Struktur angegeben sein.

– Zuweisung an eine statische Struktur: Elemente der links stehenden Struktur werden mit den Inhalten der Elemente auf der rechten Seite des Gleichheitszeichens überschrieben, wenn die Elemente den gleichen Elementnamen haben. Enthält die rechts stehende Struktur Elemente, für die es kein „Pendant“ in der links stehenden Struktur gibt, werden diese Elemente ignoriert. Enthält die links stehende Struktur Elemente, für die es kein „Pendant“ in der rechts stehenden Struktur gibt, dann bleiben diese Elemente unbeeinflusst.
– Zuweisung an eine dynamische Struktur: Implizit werden alle Elemente der rechts stehenden Struktur als Elemente der links stehenden Struktur deklariert.

WRITE-MODE = *EXTEND
Nur für Variablen vom Typ *Liste.
Die rechte Seite der Zuweisung muss ein Ausdruck sein oder eine zusammengesetzte Variable vom Typ Liste bezeichnen.

Die Liste auf der linken Seite der Zuweisung wird erweitert:
– Enthält die Zuweisung auf der rechten Seite einen Ausdruck, so wird die Liste um ein Element erweitert; diesem Element wird das Ergebnis des Ausdrucks zugewiesen.
– Enthält die Zuweisung auf der rechten Seite den Variablennamen einer Liste, so wird diese Liste (rechts) an die Liste (links) angehängt: die ursprüngliche Liste wird um die entsprechende Anzahl Elemente erweitert; diesen Elementen wird der Reihe nach der Inhalt der Elemente der rechts stehenden Liste zugewiesen.

WRITE-MODE = *PREFIX
Nur für Variablen vom Typ *Liste.
Die rechte Seite der Zuweisung muss ein Ausdruck sein oder eine zusammengesetzte Variable vom Typ Liste bezeichnen.

Die Liste auf der linken Seite der Zuweisung wird erweitert:
– Enthält die Zuweisung auf der rechten Seite einen Ausdruck, wird vor das bisher erste Element der Liste ein neues Element eingefügt; diesem Element wird das Ergebnis des Ausdrucks zugewiesen.
– Enthält die Zuweisung auf der rechten Seite den Variablennamen einer Liste, wird diese (rechte) Liste in die (linke) Liste eingefügt, vor das bisher erste Element. Die (linke) Liste wird nach vorn um so viele Elemente erweitert, wie die (rechte) Liste enthält. Diesen neuen Elementen wird der Reihe nach der Inhalt der Elemente der rechts stehenden Liste zugewiesen.

Kommando-Returncode

Während der Zuweisung von Strukturen, Arrays oder Listen ist es möglich, dass ein Teil des Kommandos abgearbeitet und ausgeführt wurde, bevor ein Fehler auftrat. In diesem Fall ist das Resultat des Kommandos nicht garantiert.

(SC2) SC1	Maincode	Bedeutung / garantierte Meldungen
0	CMD0001	Ohne Fehler
1	CMD0202	Syntaxfehler
1	SDP0118	Kommando im falschen Kontext
3	CMD2203	Falsche Syntaxdatei
32	CMD0221	Systemfehler (interner Fehler)
64	SDP0091	Semantikfehler garantierte Meldung: SDP1030
130	SDP0099	Kein Adressraum mehr verfügbar

Zulässige Kombinationen von Variablentypen und Operanden

Variable	**Wert/Variable**	**WRITE-MODE =**			
(links)	(rechts)	***REPLACE**	***MERGE**	***EXTEND**	***PREFIX**
einfache Variable	Ausdruck	x	-	-	-
	einfache Variable	x	-	-	-
Array	Ausdruck	-	-	-	-
	einfache Variable	-	-	-	-
	Array	x	x	-	-
	Liste	-	-	-	-
	Struktur	-	-	-	-
Liste	Ausdruck	-	-	x	x
	einfache Variable	-	-	x	x
	Array	-	-	x	x
	Liste	x	-	x	x
	Struktur	-	-	x	x
Struktur	Ausdruck	-	-	-	-
	einfache Variable	-	-	-	-
	Array	-	-	-	-
	Liste	-	-	-	-
	Struktur	x	x	-	-

Legende

x Kombination zulässig, Auswirkung siehe Beschreibung des jeweiligen Operanden

- Kombination führt zum Fehler

Beispiel

```
/SET-VARIABLE A = B
/A = B
```

Beide Zuweisungen sind gleichwertig: Der Variablen A wird der Inhalt der Variablen B zugewiesen.

```
/SET-VARIABLE PERSON = 'HUGO'
/PERSON = 'HUGO'
```

In beiden Zuweisungen wird der Variablen PERSON der String 'HUGO' zugewiesen.

```
/DECLARE-VARIABLE SUMME
/SET-VARIABLE SUMME = -(1 + 3) * 4
/SUMME = -(1 + 3) * 4
```

Mit DECLARE-VARIABLE wird eine Variable SUMME deklariert, standardmäßig mit dem Datentyp ANY. Die beiden folgenden Zuweisungen sind gleichwertig: der Variablen SUMME wird der Integer-Wert -16 zugewiesen.

Beispiel: Arrays

```
/DECLARE-VARIABLE A, MULTIPLE-ELEMENTS = *ARRAY
/DECLARE-VARIABLE B, MULTIPLE-ELEMENTS = *ARRAY
/SET-VARIABLE A#1 = 5
/SET-VARIABLE A#3 = 3
/SET-VARIABLE B#5 = 1
/SET-VARIABLE B = A
/SHOW-VARIABLE B
```

Ausgabe

```
B#1 = 5
B#3 = 3
```

Variante

```
/SET-VARIABLE A#4 = 5
/SET-VARIABLE B#5 = 1
/SET-VARIABLE A#6 = 3
/SET-VARIABLE B = A, WRITE-MODE = *MERGE
/SHOW-VARIABLE B
```

Ausgabe

```
B#4 = 5
B#5 = 1
B#6 = 3
```

Beispiel: Strukturen

```
/DECLARE-VARIABLE S1(TYPE = *STRUCTURE(*BY-SYSCMD))
/ BEGIN-STRUCTURE
/ DECLARE-ELEMENT X(INITIAL-VALUE = 11)
/ DECLARE-ELEMENT Y(INITIAL-VALUE = 12)
/ DECLARE-ELEMENT Z(INITIAL-VALUE = 13)
/ END-STRUCTURE
/DECLARE-VARIABLE S2(TYPE = *STRUCTURE(*DYNAMIC))
/ DECLARE-ELEMENT S2.X('AB')
/ DECLARE-ELEMENT S2.Y('CD')
/ SET-VARIABLE S1 = S2
/SHOW-VARIABLE S1, SELECT=*BY-ATTRIBUTES(INITIALIZATION=*ANY)
```

Ausgabe

```
S1.X = AB
S1.Y = CD
S1.Z = *NO-INIT
```

Variante

```
/DECLARE-VARIABLE S1(TYPE = *STRUCTURE(*BY-SYSCMD))
/ BEGIN-STRUCTURE
/ DECLARE-ELEMENT X(INIT = 11)
/ DECLARE-ELEMENT Y(INIT = 12)
/ DECLARE-ELEMENT Z(INIT = 13)
/ END-STRUCTURE
/DECLARE-VARIABLE S2(TYPE = *STRUCTURE(*DYNAMIC))
/ DECLARE-ELEMENT S2.X('AB')
/ DECLARE-ELEMENT S2.Y('CD')
/ SET-VARIABLE S2 = S1
/SHOW-VARIABLE S2
```

Ausgabe

```
S2.X = 11
S2.Y = 12
S2.Z = 13
```

Beispiel: Listen von Strukturen verwalten

Das folgende Beipiel zeigt, wie Listen von Strukturen verwaltet werden können: Es wird eine Liste von Strukturen erzeugt, die für jeden Benutzer den Namen und die Telefonnummer enthält. Wenn die Liste vollständig ist, wird sie in eine Datei kopiert.

```
/BEGIN-STRUCTURE MYLAYOUT
/  DECLARE-ELEMENT NAME(TYPE=*STRING)
/  DECLARE-ELEMENT TELEFONNUMMER(TYPE=*STRING)
/END-STRUCTURE
```

```
/DECLARE-VARIABLE LIST-OF-STRUCT(TYPE=*STRUCTURE(MYLAYOUT))-
/                  ,MULTIPLE-ELEMENTS=*LIST
/DECLARE-VARIABLE STRUCT(TYPE=*STRUCTURE(MYLAYOUT))
/READ-VARIABLE VARIABLE-NAME = STRUCT.NAME,INPUT=*TERMINAL-
/          (PROMPT-STRING = 'GEBEN SIE EINEN BENUTZERNAMEN (ENDE MIT '''')')
/WHILE (STRUCT.NAME <>'')
/   READ-VARIABLE VARIABLE-NAME = STRUCT.TELEFONNUMMER,INPUT=*TERMINAL-
/         (PROMPT-STRING = 'GEBEN SIE EINE BENUTZER-TELEFON#')
/   LIST-OF-STRUCT=STRUCT,WRITE-MODE=*EXTEND
/   READ-VARIABLE VARIABLE-NAME = STRUCT.NAME,INPUT=*TERMINAL-
/          (PROMPT-STRING = 'GEBEN SIE EINEN BENUTZERNAMEN (ENDE MIT '''')')
/END-WHILE
/SHOW-VARIABLE LIST-OF-STRUCT
```

Beispiele: SDF-Kommando-String konvertieren

Beispiel 1

```
/DECLARE-VARIABLE MYSTRUCT(TYPE=*STRUCTURE(*DYNAMIC))
/SET-VARIABLE MYSTRUCT = *STRING-TO-VARIABLE-
/('OPERAND1=VALUE1(OPERAND2=VALUE2)')
/SHOW-VARIABLE MYSTRUCT
```

Ausgabe

```
MYSTRUCT.OPERAND1.SYSSTRUC = VALUE1
MYSTRUCT.OPERAND1.OPERAND2 = VALUE2
```

Beispiel 2

```
/DCV V1(TYPE=*ANY),MULT-ELEM=*LIST
/DCV V2(TYPE=*ANY),MULT-ELEM=*LIST
/S = '(A,B,1, ,,F)'
/V1 = *STR-TO-VAR(S)
/V2 = *STR-TO-VAR(S,VAL-TYPE=*STR)
/SHV V1,VAL=*C-LIT,LIST-INDEX=YES
V1#1 = 'A'
V1#2 = 'B'
V1#3 = 1
V1#4 = 'F'
/SHV V2,VAL=*C-LIT,LIST-INDEX=YES
V2#1 = 'A'
V2#2 = 'B'
V2#3 = '1'
V2#4 = ' '
V2#5 = ''
V2#6 = 'F'
```

SHOW-STREAM-ASSIGNMENT
S-Variablenenstrom anzeigen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

SHOW-STREAM-ASSIGNMENT zeigt die aktuelle Zuweisung der angegebenen S-Variablenströme an.

Format

SHOW-STREAM-ASSIGNMENT
STREAM-NAME = ***ALL** / ***STD-STREAMS** / <structured-name 1..20 with-wild(40)> / list-poss(100): <structured-name 1..20> ,**INF**ORMATION = ***CURR**ENT-**ASSIGNMENT** / ***FINAL-DEST**INATION ,**OUTPUT** = ***SYSOUT** / ***SYSLST**

Operandenbeschreibung

STREAM-NAME =
Name des S-Variablenstroms, der angezeigt werden soll.

STREAM-NAME = *ALL
Alle S-Variablenströme, die in der aktuellen Prozedur sichtbar sind, werden aufgelistet. D.h. alle S-Variablenströme, die in der aktuellen Prozedur erzeugt wurden oder von der aufrufenden Prozedur vererbt wurden, werden aufgelistet.

STREAM-NAME = *STD-STREAMS
Alle Standard-Variablenströme, die im System implementiert sind, werden angezeigt. Die Namen dieser Variablenströme haben alle das Prefix „SYS“. D.h. alle SYS-Ströme, die in der Syntaxbeschreibung für den Operanden STREAM-NAME des Kommandos ASSIGN-STREAM aufgelistet sind, werden ausgegeben.

STREAM-NAME = <structured-name 1..20 with-wild(40)>
Alle S-Variablenströme, die dieses Suchmuster erfüllen, werden angezeigt.

STREAM-NAME = list-poss(100): <structured-name 1..20>
Liste von S-Variablenstrom-Namen, die angezeigt werden sollen.

INFORMATION =
Gibt an, welche Information ausgegeben werden muss.

INFORMATION = *CURRENT-ASSIGNMENT
Es wird der Name ausgegeben, der im Operanden TO des Kommandos ASSIGN-STREAM eingestellt ist.
Wenn der Variablenstrom noch einem anderen Variablenstrom-Namen zugewiesen ist, wird dieser Name ausgegeben.

INFORMATION = *FINAL-DESTINATION
Es wird der Name des aktuellen Servers, der mit dem S-Variablenstrom verbunden ist, ausgegeben.
Wenn der Variablenstrom noch einem anderen Variablenstrom-Namen zugewiesen ist, wird die letzte Zuweisung ausgegeben. Wenn der Variablenstrom *STD, *DUMMY, *VAR oder *SERVER zugewiesen ist, wird dieser Wert ausgegeben.

OUTPUT =
Gibt an, wohin die Ausgabe des Kommandos geschickt werden soll.

OUTPUT = *SYSOUT
Die Informationen werden nach SYSOUT ausgegeben. Mit den Kommandos ASSIGN-SYSOUT oder ASSIGN-STREAM kann die Ausgabe auch in eine S-Variable erfolgen oder einem S-Variablenstrom zugewiesen werden.

OUTPUT = *SYSLST
Die Informationen werden nur nach SYSLST geschrieben. Eine strukturierte Ausgabe wird nicht unterstützt.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	32	CMD2009	Fehler während S-Variablen-Ersetzung
	64	CMD0216	Erforderliches Privileg fehlt
	64	OPS0001	Kein Speicher für S-Variablen
	64	SDP0091	Semantikfehler
	64	SDP0517	Angegebener Variablenstrom-Name existiert nicht
	64	SDP0519	Kein Treffer für angegebene Wildcards

Beispiel

Eingabe

```
/DECLARE-VARIABLE OPS-VAR(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
/ASSIGN-STREAM SYSINF,TO=*VARIABLE(OPS-VAR)
/ASSIGN-SYSOUT TO=#ERROR-SYSOUT
/SHOW-STREAM-ASSIGNMENT SYSINF
```

Ausgabe

```
STREAM-NAME = SYSINF
 ASSIGN-LEVEL = 0
 DESTINATION = *VARIABLE
           VARIABLE-NAME = OPS-VAR
                      VAR-MODE = *EXTEND
           RETURN-VARIABLE-NAME = *NONE
           CONTROL-VAR-NAME = *NONE
           RET-CONTROL-VAR-NAME = *NONE
```

Strukturierte Ausgaben

Für die strukturierte Ausgabe in Variablen werden beim Kommando SHOW-STREAM-ASSIGNMENT die folgenden Operanden unterstützt:

– STREAM-NAME (alle Werte)
– INFORMATION (alle Werte)

Weitere Informationen, wie z.B. die Bedingungen für die Belegungen der einzelnen Variablen, sind der nachfolgenden Tabelle zu entnehmen.

Ausgabestruktur

Ausgabe-Information	Name der S-Variablen[1)]	T[2)]	Inhalt	Bedingung
Prozedurlevel	var#.ASS-LEV	I	<integer>	
Art der Listenerweiterung bei Kontroll-Variablen	var#.CONTR-VAR-MODE	S	*EXT *PREFIX	DEST='*VARIABLE'
Name der Kontrollvariablen	var#.CONTR-VAR-NAME	S	*NONE <comp.-name 1..255>	DEST='*VARIABLE'
Ausgabeziel	var#.DEST	S	*DUMMY *SERVER *VAR <struc.-name 1..20>	INF
Art der Listenerweiterung bei Return-Kontroll-Variablen	var#.RET-CONTR-VAR-MODE	S	*EXT *PREFIX	DEST='*VARIABLE'
Name der Return-Kontroll-Variablen	var#.RET-CONTR-VAR-NAME	S	*NONE <comp.-name 1..255>	DEST='*VARIABLE'

Ausgabe-Information	**Name der S-Variablen[1)]**	**T[2)]**	**Inhalt**	**Bedingung**
Art der Listenerweiterung bei Returnvariablen	var#.RET-VAR-MODE	S	*EXT *PREFIX	DEST= '*VARIABLE'
Name der Return-Variablen	var#.RET-VAR-NAME	S	*NONE <comp.-name 1..255>	DEST= '*VARIABLE'
Server-Information	var#.SERVER-INFO	S	*NONE <string 1..1800>	DEST= '*SERVER'
Name des Servers	var#.SERVER-NAME	S	<struc.-name 1..30>	DEST= '*SERVER'
Name des S-Variablenstroms	var#.STREAM-NAME	S	<struc.-name 1..20> SYSVAR SYSMSG SYSINF	
Art der Listenerweiterung bei S-Variablen	var#.VAR-MODE	S	*EXT *PREFIX	DEST= '*VARIABLE'
Name der S-Variablen	var#.VAR-NAME	S	*NONE <comp.-name 1..255>	DEST= '*VARIABLE'

[1)]Die Variablennamen sind alphabetisch geordnet

[2)]In der Spalte T sind die Datentypen aufgeführt: B entspricht Boolean, S entspricht String und I entspricht Integer

SHOW-STRUCTURE-LAYOUT
Elementnamen eines Strukturlayouts ausgeben

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Ausgabemedium: SYSOUT / SYSLST / Datei / Listenvariable / Bibliothekselement

Das Kommando SHOW-STRUCTURE-LAYOUT gibt die bei NAME=.... angegebenen Structurlayouts aus. Das Strukturlayout muss zuvor mit BEGIN-STRUCTURE definiert worden sein.

Hinweis

Statische Strukturen werden mit dem Kommando SHOW-VARIABLE ... , INFO=*PAR (VALUE = *NONE) ausgegeben.

Format

SHOW-STRUCTURE**-LAYOUT** Kurzname: **SHSTRL**

NAME = ***ALL** / <structured-name 1..20 with-wild(40)> / list-poss(2000): <structured-name 1..20>

,**SCOPE** = ***VISIBLE** / ***PROC**EDURE / ***CURR**ENT / ***TASK**

,**OUTPUT** = ***SYSOUT** / ***SYSLST** / <filename 1..54 without-gen-vers>(...) / ***VAR**IABLE(...) / ***LIB**RARY**-ELEM**ENT(...)

<filename 1..54 without-gen-vers>(...)

WRITE-MODE = ***REPLACE** / ***EXTEND**

***VAR**IABLE(...)

VARIABLE**-NAME** = <composed-name 1..20>

,**WRITE-MODE** = ***REPLACE** / ***EXTEND**

***LIB**RARY**-ELEM**ENT(...)

LIBRARY = <filename 1..54 without-vers>

,**ELEM**ENT = <composed-name 1..64>(...)

<composed-name 1..64>(...)

VERSION = ***HIGH**EST**-EXIST**ING / ***UP**PER**-LIM**IT / <composed-name 1..24>

,**TYPE** = **S** / <alphanum-name 1..8>

Operandenbeschreibung

NAME =
Bezeichnet das Strukturlayout.

NAME = *ALL
Wählt alle Strukturlayouts.

NAME = <structured-name 1..20 with-wild(40)>
Name des auszugebenden Strukturlayouts.
Wenn der Name Musterzeichen enthält, werden alle Strukturlayouts ausgegeben, deren Namen dem angegebenen Suchmuster entsprechen. Falls eine Musterzeichenfolge keinem Strukturlayout entspricht, wird die Meldung SPD0519 ausgegeben.

NAME = list-poss(2000): <structured-name 1..20>
Ein oder mehrere Namen von auszugebenden Strukturlayouts. Die Ausgabe erfolgt in der angegebenen Reihenfolge.

SCOPE =
Bezeichnet den Geltungsbereich der auszugebenden Strukturlayouts.

SCOPE = *VISIBLE
Gibt alle sichtbaren Strukturlayouts der aktuellen Prozedur aus.
Ein Strukturlayout ist sichtbar, wenn es nicht von einer Deklaration in einer Include-Prozedur überdeckt ist.

SCOPE = *PROCEDURE
Gibt alle Strukturlayouts aus, auch wenn sie von Deklarationen in Include-Prozeduren überdeckt sind.

SCOPE = *CURRENT
Gibt die aktuellen Strukturlayouts aus: innerhalb einer Call-Prozedur die Strukturlayouts der Prozedur, innerhalb einer Include-Prozedur die Strukturlayouts dieser Include-Prozedur.

SCOPE = *TASK
Gibt die taskglobalen Strukturlayouts aus.

OUTPUT =
Bezeichnet das Ausgabemedium.

OUTPUT = *SYSOUT
Ausgabe nach SYSOUT.

OUTPUT = *SYSLST
Ausgabe nach SYSLST.

OUTPUT = <filename 1..54 without-gen-vers>(...)
Ausgabe in die angegebene SAM-Datei.

WRITE-MODE = *REPLACE
Der bisherige Inhalt der Datei soll überschrieben werden.

WRITE-MODE = *EXTEND
Die Ausgabe soll an den bisherigen Inhalt angehängt werden.

OUTPUT = *VARIABLE(...)
Ausgabe in eine Listenvariable.

VARIABLE-NAME = <structured-name 1..20>
Name der Listenvariablen.

WRITE-MODE = *REPLACE
Der bisherige Inhalt der Listenvariablen soll überschrieben werden.

WRITE-MODE = *EXTEND
Die Listenvariable soll erweitert werden, das heißt, die Ausgabe soll an den bisherigen Inhalt angehängt werden.

OUTPUT = *LIBRARY-ELEMENT(...)
Ausgabe in das Element einer PLAM-Bibliothek.

LIBRARY = <filename 1..54 without-vers>
Name der PLAM-Bibliothek.

ELEMENT = <composed-name 1..64>(...)
Name des Elements.

VERSION =
Bezeichnet die Versionsnummer des Elements.

VERSION = *HIGHEST-EXISTING
Wählt die höchste existierende Versionsnummer.

VERSION = *UPPER-LIMIT
Wählt die höchste mögliche Versionsnummer.

VERSION = <composed-name 1..24>
Wählt die angegebene Versionsnummer.

TYPE = S / <alphanum-name 1..8>
Bezeichnet den Elementtyp.

Kommando-Returncode

Es ist möglich, dass ein Teil des Kommandos schon abgearbeitet und ausgeführt wurde, bevor der Fehler auftrat. In diesem Fall ist das Ergebnis des Kommandos nicht garantiert.

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; kein Layout gefunden
2	0	SDP2000	Warnung: Nicht alle Elemente der Eingabeliste konnten erfolgreich-abgearbeitet werden. garantierte Meldung: SDP2000
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	64	SDP1008	Variable existiert nicht
	64	SDP2001	Keines der Elemente konnte angezeigt werden
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/BEGIN-STRUCTURE STRUCT-01 ————————————————————————————————————————  (1)
/  DECLARE-ELEMENT ( X, Y )
/END-STRUCT STRUCT-01
/BEGIN-STRUCTURE STRUCT-02
/  DECLARE-ELEMENT ( W, A )
/END-STRUCT STRUCT-02
/BEGIN-STRUCTURE STRUCT-03
/  DECLARE-ELEMENT ( U, P )
/END-STRUCTURE STRUCT-03
/SHOW-STRUCTUR-LAYOUT STRUCT-0<1,3> ———————————————————————————————  (2)
STRUCT-01.X
STRUCT-01.Y
STRUCT-03.U
STRUCT-03.P
*END-OF-CMD
```

(1) Die Strukturlayouts `STRUCT-01`, `STRUCT-02`, `STRUCT-03` werden nacheinander definiert.

(2) Mit der Musterzeichenfolge `STRUCT-0<1,3>` wird die Ausgabe der Strukturlayouts `STRUCT-01` und `STRUCT-03` angefordert. Alternativ könnten die beiden Namen auch als Liste `(STRUCT-01,STRUCT-03)` angegeben werden.

SHOW-VARIABLE
Inhalte von Variablen ausgeben

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Ausgabemedium: SYSOUT / SYSLST / Datei / Listenvariable / Bibliothekselement

Ausgabeform:

– Der Inhalt von Variablen mit dem Datentyp INTEGER wird als Zeichenfolge der Zeichen 0-9 ausgegeben, ggf. mit vorangestelltem Minuszeichen.
– Der Inhalt von Variablen mit dem Datentyp BOOLEAN wird als Zeichenkette FALSE oder TRUE ausgegeben.

Die Reihenfolge, in der die Inhalte von Variablenelementen ausgegeben werden, entspricht bei zusammengesetzten Variablen vom Typ STRUKTUR der Reihenfolge der Elementdeklarationen, bei zusammengesetzten Variablen vom Typ ARRAY der numerischen Folge der Arrayindizes. Die Ausgabe beginnt für jede Variable jeweils in einer neuen Zeile.

Format

(Teil 1 von 2)

SHOW-VARIABLE Kurzname: **SHV**

VARIABLE-**NAME** = ***ALL** / ***LIST**(...) /
list-poss(2000): <composed-name 1..255> /<structured-name 1..20 with-wild(40)>

***LIST(...)**

LIST-NAME = <composed-name 1..255>
,**FROM-INDEX** = ***FIRST** / ***LAST** / <integer 1..2147483647>
,**NUMBER-OF-ELEM**ENTS = **1** / ***REST** / <integer 1..2147483647>

,**SEL**ECT = ***BY-ATTR**IBUTES(...)

***BY-ATTR**IBUTES(...)

SCOPE = ***VISIBLE** / ***PROC**EDURE / ***CURR**ENT / ***CURR**ENT-**PARAM**ETERS / ***TASK-VISIBLE** / ***TASK-ALL** / ***CALL**ING-**PROC**EDURES
,**INIT**IALIZATION = ***YES** / ***ANY**

Fortsetzung ➠

(Teil 2 von 2)

```
,INFORMATION = *PARAMETERS(...)
    *PARAMETERS(...)
        VALUE = *WITHOUT-QUOTES / *C-LITERAL / *X-LITERAL / *NONE
        ,NAME = *FULL-NAME (...) / *ELEMENT-NAME (...) / *NONE
            *FULL-NAME(...)
                LIST-INDEX-NUMBER = *NO / *YES
            *ELEMENT-NAME(..)
                LIST-INDEX-NUMBER = *NO / *YES
,OUTPUT = *SYSOUT / *SYSLST / <filename 1..54 without-gen-vers>(...) / *VARIABLE(...) /
          *LIBRARY-ELEMENT(...)
    <filename 1..54 without-gen-vers>(...)
        WRITE-MODE = *REPLACE / *EXTEND
    *VARIABLE(...)
        VARIABLE-NAME = <composed-name 1..20>
        ,WRITE-MODE = *REPLACE / *EXTEND
    *LIBRARY-ELEMENT(...)
        LIBRARY = <filename 1..54 without-vers>
        ,ELEMENT = <composed-name 1..64>(...)
            <composed-name 1..64>(...)
                VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT / <composed-name 1..24>
        ,TYPE = S / <alphanum-name 1..8>
        ,WRITE-MODE = *REPLACE / *EXTEND
```

Operandenbeschreibung

VARIABLE-NAME =
Bezeichnet die auszugebenden Variablen.

VARIABLE-NAME = *ALL
Alle Variablen mit dem unter SCOPE angegebenen Geltungsbereich werden in lexikalischer Reihenfolge ihrer Variablennamen ausgegeben. Elemente von Strukturen werden in der Reihenfolge ihrer Deklaration, Arrayelemente in numerischer Reihenfolge ihrer Arrayindizes ausgegeben.

VARIABLE-NAME = *LIST(...)
Es sollen die Elemente einer Listenvariablen ausgegeben werden.

LIST-NAME = <composed-name 1..255>
Name der Listenvariable.

FROM-INDEX = *FIRST / *LAST / <integer 1..2147483647>
Index des Elementes der Listenvariablen, mit dem die Ausgabe beginnen soll.
*FIRST: Die Ausgabe beginnt mit dem ersten Element der Liste; Voreinstellung.
Die Angabe *LAST gibt genau das letzte Element der Liste aus. Der Operand NUMBER-OF-ELEMENTS wird in diesem Fall ignoriert.

NUMBER-OF-ELEMENTS = 1 / *REST / <integer 1..2147483647>
Anzahl der Listenelemente, die ausgegeben werden sollen. Voreinstellung: Ein Element wird ausgegeben.
Die Angabe *REST gibt alle Elemente vom angegebenen Startelement (Operand FROM-INDEX) bis zum letzten Element der Liste aus.

VARIABLE-NAME = list-poss(2000): <composed-name 1..255>
Namen der auszugebenden Variablen.
Diese werden in der angegebenen Reihenfolge ausgegeben.

VARIABLE-NAME = <structured-name 1..20 with-wild(40)>
Die Variablen, deren Namen das Suchmuster erfüllen, werden in lexikalischer Reihenfolge ihrer Namen ausgegeben.

SELECT = *BY-ATTRIBUTES(...)
Bestimmt die auszugebenden Variablen näher.

SCOPE =
Bezeichnet den Geltungsbereich der auszugebenden Variablen.

SCOPE = *VISIBLE
Gibt alle sichtbaren Variablen aus.
Eine Variable ist sichtbar, wenn sie nicht von einer Deklaration in einer Include-Prozedur überdeckt ist.

SCOPE = *PROCEDURE
Gibt alle Variablen aus, auch wenn sie von Deklarationen in einer Include-Prozedur überdeckt sind.

SCOPE = *CURRENT
Gibt die aktuellen Variablen aus: innerhalb einer Call-Prozedur die Variablen der Call-Prozedur; innerhalb einer Include-Prozedur die Variablen der Include-Prozedur.

SCOPE = *CURRENT-PARAMETERS
Gibt die aktuellen Prozedurparameter aus: innerhalb einer Call-Prozedur die Prozedurparameter der Call-Prozedur; innerhalb einer Include-Prozedur die Prozedurparameter der Include-Prozedur.

SCOPE = *TASK-ALL
Gibt alle taskglobalen Variablen aus.

SCOPE = *TASK-VISIBLE
Gibt die importierten taskglobalen Variablen aus bzw. die taskglobalen Variablen, die in der Prozedur deklariert wurden.

SCOPE = *CALLING-PROCEDURE
Gibt alle Variablen des Geltungsbereichs der aufrufenden Prozedur aus, die mit IMPORT-ALLOWED = *YES deklariert wurden. Dieser Geltungsbereich besteht bei Vordergrund-Prozeduren aus allen aufrufenden Prozeduren ab der Dialogebene, bei Hintergrund-Prozeduren aus allen aufrufenden Prozeduren ab der ersten Prozedur.

INITIALIZATION =
Bezeichnet, ob nicht-initialisierte Variablen ausgegeben werden sollen oder nicht.

INITIALIZATION = *YES
Es werden nur initialisierte Variablen ausgegeben.

INITIALIZATION = *ANY
Es werden alle (egal ob initialisierte oder nicht-initialisierte) Variablen ausgegeben.

INFORMATION = *PARAMETERS(...)
Bezeichnet die Informationen, die ausgegeben werden.

VALUE =
Gibt an, ob der Wert der Variablen ausgegeben werden soll und in welcher Form.

VALUE = *WITHOUT-QUOTES
Gibt Variablen mit dem Datentyp `STRING` ohne Hochkomma aus.

VALUE = *C-LITERAL
Gibt Variablen mit dem Datentyp `STRING` als C-Literal aus. Falls nicht initialisiert, wird der String '*NO-INIT' ausgegeben.

VALUE = *X-LITERAL
Gibt Variablen mit dem Datentyp `STRING` als X-Literal aus. Falls nicht initialisiert, wird der String '*NO-INIT' ausgegeben.

VALUE = *NONE
Der Wert der Variablen wird nicht ausgegeben, sondern nur der Name (siehe Operand NAME).

NAME =
Bezeichnet in welcher Form der Name der Variablen ausgegeben wird.

NAME = *FULL-NAME(...)
Gibt den vollen Variablennamen aus.

LIST-INDEX-NUMBER = *NO / *YES
Es kann angegeben werden, ob bei Listenlementen statt (*LIST) die Elementnummer an den Namen angehängt wird

*LIST-INDEX-NUMBER = *NO*
Variablenname(*LIST) = <inhalt> für das erste Element
Variablenname(*LIST) = <inhalt> für das zweite Element, usw.

*LIST-INDEX-NUMBER = *YES*
Variablenname#1 = <inhalt> für das erste Element
Variablenname#2 = <inhalt> für das zweite Element, usw.

NAME = *ELEMENT-NAME(...)
Gibt die Elementnamen der Variablen aus. (Auch bei Variablen vom Datentyp STRUCTURE.)

LIST-INDEX-NUMBER = *NO / *YES
Es kann angegeben werden, ob bei Listenlementen die Ausgabe mit (*LIST) oder der Elementnummer beginnt.

*LIST-INDEX-NUMBER = *NO*
(*LIST) = <inhalt> für das erste Element
(*LIST) = <inhalt> für das zweite Element, usw.

*LIST-INDEX-NUMBER = *YES*
#1 = <inhalt> für das erste Element
#2 = <inhalt> für das zweite Element, usw.

NAME = *NONE
Der Name der Variablen wird nicht ausgegeben.

OUTPUT =
Bezeichnet das Ausgabemedium.

OUTPUT = *SYSOUT
Ausgabe nach SYSOUT.

OUTPUT = *SYSLST
Ausgabe nach SYSLST. (Jede Datenzeile beginnt mit einem „␣“, um eine korrekte Druckausgabe zu ermöglichen.)

OUTPUT = <filename 1..54 without-gen-vers>(...)
Ausgabe in die angegebene Datei, die eine SAM-Datei sein muss.

WRITE-MODE = *REPLACE
Der bisherige Inhalt der Datei soll überschrieben werden.

WRITE-MODE = *EXTEND
Die Ausgabe soll an den bisherigen Inhalt angehängt werden.

OUTPUT = *VARIABLE(...)
Ausgabe in eine Listenvariable.

VARIABLE-NAME = <structured-name 1..20>
Name der Listenvariablen.

WRITE-MODE = *REPLACE
Der bisherige Inhalt der Listenvariablen soll überschrieben werden.

WRITE-MODE = *EXTEND
Die Listenvariable soll erweitert werden, das heißt, die Ausgabe soll an den bisherigen Inhalt angehängt werden.

OUTPUT = *LIBRARY-ELEMENT(...)
Ausgabe in das Element einer PLAM-Bibliothek.

LIBRARY = <filename 1..54 without-vers>
Name der PLAM-Bibliothek.

ELEMENT = <composed-name 1..64>(...)
Name des Elements.

VERSION =
Bezeichnet die Versionsnummer des Elements.

VERSION = *HIGHEST-EXISTING
Wählt die höchste existierende Versionsnummer.

VERSION = *UPPER-LIMIT
Wählt die höchste mögliche Versionsnummer.

VERSION = <composed-name 1..24>
Wählt die angegebene Versionsnummer.

TYPE = S / alphanum-name 1..8
Bezeichnet den Elementtyp.

WRITE-MODE = *REPLACE
Der bisherige Inhalt des Elements soll überschrieben werden.

WRITE-MODE = *EXTEND
Das Element soll erweitert werden, d.h. die Ausgabe soll an den bisherigen Inhalt angehängt werden.

Kommando-Returncode

Es ist möglich, dass ein Teil des Kommandos schon abgearbeitet und ausgeführt wurde, bevor der Fehler auftrat. In diesem Fall ist das Ergebnis des Kommandos nicht garantiert.

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Warnung; keine Variable gefunden
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler garantierte Meldung: SDP1008
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

In Prozedur „Proz1“ wird folgende Variable deklariert:

```
/DECLARE-VARIABLE VARIABLE-NAME=WERT(TYPE=*INTEGER,INITIAL-VALUE=122),-
/SCOPE=*PROCEDURE(IMPORT-ALLOWED=*YES)
/CALL-PROCEDURE Proz2
```

In Prozedur „Proz2“ ist Folgendes geschrieben:

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL,SELECT=*BY-ATTRIBUTES-
/(SCOPE=*CALLING-PROCEDURES)
A=122
```

Beschreibung der Ausgabeformate

Abhängig von den Angaben bei INFORMATION werden unterschiedliche Ausgabeformate erzeugt. Die Ausgabe entspricht nicht einer Folge von SET-VARIABLE-Kommandos.

Beispiel

```
/DECLARE-VARIABLE NAME(INIT-VALUE = 'MUELLER')
/DECLARE-VARIABLE ALTER(TYPE = *INTEGER, INIT-VALUE = 22)
/DECLARE-VARIABLE SPRACHEN,MULTIPLE-ELEMENTS = *LIST
/SPRACHEN = 'DEUTSCH', WRITE-MODE = *EXTEND
/SPRACHEN = 'ENGLISCH', WRITE-MODE = *EXTEND
/DECLARE-VARIABLE NOTEN(TYPE = *STRUCTURE(*BY-SYSCMD))
/BEGIN-STRUCTURE
/DECLARE-ELEMENT DEUTSCH(TYPE = *INTEGER, INIT-VALUE = 2)
/DECLARE-ELEMENT ENGLISCH(TYPE =*INTEGER)
/END-STRUCTURE
/DECLARE-VARIABLE DOLMETSCHER(TYPE = *BOOLEAN,INIT-VALUE = TRUE)
```

*1. Ausgabe mit SELECT = *BY-ATTRIBUTES(INITIALIZATION = *YES)*

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL, SELECT=*BY-ATTRIBUTES -
(INITIALIZATION = *YES)
```

Ausgabe

```
ALTER = 22
DOLMETSCHER= TRUE
NAME = MUELLER
NOTEN.DEUTSCH = 2
SPRACHEN(*LIST)=DEUTSCH
SPRACHEN(*LIST)=ENGLISCH
*END-OF-CMD
```

Nur initialisierte Variablen werden ausgegeben.

*2. Ausgabe mit SELECT = *BY-ATTRIBUTES(INITIALIZATION = *ANY)*

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL, SELECT=*BY-ATTRIBUTES -
/(INITIALIZATION = *ANY)
```

Ausgabe

```
ALTER = 22
DOLMETSCHER= TRUE
NAME = MUELLER
NOTEN.DEUTSCH = 2
NOTEN.ENGLISCH = *NO-INIT
SPRACHEN(*LIST)=DEUTSCH
SPRACHEN(*LIST)=ENGLISCH
*END-OF-CMD
```

Alle Variablen, ob initialisiert oder nicht, werden ausgegeben.

*3. Ausgabe mit INFORMATION = *PARAMETERS(VALUE = *WITHOUT-QUOTES)*

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL, INFORMATION = *PARAMETERS -
/(VALUE = *WITHOUT-QUOTES)
```

Ausgabe

```
ALTER = 22
DOLMETSCHER= TRUE
NAME = MUELLER
NOTEN.DEUTSCH = 2
SPRACHEN(*LIST)=DEUTSCH
SPRACHEN(*LIST)=ENGLISCH
*END-OF-CMD
```

String-Variable werden ohne Anführungszeichen ausgegeben.

*4. Ausgabe mit INFORMATION = *PARAMETERS(VALUE = *C-LITERAL)*

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL, INFORMATION = *PARAMETERS -
/(VALUE = *C-LITERAL)
```

Ausgabe

```
ALTER = 22
DOLMETSCHER = TRUE
NAME = 'MUELLER'
NOTEN.DEUTSCH = 2
SPRACHEN(*LIST) = 'DEUTSCH'
SPRACHEN(*LIST) = 'ENGLISCH'
*END-OF-CMD
```

String-Variable werden als C-Literale ausgegeben.

*5. Ausgabe mit INFORMATION = *PARAMETERS(VALUE = *X-LITERAL)*

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL, INFORMATION = *PARAMETERS -
/(VALUE = *X-LITERAL)
```

Ausgabe

```
ALTER = 22
DOLMETSCHER = TRUE
NAME = X'D4E4C5D3D3C5D9'
NOTEN.DEUTSCH = 2
SPRACHEN(*LIST) = X'C4C5E4E3E2C3C8'
SPRACHEN(*LIST) = X'C5D5C7D3C9E2C3C8'
*END-OF-CMD
```

String-Variable werden als X-Literale ausgegeben.

*6. Ausgabe mit INFORMATION = *PARAMETERS(VALUE = *NONE)*

```
/SHOW-VARIABLE VARIABLE-NAME=*ALL, INFORMATION = *PARAMETERS -
/(VALUE = *NONE)
```

Ausgabe

```
ALTER
DOLMETSCHER
NAME
NOTEN.DEUTSCH
SPRACHEN(*LIST)
SPRACHEN(*LIST)
*END-OF-CMD
```

Es werden nur Variablennamen ausgegeben.

*7. Ausgabe mit INFORMATION = *PARAMETERS(NAME=*FULL-NAME)*

```
/SHOW-VARIABLE VARIABLE-NAME=NOTEN, INFORMATION = *PARAMETERS -
/(NAME=*FULL-NAME)
```

Ausgabe

```
NOTEN.DEUTSCH = 2
```

Der vollständige Elementname wird ausgegeben.

*8. Ausgabe mit INFORMATION = *PARAMETERS(NAME=*ELEMENT-NAME)*

```
/SHOW-VARIABLE VARIABLE-NAME=NOTEN, INFORMATION = *PARAMETERS -
/(NAME=*ELEMENT-NAME)
```

Ausgabe

```
DEUTSCH = 2
```

Nur der Element-Teilname wird ausgegeben.

*9. Ausgabe mit INFORMATION = *PARAMETERS(NAME=*NONE)*

```
/SHOW-VARIABLE VARIABLE-NAME=NOTEN, INFORMATION = *PARAMETERS -
/(NAME=*NONE)
```

Ausgabe

```
2
```

Nur Variablenwerte werden ausgegeben

Weitere Beispiele

Eingabe

```
/DECLARE-VARIABLE STRASSE('XYZ WEG', *STRING)
/DECLARE-VARIABLE NUMMER(12, *INTEGER)
/DECLARE-VARIABLE NAME('HUGO')
/DECLARE-VARIABLE VERHEIRATET(TRUE, *BOOLEAN)
/SHOW-VARIABLE *ALL
```

Ausgabe

```
NAME = HUGO
NUMMER = 12
STRASSE = XYZ WEG
VERHEIRATET = TRUE
*END-OF-CMD
```

Eingabe

```
/SHOW-VARIABLE (NAME, STRASSE, NUMMER), -
/ INFO=*PAR(VALUE=*C-LITERAL,NAME= *FULL-NAME)
```

Ausgabe

```
NAME = 'HUGO'
STRASSE = 'XYZ WEG'
NUMMER = 12
```

Eingabe

```
/DECLARE-VARIABLE A(TYPE = *STRUCTURE(*DYNAMIC))

/A.C = 'ZWEI'
/A.D = 'DREI'
/A.B = 'EINS'
/SHOW-VARIABLE A, INFO=*PAR(VALUE=*C-LITERAL,NAME= *FULL-NAME)
```

Ausgabe

```
A.C = 'ZWEI'
A.D = 'DREI'
A.B = 'EINS'
```

Eingabe

```
/SHOW-VARIABLE A,INFO=*PAR(VALUE=*C-LITERAL,NAME=*ELEMENT-NAME)
```

Ausgabe

```
C = 'ZWEI'
D = 'DREI'
B = 'EINS'
```

Eingabe

```
/DECLARE-VARIABLE V2(TYPE=*ANY),MULT-ELEM=*LIST
/S = '(ANTON,BERTA,CAESAR,HUGO,FRANZ)'
/V2 = *STRING-TO-VARIABLE(S)
```

Ausgabe

```
/SHOW-VARIABLE *LIST(LIST=V2,FROM-INDEX=3,NUM-OF-ELEM=*REST),-
/ INFO=*PAR(VALUE=*C-LITERAL,LIST-INDEX-NUMBER=*YES)
V2#3 = 'CAESAR'
V2#4 = 'HUGO'
V2#5 = 'FRANZ'
```

SHOW-VARIABLE-ATTRIBUTES
Variablenattribute ausgeben

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando SHOW-VARIABLE-ATTRIBUTES informiert über die Attribute der angegebenen Variablen. Die Ausgabe erfolgt nach SYSOUT (oder S-Variable / S-Variablenstrom) oder nach SYSLST.
Die Attribute umfassen Name der Variablen, Anfangswert, Datentyp, einfache oder zusammengesetzte Variable usw. Die Attribute werden mit dem Kommando /DECLARE-VARIABLE oder per Voreinstellung bei der Erzeugung der Variablen festgelegt.

Format

SHOW-VARIABLE-**ATTR**IBUTES

VARIABLE-**NAME** = ***ALL** / <composed-name 1..255> /<structured-name 1..20 with-wild(40)> / ***LIST**(...)

***LIST(...)**
- **LIST-NAME** = <composed-name 1..255>
- ,**FROM-INDEX** = ***FIRST** / ***LAST** / <integer 1..2147483647>
- ,**NUMBER-OF-ELEM**ENTS = **1** / ***REST** / <integer 1..2147483647>

,**INF**ORMATION = ***NAME** / ***VAR**IABLE-**ATTR**IBUTES-ONLY / ***ALL-ATTR**IBUTES

,**ATT**ACHED-**INF**ORMATION = ***NO** / ***YES**

,**OUTPUT** = ***SYSOUT** / ***SYSLST**

Operandenbeschreibung

VARIABLE-NAME =
Name der Variablen, deren Attribute ausgegeben werden sollen.

VARIABLE-NAME = *ALL
Alle prozedurlokalen, sichtbaren Variablen.

VARIABLE-NAME = <composed-name 1...255>
Name der Variablen, deren Attribute ausgegeben werden sollen.

VARIABLE-NAME = <structured-name 1...20 with-wildcards(40)>
Bezeichnung einer oder mehrerer Variablen unter Benutzung von Musterzeichen im Namen.

VARIABLE-NAME = *LIST(...)
Es sollen die Attribute der Elemente einer Listenvariablen ausgegeben werden.

LIST-NAME = <composed-name 1..255>
Name der Listenvariable.

FROM-INDEX = *FIRST / *LAST / <integer 1..2147483647>
Index des Elementes der Listenvariablen, mit dem die Ausgabe beginnen soll.
*FIRST: Die Ausgabe beginnt mit dem ersten Element der Liste; Voreinstellung.
Die Angabe *LAST gibt genau die Attribute des letztes Elements der Liste aus. Der Operand NUMBER-OF-ELEMENTS wird in diesem Fall ignoriert.

NUMBER-OF-ELEMENTS = 1 / *REST / <integer 1..2147483647>
Anzahl der Listenelemente, deren Attribute ausgegeben werden sollen. Voreinstellung: Die Attribute eines Elements werden ausgegeben.
Die Angabe *REST gibt die Attribute aller Elemente vom angegebenen Startelement (Operand FROM-INDEX) bis zum letzten Element der Liste aus.

INFORMATION =
Bestimmt den Umfang der auszugebenden Information.

INFORMATION = *NAME
Nur der Name der Variablen wird ausgegeben.

INFORMATION = *VARIABLE-ATTRIBUTES-ONLY
Es werden alle Attribute ausgegeben, die mit dem Kommando DECLARE-VARIABLE spezifiziert wurden.

INFORMATION = *ALL-ATTRIBUTES
Es werden alle Attribute ausgegeben, die mit dem Kommando DECLARE-VARIABLE spezifiziert wurden. Zusätzlich werden die Namen der Elemente der Variablen aufgelistet.

ATTACHED-INFORMATION =
Bestimmt, ob die Attribute der Variablenelemente ausgegeben werden sollen.

ATTACHED-INFORMATION = *NO
Es werden nur die Attribute der Variablen ausgegeben.

ATTACHED-INFORMATION = *YES
Die Attribute der Variablen und der Variablenelemente werden ausgegeben.

OUTPUT =
Bestimmt, ob die Informationen nach SYSOUT oder nach SYSLST ausgegeben werden.

OUTPUT = *SYSOUT
Die Informationen werden nach SYSOUT ausgegeben. Mit den Kommandos ASSIGN-SYSOUT oder ASSIGN-STREAM kann die Ausgabe auch in eine S-Variable erfolgen oder einem S-Variablenstrom zugewiesen werden.

OUTPUT = *SYSLST
Die Informationen werden nur nach SYSLST geschrieben.

Kommando-Returncode

(SC2) SC1	Maincode	Bedeutung / garantierte Meldungen
0	CMD0001	Ohne Fehler
1	CMD0202	Syntaxfehler
3	CMD2203	Falsche Syntaxdatei
32	CMD0221	Systemfehler (interner Fehler)
64	SDP0091	Semantikfehler (Variable existiert nicht) garantierte Meldung: SDP1008
130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/declare-variable S(type = *structure),multiple-elements=*array
/S#81.A = 'First Value'
/S#81.B#5 = 'Second Value'
/S#81.B#27 = 'Third Value'
/S#97.A = 'First Value'
/S#97.B#8 = 'Second Value'
/S#97.B#13 = 'Third Value'


/show-variable-attributes s,info=*all-attributes,attached-information=*yes

VARIABLE-NAME = S
 TYPE = *STRUCTURE(DEFINITION = *DYNAMIC)
 MULTIPLE-ELEMENTS = *ARRAY(LOWER-BOUND = 0, UPPER-BOUND = 2147483647)
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *NONE
 VALUE =
 NUMBER-OF-ELEMENTS = 2
 ELEM#1 = S#81
 ELEM#2 = S#97
```

```
VARIABLE-NAME = S#81
 TYPE = *STRUCTURE(DEFINITION = *DYNAMIC)
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *NONE
 VALUE =
 NUMBER-OF-ELEMENTS = 2
 ELEM#1 = S#81.A
 ELEM#2 = S#81.B
VARIABLE-NAME = S#81.A
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *STRING
 VALUE = First Value
 NUMBER-OF-ELEMENTS = 0
VARIABLE-NAME = S#81.B
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *ARRAY(LOWER-BOUND = -2147483648, UPPER-BOUND =
                     2147483647)
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *NONE
 VALUE =
 NUMBER-OF-ELEMENTS = 2
 ELEM#1 = S#81.B#5
 ELEM#2 = S#81.B#27
VARIABLE-NAME = S#81.B#5
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *STRING
 VALUE = Second Value
 NUMBER-OF-ELEMENTS = 0
```

```
VARIABLE-NAME = S#81.B#27
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *STRING
 VALUE = Third Value
 NUMBER-OF-ELEMENTS = 0
VARIABLE-NAME = S#97
 TYPE = *STRUCTURE(DEFINITION = *DYNAMIC)
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *NONE
 VALUE =
 NUMBER-OF-ELEMENTS = 3
 ELEM#1 = S#97.A
 ELEM#2 = S#97.B
 ELEM#3 = S#97.B13
VARIABLE-NAME = S#97.A
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *STRING
 VALUE = First Value
 NUMBER-OF-ELEMENTS = 0
VARIABLE-NAME = S#97.B
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *ARRAY(LOWER-BOUND = -2147483648, UPPER-BOUND =
                     2147483647)
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *NONE
 VALUE =
 NUMBER-OF-ELEMENTS = 1
 ELEM#1 = S#97.B#8
```

```
VARIABLE-NAME = S#97.B#8
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *STRING
 VALUE = Second Value
 NUMBER-OF-ELEMENTS = 0
VARIABLE-NAME = S#97.B13
 TYPE = *ANY
 MULTIPLE-ELEMENTS = *NO
 SCOPE = *PROCEDURE(IMPORT-ALLOWED = *NO)
 CONTAINER = *STD
 CONSTANT = *NO
 VALUE-TYPE = *STRING
 VALUE = Third Value
 NUMBER-OF-ELEMENTS = 0
```

Hinweis

Bei einer Listenvariablen werden die Elementnamen nicht aufgelistet. Der Elementname ist immer der Variablenname, gefolgt von dem Zeichen „#“ und der (sequenziellen) Elementnummer.

Strukturierte Ausgabe

Für die strukturierte Ausgabe in Variablen werden beim Kommando SHOW-VARIABLE-ATTRIBUTES die folgenden Operanden unterstützt:

– VARIABLE-NAME (alle Werte)
– INFORMATION (alle Werte)
– ATTACHED-INFORMATION (alle Werte)

Weitere Informationen, wie z.B. die Bedingungen für die Belegungen der einzelnen Variablen, sind der nachfolgenden Tabelle zu entnehmen.

Ausgabestruktur

Ausgabe-Information	Name der S-Variablen[1)]	T[2)]	Inhalt
hat die Variable einen konstanten Wert (/DECLARE-CONSTANT)	var#.CONSTANT	S	*YES *NO
Typ des Variablencontainers	var#.CONTAIN	S	<composed-name 1..64> *STD *VAR *JV
Name des Variablencontainers	var#.CONTAIN-NAME	S	<structured-name 1..20> <filename 1..54>
Elementname	var#.ELEM(*LIST)	S	<composed-name 1..255>
Import der Variablen erlaubt	var#.IMP-ALLOW	S	*YES *NO
Maximale Anzahl (Limit) der Listelemente	var#.LIM	I	<integer 0..2147483647>
untere Grenze des Arrayindexes	var#.LOWER-BOUND	I	<integer -2147483648..2147483647>
Die Variable ist eine List- oder eine Arrayvariable oder eine einfache Variable (*NO)	var#.MULT-ELEM	S	*LIST *ARRAY *NO
Anzahl der Variablenelemente	var#.NUM-OF-ELEM	I	<integer 0..2147483647>
Geltungsbereich der Variablen	var#.SCOPE	S	*INC *PROC *TASK
Name des Strukturlayouts	var#.STRUCT-DEFI	S	<structured-name 1..20> *BY-SYSCMD *DYNAMIC
Variablentyp	var#.TYPE	S	*ANY *BOOLEAN *STRING *INTEGER *STRUCT
obere Grenze des Arrayindexes	var#.UPPER-BOUND	I	<integer -2147483648..2147483647>
Variablenwert	var#.VALUE	S I B	<string 0..4096> <integer -2147483648..2147483647> FALSE TRUE
Typ des Variablenwerts	var#.VALUE-TYPE	S	*NONE *BOOLEAN *STRING *INTEGER
Name der Variablen	var#.VAR-NAME	S	<composed-name 1..255>

[1)]Die Variablennamen sind alphabetisch geordnet

[2)]In der Spalte T sind die Datentypen aufgeführt: B = Boolean, I = INTEGER, S = String

SHOW-VARIABLE-CONTAINER-ATTR
Offene Variablenbehälter anzeigen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando SHOW-VARIABLE-CONTAINER-ATTR zeigt alle offenen Variablenbehälter an.

Format

SHOW-VARIABLE**-CONTAINER-ATTR**
CONTAINER-NAME = ***ALL** / <composed-name 1..64 with-wild(80)> / list-poss: <composed-name 1..64> ,**CONTAINER-SCOPE** = ***VISIBLE** / ***PROC**EDURE / ***CURR**ENT / ***TASK** ,**OUTPUT** = ***SYSOUT** / ***SYSLST**

Operandenbeschreibung

CONTAINER-NAME =
Name des offenen Variablenbehälters, der angezeigt werden soll.

CONTAINER-NAME = *ALL
Es werden alle offenen Variablenbehälter angezeigt.

CONTAINER-NAME = <composed-name 1..64 with-wild(80)>
Es werden alle offenen Variablenbehälter angezeigt, die das angegebene Suchmuster erfüllen.

CONTAINER-NAME = list-poss: <composed-name 1..64>
Liste der Variablenbehälter, die angezeigt werden sollen.

CONTAINER-SCOPE =
Geltungsbereich, in dem sich die Variablenbehälter befinden, die angezeigt werden sollen.

CONTAINER-SCOPE = *VISIBLE
Es werden die Variablenbehälter angezeigt, auf die von der aktuellen Prozedurebene zugegriffen werden kann. Aktuelle Namen von Variablenbehälter verdecken Variablenbehälter von höheren Prozedurebenen oder der Taskebene.

CONTAINER-SCOPE = *PROCEDURE
Es werden die Variablenbehälter angezeigt, auf die von der aktuellen Prozedurebene zugegriffen werden kann, und die mit Geltungbereich PROCEDURE eröffnet wurden.

CONTAINER-SCOPE = *CURRENT
Es werden die Variablenbehälter angezeigt, auf die von der aktuellen Prozedurebene zugegriffen werden kann und die mit Geltungsbereich CURRENT eröffnet wurden.

CONTAINER-SCOPE = *TASK
Es werden die Variablenbehälter mit Geltungsbereich TASK angezeigt.

OUTPUT =
Adresse für die Output-Informationen.

OUTPUT = *SYSOUT
Die Output-Informationen werden auf SYSOUT ausgegeben und/oder in die Variablen, abhängig von der ASSIGN-SYSOUT- und ASSIGN-STREAM-Zuweisung.

OUTPUT = *SYSLST
Die Output-Informationen werden auf SYSLST ausgegeben ohne Rücksicht darauf wie der SYSINF-Variablenstrom zugewiesen wurde.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Eingabe

```
/OPEN-VARIABLE-CONTAINER MY-CONT1, *LIBRARY-ELEMENT(#MY-CONTAINER-LIB)
/SHOW-VARIABLE-CONTAINER-ATTR MY-CONT1
```

Ausgabe

```
CONTAINER-NAME = MY-CONT1
  FROM-FILE = *LIBRARY-ELEMENT
    LIBRARY = :1OSN:$QM123.S.152.OWDK.MY-CONTAINER-LIB
    ELEMENT = MY-CONT1
    VERSION = *HIGHEST-EXISTING
  LOCK      = *NO
  SCOPE     = *PROCEDURE
```

Eingabe

```
/SAVE-VARIABLE-CONTAINER MY-CONT1
/CLOSE-VARIABLE-CONTAINER MY-CONT1
```

Strukturierte Ausgabe

Für die strukturierte Ausgabe in Variablen werden beim Kommando SHOW-VARIABLE-CONTAINER-ATTR die folgenden Operanden unterstützt:

– CONTAINER-NAME (alle Werte)
– CONTAINER-SCOPE (alle Werte)

Weitere Informationen, wie z.B. die Bedingungen für die Belegungen der einzelnen Variablen, sind der nachfolgenden Tabelle zu entnehmen.

Ausgabestruktur

Ausgabe-Information	**Name der S-Variablen[1)]**	**T[2)]**	**Inhalt**
Name des Variablen-Containers	var#.CONTAIN-NAME	S	<comp.-name 1..64>
Name des Bibliothekelements, das den Variablen-Container enthält	var#.FROM-F	S	*LIB-ELEM(...)
Lese-/Schreibzugriff auf Bibliothekselement	var#.LOCK	S	*NO *YES
Sicherung des Variablen-Containers bei EXIT-JOB/LOGOFF (nur bei SCOPE=*TASK)	var#.SAVE-AT-TERM	S	*NO *YES
Geltungsbereich des Variablen-Behälters	var#.SCOPE	S	*INC *PROC *TASK

[1)]Die Variablennamen sind alphabetisch geordnet

[2)]In der Spalte T sind die Datentypen aufgeführt: S entspricht String.

SORT-VARIABLE
Listenvariable sortieren

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando SORT-VARIABLE sortiert die Elemente einer Listenvariablen nach ihrem Inhalt in aufsteigender oder absteigender Reihenfolge. Es können nur Listenvariablen sortiert werden, deren Elemente einfache Variablen gleichen Typs (STRING, INTEGER, BOOLEAN oder ANY) sind.

Format

SORT-VARIABLE
VARIABLE**-NAME** = list-poss(2000): <composed-name 1..255> ,**SORT**ING**-ORDER** = ***ASCEND**ING / ***DESCEND**ING

Operandenbeschreibung

VARIABLE-NAME = list-poss(2000): <composed-name 1..255>
Bezeichnet die zu sortierende Variable. Bei Angabe in Listenform können mehrere Variablennamen angegeben werden.

SORTING-ORDER = *ASCENDING / *DESCENDING
Bestimmt die Sortierreihenfolge.

SORTING-ORDER = *ASCENDING
Sortiert die Elemente in die Reihenfolge niedrigster bis zum höchster Wert. Abhängig vom Typ der zu vergleichenden Elemente wird der Größenvergleich wie folgt durchgeführt:

– Elemente vom Typ STRING
  Bei einer Liste mit Elementen vom Typ STRING werden die Elemente aufsteigend nach Größe der String-Werte sortiert. Verglichen wird pro Byte der Wert der Hexadezimal-Codierung (siehe auch „String-Vergleich“ auf Seite 267). Durch diese Art des Vergleiches können auch X-Literale sortiert werden.

  Es gilt:
  – Von zwei Strings, die mit den selben Werten beginnen, aber unterschiedliche Länge haben, besitzt der kürzere String den kleineren Wert.
  – Ein leerer String besitzt immer den kleinstmöglichen Wert.

Beispiel: Bei den Strings BC, ABC und BCD ergibt sich folgende Größenrelation:

ABC < BC und ABC < BCD und BC < BCD

Die aufsteigende Sortierung ergibt die Reihenfolge ABC, BC und BCD.

– Elemente vom Typ INTEGER
  Bei einer Liste mit Elementen vom Typ INTEGER werden die Elemente aufsteigend nach Größe der Integer-Werte sortiert (numerischer Vergleich).

– Elemente vom Typ BOOLEAN
  Bei einer Liste mit Elementen vom Typ BOOLEAN werden die Elemente aufsteigend nach Größe der Boolean-Werte sortiert. Es gilt: der Wert „FALSE“ kleiner als „TRUE“.

– Elemente vom Typ ANY
  Bei einer Liste mit Elementen vom Typ ANY werden die Element aufsteigend gemäß dem Typ des aktuellen Wertes sortiert (aufsteigende Reihenfolge von String-, Integer- oder Boolean-Werte, wie oben beschrieben).
  Falls die Elemente der Liste Werte unterschiedlichen Typs besitzen, kann die Liste nicht sortiert werden.

SORTING-ORDER = *DESCENDING
Sortiert die Elemente in die Reihenfolge höchster bis zum niedrigster Wert. Die Art des Größenvergleichs ist abhängig vom Typ der zu vergleichenden Elemente (siehe SORTING-ORDER=*ASCENDING).

Kommando-Returncode

Während der Sortierung von Listen ist es möglich, dass ein Teil des Kommandos abgearbeitet und ausgeführt wurde, bevor ein Fehler auftrat. In diesem Fall ist das Resultat des Kommandos nicht garantiert.

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	SDP2000	Warnung; einige Elemente konnten nicht sortiert werden
	64	SDP2001	Keines der Elemente konnte sortiert werden

Beispiel

```
/DECLARE-VARIABLE LANGUAGE-LIST (TYPE = *STRING), -
/           MULTIPLE-ELEMENTS = *LIST
/ LANGUAGE-LIST = 'GERMAN', WRITE-MODE=*EXTEND
/ LANGUAGE-LIST = 'ENGLISH', WRITE-MODE=*EXTEND
/ LANGUAGE-LIST = 'FRENCH', WRITE-MODE=*EXTEND
/ LANGUAGE-LIST = 'ITALIAN', WRITE-MODE=*EXTEND
/ LANGUAGE-LIST = 'GREEK', WRITE-MODE=*EXTEND
/
/SORT-VARIABLE VARIABLE-NAME=LANGUAGE-LIST, SORTING-ORDER=*ASCENDING
/SHOW-VARIABLE VARIABLE-NAME=LANGUAGE-LIST
LANGUAGE-LIST(*LIST) = ENGLISH
LANGUAGE-LIST(*LIST) = FRENCH
LANGUAGE-LIST(*LIST) = GERMAN
LANGUAGE-LIST(*LIST) = GREEK
LANGUAGE-LIST(*LIST) = ITALIAN
```

TRACE-PROCEDURE
Unterbrochene Prozedur schrittweise fortsetzen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Es kann festgelegt werden, nach wie vielen Kommandos die Prozedur erneut unterbrochen wird. Falls der Unterbrechungspunkt in einem Bereich liegt, in dem Prozedurunterbrechungen verboten sind, so wird die Unterbrechung erst wirksam, wenn wieder Unterbrechungen erlaubt sind.

Falls in der Prozedur Protokollierung erlaubt ist, so wird sie eingeschaltet für die Kommandos, die nach dem TRACE-PROCEDURE-Kommando durchlaufen werden (unabhängig vom LOGGING-Operanden im Kommando CALL-PROCEDURE oder MODIFY-PROCEDURE-TEST-OPTIONS).

Format

TRACE-PROCEDURE	Kurzname: **TCP**
STEPS = ***LAST-INP**UT / <text 0..1800 with-low *arith-expr*>	

Operandenbeschreibung

STEPS =
Bestimmt die Anzahl der Kommandos, die bis zur nächsten Unterbrechung ausgeführt werden sollen.

STEPS = *LAST-INPUT
Vereinbart die zuletzt definierte Anzahl. Fehlt eine vorherige Vereinbarung, so gilt der Wert 1, das heißt: Unterbrechung nach jedem Kommando.

STEPS = <text 0..1800 with-low *arith-expr*>
Integer-Ausdruck; vereinbart die Anzahl Kommandos bis zur nächsten Unterbrechung.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

TRANSMIT-BY-STREAM
Variablen übertragen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

Das Kommando TRANSMIT-BY-STREAM führt von Client-Seite eine Variablen-Übertragung über den angegebenen S-Variablenstrom zu oder vom angesprochenen Server durch.

TRANSMIT-BY-STREAM sendet zuerst die in VARIABLE-NAME angegebene Variable zur Steuereinheit SDF-P und empfängt dann von SDF-P Daten in der in RETURN-VARIABLE-NAME angegebenen Variablen.

Wenn *VARIABLE dem S-Variablenstrom im Kommando ASSIGN-STREAM zugewiesen ist, werden die Variablen, die beim Kommando TRANSMIT-BY-STREAM in RETURN-VARIABLE-NAME und in RET-CONTROL-VAR-NAME enthalten sind, durch die gleichnamigen Variablen vom Kommando ASSIGN-STREAM überschrieben.

Wenn *DUMMY dem S-Variablenstrom zugewiesen ist, gibt das Kommando einen Returncode zurück, der besagt, dass nichts geändert wurde.

Die Reihenfolge der Abarbeitung ist im Kapitel „S-Variablenströme“ auf Seite 193 beschrieben.

Format

TRANSMIT-BY-STREAM
STREAM-NAME = <structured-name 1..20>
,**VAR**IABLE-**NAME** = ***NONE** / <composed-name 1..255>
,**RETURN-VAR**IABLE-**NAME** = ***SAME** / ***NONE** / <composed-name 1..255>
,**CONTROL-VAR-NAME** = ***NONE** / <composed-name 1..255>
,**RET-CONTROL-VAR-NAME** = ***SAME** / ***NONE** / <composed-name 1..255>

Operandenbeschreibung

STREAM-NAME = <structured-name 1..20>
Name des S-Variablenstroms, in dem die Variable übertragen wird.

VARIABLE-NAME =
Bei der Übertragung gesendete S-Variable (Output-Variable).

VARIABLE-NAME = *NONE
Es wird keine S-Variable übertragen.
Wenn der Variablenstrom einer Variable zugewiesen wird, bleibt die zugewiesene S-Variable unverändert.

VARIABLE-NAME = <composed-name 1..255>
Name der S-Variable, die zum Server gesendet wird.
Die angegebene S-Variable muss eine Struktur sein, die alle Daten enthält, die vom Client gesendet werden müssen. Wenn eine leere S-Variable angegeben und der Variablenstrom zu *VARIABLE(...) zugewiesen wird, wird ein leeres Listenelement erzeugt.

RETURN-VARIABLE-NAME =
Name der Return-Variable bei der Übertragung.

RETURN-VARIABLE-NAME = *SAME
Die Return-Variable ist die gesendete S-Variable.
Zuerst wird die übertragene S-Variable zum Server geschrieben, danach wird sie mit den Return-Informationen vom Server überschrieben.
Z.B. wenn der S-Variablenstrom zu *VARIABLE(...) zugewiesen wird, wird die übertragene Variable zuerst zur S-Variable (Liste) geschrieben, die im Kommando ASSIGN-STREAM im Operanden VARIABLE-NAME angegeben ist, danach wird sie durch die S-Variable (Liste) überschrieben, die bei RETURN-VARIABLE-NAME angegeben ist.

RETURN-VARIABLE-NAME = *NONE
Es wird vom Client keine Return-Variable zurückerwartet.
Wenn der S-Variablenstrom zu *VARIABLE(...) zugewiesen wird, wird die angegebene Return-Variable (Listenelement) jedoch behandelt, als ob eine Return-Variable in TRANSMIT-BYSTREAM angegeben würde.

RETURN-VARIABLE-NAME = <composed-name 1..255>
Name der S-Variable, die die Daten enthält, die vom Server nach Ausführung von TRANSMIT-BY-STREAM zurückgesendet werden.
Bei RETURN-VARIABLE-NAME =VARIABLE-NAME wird dasselbe durchgeführt wie bei RETURN-VARIABLE-NAME = *SAME.
Wenn der Server keine RETURN-VARIABLE zurücksendet, ist die Variable nicht geändert worden.

CONTROL-VAR-NAME =
Gibt die Kontroll-Informationen für den Server an.
Diese Information wird von einem Standard Header identifiziert, dessen Format und Aufbau nach der Beschreibung der Kommando-Returncodes aufgelistet ist.
Die Art und Weise dieser Kontroll-Information wird von den möglichen Servern bestimmt. Die Kontroll-Information ist nicht Teil der Benutzerdaten. Es sind für diesen Prozess beim Server eigene Feldnamen reserviert: z.B. definiert das Subsystem FHS (TPR-Anzeige) Variablen, die u.a. die Panels bestimmen, die geladen werden müssen und in denen FHS seine Aktionen ausführt.

CONTROL-VAR-NAME = *NONE
Es ist keine Kontroll-Variable angegeben.
Wenn der Server Kontroll-Variablen benötigt, werden entweder (wenn möglich) die Standard-Variablen benutzt oder die Übertragung wird abgewiesen.

CONTROL-VAR-NAME = <composed-name 1..255>
Name der S-Variable, die die Kontroll-Informationen enthält.
Die angegebene S-Variable muss eine Struktur sein.

Hinweis

Wenn der Server eine Kontroll-Variable fordert, und es wurde vom Aufrufer keine oder nur eine unvollständige angegeben, gibt es zwei mögliche Ergebnisse der Übertragung:

1. Wenn die fehlende Kontroll-Information für den steuernden Server optional ist, wird die Übertragung mit der Voreinstellung vorgenommen.
2. Wenn die fehlende Kontroll-Information für den steuernden Server obligatorisch ist, wird die Übertragung mit einer Fehlermeldung abgebrochen.

RET-CONTROL-VAR-NAME =
Gibt die S-Variable an, von der Return-Kontroll-Informationen zum Client geschickt werden können.

RET-CONTROL-VAR-NAME = *SAME
Es wird dieselbe S-Variable wie bei CONTROL-VAR-NAME angegeben.
Es gelten dieselben Regeln wie bei gleichzeitiger Angabe von VARIABLE-NAME und RETURN-VARIABLE-NAME.

RET-CONTROL-VAR-NAME = *NONE
Es werden keine Return-Kontroll-Informationen vom Client erwartet.

RET-CONTROL-VAR-NAME = <composed-name 1..255>
Name der S-Variable, durch die Return-Kontroll-Informationen vom Server übertragen werden. Die angegebene S-Variable muss eine Struktur sein.
Wenn der Server keine RET-CONTROL-VARIABLE zurücksendet, ist die Variable nicht geändert worden.

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung / garantierte Meldungen
	0	CMD0001	Ohne Fehler
1	0	CMD0001	Variablenstrom ist *DUMMY zugewiesen; keine Übertragung, Variable bleibt unverändert
2	0	SDP0512	Server ist nicht länger aktiv. Datenstrom ist *DUMMY zugewiesen
2	0	SDP0531	Warnung vom Server; Prozess wird fortgesetzt
	1	CMD0202	Syntaxfehler
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	CMD0216	Erforderliches Privileg fehlt
	64	SDP0091	Semantikfehler garantierte Meldungen: SDP1008
	64	SDP0532	Server-Fehler; Kommando abgewiesen
	64	SDP0534	Interner Server-Fehler; Kommando abgebrochen. Server-Verbindung abgebrochen nach unerwartetem Ereignis oder bei mangelhaften oder fehlenden System-Ressourcen
	64	SDP0517	Variablenstrom existiert nicht
	64	SDP0522	Übertragene Daten inkompatibel zum Format, das der Server bearbeitet
	64	SDP1132	Variablenname zu lang
	130	SDP0099	Kein Adressraum mehr verfügbar

Hinweis

Beim Returncode SDP0512 muss berücksichtigt werden: Wenn S-Variablenströme, die von der Aufruf-Prozedur abhängig sind, zu prozedurlokalen Variablen zugewiesen werden, können sie nach Prozedurende nicht mehr benützt werden. Die darauf folgende Übertragung wird mit einer Warnung ignoriert und der S-Variablenstrom wird auf *DUMMY zurückgesetzt.

Beispiel

```
/DECLARE-VARIABLE OPS-VAR(TYPE=*STRUCTURE),MULTIPLE-ELEMENTS=*LIST
/DECLARE-VARIABLE OPS-VAR1(TYPE=*STRUCTURE)
/ASSIGN-STREAM SYSINF,TO=*VARIABLE(OPS-VAR)
/ASSIGN-SYSOUT TO=#ERROR-SYSOUT
/TRANSMIT-BY-STREAM SYSINF, VARIABLE=OPS-VAR1, RETURN-VARIABLE=*NONE
```

Standard Header

SDF-P liefert einen Standard Header als ein reserviertes Strukturlayout. Dieser Standard Header kann als erstes Element der Kontroll-Variablenstruktur definiert werden.

Sein Layout ist in einer Prozedur deklariert, die mit INCLUDE-PROCEDURE aufgerufen werden muss.

Diese Prozedur steht im Element FHDR in der (ausgelieferten) Bibliothek $TSOS.SYSPRC.SDF-P-BASYS.024, und sieht folgendermaßen aus:

```
/set-proc-options "caller=include    not supported by sdf-p-basys"
/begin-parameter-declaration
/   declare-parameter -
/   "------------ std param ---------------------------------*"-
                /(PREFIX       (init='SYSSDP') -
                /,INCLUDE-FORM (init='LAYOUT') "/INITIALIZE" -
                /,VARIABLE-NAME(init='') -
/   "------------ include specific param -------------------*"-
                /,UNIT         (init='') -
                /,FUNCTION     (init='') -
                /,VERSION      (init=0) -
                /,SUBCODE2     (init=0) -
                /,SUBCODE1     (init=0) -
                /,MAINCODE     (init='CMD0001') -
                /)
/end-parameter-declaration
/
/if (not is-sdf-p() )
/   exit-procedure error=*yes(subcode1=41,maincode=cmd2241)
/end-if
/
/if (upper-case(INCLUDE-FORM) == 'LAYOUT')
/
/begin-structure name=&PREFIX.IFID-MDL,scope=proc
/  declare-element -
                  /(UNIT      (type=string) -
                  /,FUNCTION  (type=string) -
                  /,VERSION   (type=integer) -
                  /)
/end-structure
/begin-structure name=&PREFIX.RETC-MDL,scope=proc
/  declare-element -
                  /(SUBCODE2 (type=integer) -
                  /,SUBCODE1 (type=integer) -
                  /,MAINCODE (type=string)  -
                  /)
/end-structure
/begin-structure name=&PREFIX.FHDR,scope=proc
```

```
/  declare-element INTERFACE-ID(type=structure(&PREFIX.IFID-MDL))
/  declare-element RETURNCODE  (type=structure(&PREFIX.RETC-MDL))
/end-structure
/
/else-if (upper-case(INCLUDE-FORM) == 'INITIALIZE')
/
/   if (VARIABLE-NAME == '')
/      write-text '%  mandatory parameter variable-name missing.'
/      raise-error
/   end-if
/   declare-variable PARAM(type=string)
/   for PARAM = ('UNIT','FUNCTION')
/      &VARIABLE-NAME..INTERFACE-ID.&PARAM = &PARAM
/   end-for
/   &VARIABLE-NAME..INTERFACE-ID.VERSION = INTEGER(VERSION)
/   for PARAM = ('SUBCODE2','SUBCODE1')
/      &VARIABLE-NAME..RETURNCODE.&PARAM = INTEGER(&PARAM)
/   end-for
/   &VARIABLE-NAME..RETURNCODE.MAINCODE = MAINCODE
/
/else
/  write-text '%  form=&INCLUDE-FORM not supported; include aborts'
/  raise-error
/end-if
/exit-procedure
```

Beschreibung der Prozedurparameter

UNIT (TYPE = *STRING)
Name des Servers, der die Kontroll-Variable definiert. Dieser Name sollte identisch mit dem bei ASSIGN-STREAM definierten Namen sein..

FUNCTION (TYPE = *STRING)
Name der Funktion, für die der Server das Kontroll-Variablen-Layout definiert hat. Dieser Name wird vom Server definiert und weitergereicht.

VERSION (TYPE = *INTEGER)
Version der Kontroll-Variablen. Frühere Versionen der Kontroll-Variablen können so kompatibel vom Server unterstützt werden.

SUBCODE2 (TYPE = *INTEGER)
Subcode2, der vom Server zurückgegeben wird (in RET-CONTROL-VAR-NAME).

SUBCODE1 (TYPE = *INTEGER)
Subcode1, der vom Server zurückgegeben wird (in RET-CONTROL-VAR-NAME).

MAINCODE (TYPE = *STRING)
Message-Id, die vom Server zurückgegeben wird (in RET-CONTROL-VAR-NAME).

Die Operanden SUBCODE2, SUBCODE1 und MAINCODE haben dieselbe Konvention wie der Kommando-Returncode.

Der Router (TRANSMIT-BY-STREAM und TRANSVV) benützt diese Variablen nicht. Warnungen oder Fehlermeldungen von TRANSMIT-BY-STREAM und TRANSVV hängen vom internen Returncode ab, der vom Server und nicht über diese Kontroll-Variablen zurückgegeben wird.

Diese Kontroll-Variablen sollen deswegen vom Server zurückgegeben werden, um genauere Fehlermeldungen als die von TRANSMIT-BY-STREAM und TRANSVV zu erhalten.

Hinweis

Die Fehler und Warnungen, die in diesen Kontroll-Variablen geschrieben werden, sollen mit dem internen Returncode des Servers übereinstimmen. Das bedeutet z.B., wenn TRANSMIT-BY-STREAM die Fehlerklasse SUBCODE2=0,SUBCODE1=64 und MAINCODE=SDP0532 zurückgibt, auch in der Return-Kontroll-Variablen ein Fehlercode enthalten ist; oder wenn SUBCODE2=2,SUBCODE1=0 und MAINCODE= SDP0531 zurückgegeben wird, soll in der Return-Kontroll-Variablen (auch) eine Warnung enthalten sein usw. Diese Konsistenz untersteht allerdings der Verantwortung des Servers.

UNTIL
REPEAT-Block abschließen

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

UNTIL enthält die Schleifenbedingung für einen REPEAT-Block, d. h. für eine REPEAT-Schleife, und schließt den REPEAT-Block ab (Einleitungskommando für den REPEAT-Block: REPEAT). Ist die Bedingung nicht erfüllt, wird ein erneuter Schleifendurchlauf mit dem ersten Kommando des REPEAT-Blocks gestartet. Andernfalls wird die Schleife beendet und der Prozedurlauf mit dem ersten Kommando, das auf das Kommando UNTIL folgt, fortgesetzt. (Siehe auch Abschnitt „REPEAT-Block“ auf Seite 100.)

Ausdrucksersetzung in der Bedingung erfolgt bei jedem Schleifendurchlauf.

Format

UNTIL
CONDITION = <text 0..1800 with-low *bool-expr*>

Operandenbeschreibung

CONDITION = <text 0..1800 with-low *bool-expr*>

Logischer Ausdruck als Bedingung für den Abbruch des REPEAT-Blocks (logischer Ausdruck siehe Kapitel „Ausdrücke“ auf Seite 255).

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0139	Back Branch-Grenze erreicht
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

Siehe Seite 100.

WHILE
WHILE-Block einleiten

Anwendungsgebiet: **PROCEDURE**

Kommandobeschreibung

WHILE leitet einen WHILE-Block, das heißt eine WHILE-Schleife, ein: Die Ausführung der Kommandofolge innerhalb des WHILE-Blocks wird wiederholt, solange die im WHILE-Kommando angegebene Bedingung erfüllt ist. Ist die Bedingung nicht erfüllt, wird die Schleife beendet und der Prozedurlauf mit dem Kommando, das auf das Abschlusskommando END-WHILE folgt, fortgesetzt. (Siehe auch Abschnitt „WHILE-Block“ auf Seite 99).

Eine Ausdrucksersetzung im Operanden erfolgt nur beim Eintritt in die WHILE-Schleife, nicht bei jedem Schleifendurchlauf.

Format

WHILE
CONDITION = <text 0..1800 with-low *bool-expr*>

Operandenbeschreibung

CONDITION = <text 0..1800 with-low *bool-expr*>
Logischer Ausdruck als Bedingung für das erneute Durchlaufen der Kommandos innerhalb der WHILE-Schleife (logischer Ausdruck siehe Kapitel „Ausdrücke“ auf Seite 255).

Kommando-Returncode

(SC2)	SC1	Maincode	Bedeutung
	0	CMD0001	Ohne Fehler
	1	CMD0202	Syntaxfehler
	1	SDP0118	Kommando im falschen Kontext
	1	SDP0223	Falsche Umgebung
	3	CMD2203	Falsche Syntaxdatei
	32	CMD0221	Systemfehler (interner Fehler)
	64	SDP0091	Semantikfehler
	130	SDP0099	Kein Adressraum mehr verfügbar

Beispiel

```
/ "Reduzieren der Listenvariablen LIST-A auf die letzten 250 Elemente"
/WHILE (SIZE ('LIST-A') > 250)
/  FREE-VARIABLE LISTE-A#
/END-WHILE
```

16 Installation und Konfiguration

Auf den folgenden Seiten werden die Installation von SDF-P und die erforderliche Software-Konfiguration beschrieben.

16.1 Installation von SDF-P

SDF-P wird als eigene Liefereinheit geliefert, SDF-P-BASYS ist Bestandteil des Grundausbaus. Die Installation erfolgt mit dem Installationmonitor IMON (siehe Handbuch „IMON“ [12]).

Die Subsysteme SDFPBASY (für SDF-P-BASYS) und SDF-P werden automatisch beim Startup geladen.

16.1.1 Installationsdateien für SDF-P

Für die Installation werden folgende Dateien benötigt:

Dateiname	Inhalt
SPMLNK.SDF-P.024	Unabhängier Teil von SDF-P V2.4A, der automatisch zur Startup-Zeit geladen wird (Anlagen mit SPARC-Architektur)
SRMLNK.SDF-P.024	Unabhängier Teil von SDF-P V2.4A, der automatisch zur Startup-Zeit geladen wird (Anlagen mit RISC-Architektur)
SYSFGM.SDF-P.024.D	Freigabemitteilungen (Deutsch)
SYSFGM.SDF-P.024.E	Freigabemitteilungen (Englisch)
SYSLNK.SDF-P.024	Unabhängier Teil von SDF-P V2.4A, der automatisch zur Startup-Zeit geladen wird (Anlagen mit /390-Architektur)
SYSPRC.SDF-P.024	Prozeduren für SDF-P V2.4A (z.B. FHDR)
SYSRMS.SDF-P.024	RMS-Lieferungssätze für SDF-P V2.4A
SYSSDF.SDF-P.024	Syntaxdatei, die das Kommandos COMPILE-PROCEDURE enthält, das nur mit SDF-P selbst ausführbar ist
SYSSII.SDF-P.024	Struktur- und Installationsinformationsdatei von IMON über Release-Units und Release-Items von SDF-P V2.4A
SYSSSC.SDF-P.024.	SSCM-Katalog, der das Subsystem SDF-P definiert

IMON-Installationsinformation für SDF-P

Logischer IMON-Name	Default-Pfadname
SYSFGM.D	$TSOS.SYSFGM.SDF-P.024.D
SYSFGM.E	$TSOS.SYSFGM.SDF-P.024.E
SYSLNK	$TSOS.SYSLNK.SDF-P.024
SYSSSC	$TSOS.SYSSSC.SDF-P.024
SYSSDF	$TSOS.SYSSDF.SDF-P.024
SYSREP	$TSOS.SYSREP.SDF-P.024
SYSRMS	$TSOS.SYSRMS.SDF-P.024
SYSPRC	$TSOS.SYSPRC.SDF-P.024
SYSSII	$TSOS.SYSSII.SDF-P.024

16.1.2 Installationsdateien für SDF-P-BASYS

Für die Installation werden folgende Dateien benötigt:

Dateiname	**Inhalt**
SIPLIB.SDF-P-BASYS.024	Bibliothek mit privilegierten Makros
SPMLNK.SDF-P-BASYS.024	Unabhängiger Teil von SDF-P-BASYS V2.4A, der automatisch zur Startup-Zeit geladen wird (Anlagen mit SPARC-Architektur)
SRMLNK.SDF-P-BASYS.024	Unabhängiger Teil von SDF-P-BASYS V2.4A, der automatisch zur Startup-Zeit geladen wird (Anlagen mit RISC-Architektur)
SYSLIB.SDF-P-BASYS.024	SDF-P-BASYS-Assembler-Makros
SYSLNK.SDF-P-BASYS.024	Unabhängiger Teil von SDF-P-BASYS V2.4A, der automatisch zur Startup-Zeit geladen wird (Anlagen mit /390-Architektur)
SYSMES.SDF-P-BASYS.024	Meldungsdatei mit Meldungen von SDF-P-BASYS V2.4A
SYSPRC.SDF-P-BASYS.024	Prozeduren für SDF-P-BASYS V2.4A (z.B. FHDR)
SYSRMS.SDF-P-BASYS.024	RMS-Lieferungssätze für SDF-P-BASYS V2.4A
SYSSDF.SDF-P-BASYS.024	Syntaxdatei, die alle Kommandos enthält, die SDF-P-BASYS selbst durchführt (entweder direkt oder innerhalb einer compilierten Prozedur)
SYSSII.SDF-P-BASYS.024	Struktur- und Installationsinformationsdatei von IMON für SDF-P-BASYS V2.4A
SYSSSC.SDF-P-BASYS.024	SSCM-Katalog, der das Subsystem SDFPBASY definiert

IMON-Installationsinformation für SDF-P-BASYS

Logischer IMON-Name	Default-Pfadname
SIPLIB	$TSOS.SIPLIB.SDF-P-BASYS.024
SYSLIB	$TSOS.SYSLIB.SDF-P-BASYS.024
SYSLNK	$TSOS.SYSLNK.SDF-P-BASYS.024
SYSMES	$TSOS.SYSMES.SDF-P-BASYS.024
SYSPRC	$TSOS.SYSPRC.SDF-P-BASYS.024
SYSSSC	$TSOS.SYSSSC.SDF-P-BASYS.024
SYSSDF	$TSOS.SYSSDF.SDF-P-BASYS.024
SYSREP	$TSOS.SYSREP.SDF-P-BASYS.024
SYSRMS	$TSOS.SYSRMS.SDF-P-BASYS.024
SYSPRC	$TSOS.SYSPRC.SDF-P-BASYS.024
SYSSII	$TSOS.SYSSII.SDF-P-BASYS.024

16.2 SW-Konfiguration

Folgende Subsysteme werden zum Ablauf von SDF-P V2.4A benötigt:

BS2000/OSD-BC ≥ V5.0
SDF-P-BASYS V2.4A
SDF ≥ V4.5A
VAS ≥ V2.0A

Für spezielle Funktionen sind erforderlich:

PLAM / ILAM V3.1A
SDF-P-BIF ≥ V1.0B
JV ≥ V13.0D

17 Meldungen

Die Meldungen im Handbuch sind, wie in der Meldungsdatei auch, nach ihren Meldungsschlüsseln sortiert. Dabei rangieren Buchstaben vor Ziffern.

Im Folgenden werden die Systemmeldungen aufgelistet. Bei garantierten Meldungen wird das Meldungsattribut „Garantie“ (siehe Handbuch „Systemmeldungen“ [15]) mit der Zeile „◆ Warranty: Y“ nach dem Meldungstext dokumentiert.

Liste der Meldungen

```
SDPF001   INTERNAL ERROR IN MODULE '(&00)', INTERNAL ERROR NUMBER '(&01)'; ADDITIONAL
          INFORMATION: '(&02)'. CONTACT THE SYSTEM ADMINISTRATOR
SDPF001   INTERNER FEHLER IN MODULE '(&00)', INTERNE FEHLERNUMMER '(&01)'; WEITERE
          INFORMATION: '(&02)'. SYSTEMVERWALTER VERSTAENDIGEN
```

Bedeutung
Ein ENTRY wurde in die SERSLOG-Datei geschrieben.
Die Inserts sollen der Systemdiagnose helfen.

Maßnahme
Falls vorhanden, leiten Sie bitte folgende Unterlagen an die Systemdiagnose weiter:
- Fehlermeldung mit Inserts oder SERSLOG Datei,
- Systemdump,
- Prozedur, die den Fehler erzeugt hat.

```
SDP0001   INCONSISTENCY BETWEEN SDF-P SYNTAX FILE AND EXECUTION MODULE
SDP0001   INKONSISTENZ ZWISCHEN SDF-P-SYNTAXDATEI UND ABLAUFMODUL
```

Bedeutung
Moegliche Ursachen:
- ein Kommandoattribut wurde in der Benutzer-Syntaxdatei geaendert.
- das Kommando wird durch ein in der Benutzer-Syntaxdatei definiertes Kommando ausser Kraft gesetzt.
- eine Syntaxdatei einer aelteren SDF-P- oder SDF-P-BASYS-Version ist installiert.

Maßnahme
Kommando aus der Benutzer-Syntaxdatei entfernen oder Systemverwalter verstaendigen.

```
SDP0002   IRREGULAR PROCEDURE TERMINATION. CONTACT THE SYSTEM ADMINISTRATOR
SDP0002   IRREGULAERE PROZEDURBEENDIGUNG. SYSTEMVERWALTER VERSTAENDIGEN
```

Bedeutung
Es ist bei der Prozedurausfuehrung ein Systemfehler aufgetreten, der zu einem unerwarteten Abbruch der Prozedur fuehrte.

```
SDP0003   SYNTAX ERROR IN COMMAND
SDP0003   SYNTAX-FEHLER IN KOMMANDO
```

```
SDP0004   ERROR DETECTED AT COMMAND LINE: (&OO) IN PROCEDURE '(&O1)'
SDP0004   FEHLER GEFUNDEN IN KOMMANDOZEILE: (&OO) IN PROZEDUR '(&O1)'
```

Bedeutung
Die vorangehende Meldung bezieht sich auf die angegebene Zeile oder eine ihrer zugehoerigen Folgezeilen oder die Meldung bezieht sich auf eine abnormale Programmbeendigung in der vorletzten Zeile.

```
SDP0005   TOO MANY PARAMETERS SPECIFIED
SDP0005   ZUVIELE PARAMETER ANGEGEBEN
```

Bedeutung
Moegliche Ursachen:
- In der Prozedur sind weniger Parameter deklariert als im Aufruf angegeben.
- Die BUILTIN-Funktion erwartet weniger Parameter.

```
SDP0006   POSITIONAL PARAMETERS ARE NOT ALLOWED FROM PARAMETER NO. (&OO)
SDP0006   STELLUNGSPARAMETER SIND AB PARAMETER NUMMER (&OO) NICHT ERLAUBT
```

Bedeutung
Moegliche Ursachen:
- Stellungsparameter sind nach Schluesselwortparametern nicht erlaubt.
- Die Anzahl der moeglichen Stellungsparameter ist erreicht.

```
SDP0007   SYNTAX ERROR IN PARAMETER LIST, PARAMETER NO. (&OO)
SDP0007   SYNTAX-FEHLER IN PARAMETERLISTE BEI PARAMETER NUMMER (&OO)
```

```
SDP0008   INVALID KEYWORD '(&OO)' SPECIFIED IN PARAMETER LIST
SDP0008   FALSCHES SCHLUESSELWORT '(&OO)' IN PARAMETERLISTE ANGEGEBEN
```

Bedeutung
1. Das Schluesselwort ist weder ein gueltiger Parametername noch eine eindeutige Abkuerzung eines Parameternamens.
2. Das Schluesselwort ist weder ein gueltiger konstanter Parameterwert noch eine eindeutige Abkuerzung eines konstanten Parameterwerts.

```
SDP0009   PARAMETER '(&OO)' MUST BE SPECIFIED AT PROCEDURE CALL
SDP0009   DER PARAMETER '(&OO)' MUSS BEIM PROZEDURAUFRUF ANGEGEBEN WERDEN
```

```
SDP0010    TYPE OF PARAMETER '(&00)' INVALID
SDP0010    TYP DES PARAMETERS '(&00)' FALSCH
```

Bedeutung
Prozeduraufruf: Aktuell-Parameter ist nicht in Typ des Formal-Parameter konvertierbar.
Builtin Funktion: der Typ des als Parameter mitgegebenen Ausdrucks wird nicht erwartet oder es sind nur Schluesselwoerter an dieser Stelle zugelassen.
FOR-Schleife: FROM, TO, INCREMENT muessen arithmetische Ausdruecke sein.
CONDITION muss ein Boolescher Ausdruck sein.

```
SDP0011    ERROR IN OPERAND '(&00)'
SDP0011    FEHLER IM OPERANDEN '(&00)'

SDP0012    VALUE OF EXPRESSION LESS THAN ZERO
SDP0012    WERT DES AUSDRUCKS KLEINER ALS NULL

SDP0013    FILE NAME INVALID OR MISSING
SDP0013    DATEINAME FEHLERHAFT BZW. NICHT ANGEGEBEN

SDP0014    WARNING IN LINE: (&00) IN PROCEDURE '(&01)'
SDP0014    WARNUNG IN ZEILE: (&00) IN PROZEDUR '(&01)'
```

Bedeutung
Die vorangehende Meldung bezieht sich auf die angegebene Zeile oder eine ihrer zugehoerigen Folgezeilen.

```
SDP0015    OPERAND VALUE '(&00)' NOT ALLOWED; DEFAULT VALUE ASSUMED
SDP0015    OPERANDEN-WERT '(&00)' NICHT ERLAUBT; STANDARDWERT VERWENDET

SDP0016    AGGREGATES NOT ALLOWED
SDP0016    AGGREGATE NICHT ERLAUBT

SDP0017    NO STRUCTURED PROCEDURE
SDP0017    KEINE STRUKTURIERTE PROZEDUR
```

Bedeutung
Moegliche Ursachen:
- die Prozedur beginnt mit /BEGIN-PROC,
- der fuehrende Schraegstrich fehlt.

```
SDP0018    ERROR RAISED BY USER
SDP0018    FEHLER VOM BENUTZER ERZEUGT
```

Bedeutung
Der Fehler wurde durch das Kommando /RAISE-ERROR oder /EXIT-PROCEDURE explizit gesetzt.

```
SDP0019    THE FILE REFERENCED IN THE COMMAND CANNOT BE PROCESSED IN RFA MODE
SDP0019    DIE ANGEGEBENE PROZEDUR KANN IM RFA-MODUS NICHT AUSGEFUEHRT WERDEN

SDP0020    SYNTAX ERROR IN FILE NAME OR NAME OF LIBRARY ELEMENT
SDP0020    SYNTAX FEHLER BEI DER ANGABE DES DATEINAMENS ODER DES BIBLIOTHEKSELEMENTS
```

SDP0021 THE STRUCTURE OF THE FILE '(&00)' DOES NOT MATCH THE REQUIRED ACCESS METHOD
SDP0021 DIE STRUKTUR DER DATEI '(&00)' PASST NICHT ZUR GEWUENSCHTEN ZUGRIFFSMETHODE

SDP0022 COMMAND OR OPERAND VALUE '(&00)' NOT ALLOWED IN SDF-P BASIC-VERSION
SDP0022 KOMMANDO ODER OPERANDENWERT '(&00)' IST IN SDF-P GRUNDAUSBAU NICHT ERLAUBT

Bedeutung
Die Funktionalitaet von SDF-P ist nicht im vollem Umfang verfuegbar oder die benoetigte Version von SDF-P ist nicht verfuegbar.

Maßnahme
Systemverwalter auffordern, das SDF-P-Subsystem neu zu starten oder auf eine hoehere SDF-P-Version hochruesten.

SDP0023 PROCEDURE CANCELLED ON USER REQUEST
SDP0023 PROZEDUR AUF WUNSCH DES BENUTZERS ABGEBROCHEN

Bedeutung
Die Prozedur wird nach Bestaetigung des Benutzers abgebrochen, nachdem waehrend der Anforderung eines Prozedurparameterwerts am Bildschirm auf K2 gedrueckt wurde.

Maßnahme
Wenn gewuenscht, kann die Prozedur mit korrekter Parametereingabe wieder aufgerufen werden.

SDP0024 Error detected at command line: (&00) in current dialog block
SDP0024 Fehler gefunden in Kommandozeile: (&00) im laufenden Dialogblock

Bedeutung
Die vorangehende Meldung bezieht sich auf die angegebene Zeile oder eine ihrer zugehoerigen Folgezeilen oder die Meldung bezieht sich auf eine abnormale Programmbeendigung in der vorletzten Zeile.

SDP0025 Warning in line: (&00) in current dialog block
SDP0025 Warnung in Zeile: (&00) im laufenden Dialogblock

Bedeutung
Die vorangehende Meldung bezieht sich auf die angegebene Zeile oder eine ihrer zugehoerigen Folgezeilen.

```
SDP0026   Task specific default not allowed for this command
SDP0026   Task-spezifischer Standardwerte fuer dieses Kommando nicht zulaessig
```

Bedeutung
Task-spezifische Standardwerte (durch das Zeichen ! eingeleitet) sind
fuer folgende SDF-P-Kontrollflusskommandos nicht zulaessig:
- BEGIN-BLOCK
- BEGIN-PARAMETER-DECLARATION
- CYCLE
- ELSE, ELSE-IF
- END-BLOCK, END-FOR, END-IF, END-WHILE
- END-PARAMETER-DECLARATION
- EXIT-BLOCK
- FOR
- GOTO
- IF, IF-BLOCK-ERROR, IF-CMD-ERROR
- REPEAT
- UNTIL
- WHILE

Fuer folgende Kommandos sind sie ebenfalls unzulaessig:
- ADD-CJC-ACTION
- END-CJC-ACTION
- DECLARE-PARAMETER
- OPEN-VARIABLE-CONTAINER wenn in einem Parameter-Deklarationsblock angegeben (d.h. zwischen BEGIN-PARAMETER-DECLARATION und END-PARAMETER-DECLARATION)
- SET-PROCEDURE-OPTIONS

```
SDP0030   SDF-P version not compatible with SDF-P-BASYS version. SDF-P start rejected
SDP0030   SDF-P Version nicht kompatibel zur SDF-P-BASYS Version. SDF-P Start
          zurueckgewiesen

SDP0039   MORE THAN ONE VALUE SPECIFIED FOR OPERAND (&00). LAST VALUE IS USED
SDP0039   MEHR ALS EIN WERT FUER OPERANDEN (&00) ANGEGEBEN. LETZTER WERT WIRD VERWENDET

SDP0040   /FOR command overflows limit defined by system
SDP0040   /FOR Kommando ueberschreibet im System festgestellte Grenze
```

Bedeutung
Maximale Laenge fuer dieses Kommando: 4000 Zeichen.

```
SDP0089   INPUT ERROR
SDP0089   FEHLER BEI DER EINGABE
```

Bedeutung
Die Eingabe fehlt oder sie passt nicht zum geforderten Typ oder Format.

SDP0090 WARNING: AN ACTION WAS PERFORMED
SDP0090 WARNUNG: EINE AKTION WURDE DURCHGEFUEHRT

- Warranty: Y

SDP0091 SEMANTIC ERROR
SDP0091 SEMANTISCHER FEHLER

- Warranty: Y

SDP0092 RESOURCE CURRENTLY NOT AVAILABLE
SDP0092 BETRIEBSMITTEL ZUR ZEIT NICHT VERFUEGBAR

- Warranty: Y

SDP0093 ERROR DURING ACCESS OF FILE/LIBRARY '(&00)', ERROR '(&01)'. MORE INFORMATION: /HELP-MSG (&01)
SDP0093 FEHLER BEIM ZUGRIFF AUF DATEI/BIBLIOTHEK '(&00)', FEHLER '(&01)'. WEITERE INFORMATIONEN: /HELP-MSG (&01)

SDP0094 CONTAINER NOT ACCESSIBLE
SDP0094 KEIN ZUGRIFF AUF BEHAELTER MOEGLICH

Bedeutung
Zugriff auf Datei, Bibliothekselement oder Variable ist nicht moeglich.

SDP0095 UNEXPECTED ERROR RETURN CODE '(&00)' FROM COMPONENT '(&01)'. CONTACT SYSTEM ADMINISTRATOR
SDP0095 UNERWARTETER FEHLER-RETURNCODE '(&00)' VON KOMPONENTE '(&01)'. SYSTEMVERWALTER VERSTAENDIGEN

SDP0096 RECORD TOO LONG
SDP0096 DATENSATZ ZU LANG

Bedeutung
Beim Schreiben/Lesen ist ein Datensatz aufgetreten, der die zulaessige Maximallaenge ueberschreitet.

SDP0097 FILE, LIBRARY OR LIBRARY ELEMENT '(&00)' IS LOCKED
SDP0097 DATEI, BIBLIOTHEK ODER BIBLIOTHEKSELEMENT '(&00)' IST GESPERRT

SDP0098 FILE, LIBRARY OR LIBRARY ELEMENT '(&00)' DOES NOT EXIST
SDP0098 DATEI, BIBLIOTHEK ODER BIBLIOTHEKSELEMENT '(&00)' EXISTIERT NICHT

```
SDP0099    NO MORE VIRTUAL MEMORY SPACE AVAILABLE AT THIS MOMENT
SDP0099    KEIN VIRTUELLER SPEICHERPLATZ Z.ZT. VORHANDEN
```

- Warranty: Y

Bedeutung
Der virtuelle Adressraum ist erschoepft.
- die deklarierten Variablen benoetigen zuviel Speicherplatz;
- die Schachtelungstiefe der Prozedur ist zu gross;
- die Summe der Laenge von allen Variablen Werten des Auftrags ist groesser als das ADDRESS-SPACE-LIMIT Attribut des Benutzer:
  - boolescher Wert = 1 byte
  - Zahl = 4 bytes
  - String-Wert= Laenge des Strings
  - deklarierte Variable ohne Wert = 0 byte

Falls dieser Fehler beim Aufbau einer Struktur auftritt, so ist diese anschliessend nicht vollstaendig deklariert.

Maßnahme
- Speicherplatz freigeben durch /FREE-VARIABLE auf Variablen
- Prozedurschachtelung abbrechen mit /CANCEL-PROCEDURE
- Task abbrechen mit /LOGOFF.

```
SDP0100    Specified file '(&00)' is not a SAM/ISAM file
SDP0100    Angegebene datei '(&00)' ist kein SAM/ISAM datei

SDP0101    Operation name 'DECLARE-PARAMETER' unknown. Contact the system administrator
SDP0101    Operations-Name 'DECLARE-PARAMETER' unbekannt. Systemverwalter verstaendigen
```

Bedeutung
DECLARE-PARAMETER Kommando nicht bekannt bei SDF.
SDF-P-BASYS Syntax File ist nicht installiert.

```
SDP0102    CONTINUATION LINE MISSING, 'EOF' ASSUMED
SDP0102    FORTSETZUNGSZEILE FEHLT, 'EOF' ANGENOMMEN

SDP0103    BUFFER TOO SMALL FOR CONTINUATION LINES
SDP0103    PUFFER FUER FORTSETZUNGSZEILEN ZU KLEIN
```

Bedeutung
Maximale Laenge: 16364 Zeichen.

```
SDP0104    SLASHES MISSING
SDP0104    SCHRAEGSTRICHE FEHLEN
```

Bedeutung
Kommandos muessen mit einem Schraegstrich, Anweisungen mit zwei Schraegstrichen beginnen, Fortsetzungszeilen entsprechend.

```
SDP0108    SLASH MISSING
SDP0108    SCHRAEGSTRICH FEHLT
```

SDP0109 SLASHES PROHIBITED
SDP0109 SCHRAEGSTRICH VERBOTEN

SDP0110 COMMAND FOUND, STATEMENT EXPECTED
SDP0110 KOMMANDO GEFUNDEN, ANWEISUNG ERWARTET

SDP0111 THE COMMAND COULD NOT BE IDENTIFIED AS /DECLARE-PARAMETER NOR AS /OPEN-VARIABLE-CONTAINER COMMAND
SDP0111 DAS KOMMANDO KONNTE NICHT ALS /DECLARE-PARAMETER AUCH NICHT ALS /OPEN-VARIABLE-CONTAINER KOMMANDO ERKANNT WERDEN

Bedeutung
Im Prozedurkopf wurde ein /DECLARE-PARAMETER oder /OPEN-VARIABLE-CONTAINER Kommando erwartet, aber nicht erkannt.
Moegliche Ursachen:
- Schreibfehler im Operationsnamen.
- Eine mehrdeutige Abkuerzung wurde verwendet.
- Durch ein Gleichheitszeichen nach dem Operationsnamen wurde das Kommando als /SET-VARIABLE Kommando interpretiert.

SDP0112 SYNTAX ERROR IN SKIP LABEL
SDP0112 SYNTAX-FEHLER IM SKIP-LABEL

SDP0113 ERROR IN COMMAND NAME
SDP0113 FEHLER IM KOMMANDO-NAMEN

SDP0114 ODD NUMBER OF APOSTROPHES
SDP0114 UNGERADE ANZAHL VON APOSTROPHEN

Bedeutung
es wurden eventuell zu viele oder zu wenig apostrophe angegeben.

SDP0115 DOUBLE APOSTROPHE MISSING
SDP0115 DOPPELAPOSTROPH FEHLT

SDP0116 PARENTHESIS MISSING
SDP0116 KLAMMER FEHLT

SDP0117 UNEXPECTED PARENTHESIS
SDP0117 KLAMMER ZUVIEL

```
SDP0118    COMMAND INVALID IN CURRENT ENVIRONMENT '(&OO)'
SDP0118    KOMMANDO UNGUELTIG IN DER GEGENWAERTIGEN UMGEBUNG '(&OO)'
```

Bedeutung

- es wurden beim Aufruf mehr Parameter angegeben, als in der Prozedur deklariert wurde.
- /DECLARE-PARAMETER wegen eines Schreibfehlers nicht gefunden, daher kein Parameter erwartet.
- /BEGIN-PARAMETER-DECLARATION nicht gefunden weil ein anderes Kommando vorher gefunden war, daher wurden die Parameter-Deklarationen nicht gefunden.
- die Klammerung der Parameter-Deklarationen mit /BEGIN-PARAMETER-DECLARATION und /END-PARAMETER-DECLARATION wurde vergessen oder ist fehlerhaft, daher wurden die Parameter-Deklarationen nicht gefunden.
- ein prozedurbeendendes Kommando wurde gegeben, obwohl keine Prozedur aktiv war.
- ein blockbeendendes Kommando wurde angegeben, obwohl ein dazugehoeriger Block nicht geoeffnet war.
- ein Kontrollflusskommando wurde durch Fluchtsymbolersetzung erzeugt.
- das angegebene Kommando ist im Unterbrechungszustand nicht erlaubt.
- es wurde keine Struktur oder kein Layout initialisiert.
- ein Layout-Deklaration ist momentan nicht erlaubt.
- der Elementname darf nicht vom SDF-Datentyp composed-name sein.

```
SDP0119    INVALID COMMAND THAT SHOULD BE EXECUTED BY /EXEC-CMD
SDP0119    UNGUELTIGES KOMMANDO, DAS DURCH DAS /EXEC-CMD AUSGEFUEHRT WERDEN SOLL

SDP0130    SYNTAX ERROR IN EXPRESSION
SDP0130    AUSDRUCK SYNTAKTISCH FALSCH

SDP0131    RECORD TOO LONG
SDP0131    GENERIERTER SATZ ZU LANG
```

Bedeutung

Moegliche Ursachen:

- Der Ausdruck bzw. Wert des Ausdrucks ist zu lang.
- Nach der Fluchtsymbol-Ersetzung fiel der Satz zu lang aus.

Maximale Laenge: 16364 Bytes.

```
SDP0132    NO PROCEDURE BODY EXISTS
SDP0132    KEIN PROZEDUR-RUMPF VORHANDEN

SDP0133    INVALID TYPE OF EXPRESSION
SDP0133    AUSDRUCK HAT FALSCHEN TYP
```

```
SDP0134   DATA FOUND, COMMAND EXPECTED
SDP0134   DATENSATZ GEFUNDEN, KOMMANDO ERWARTET
```

Bedeutung
Der Satz beginnt nicht mit einem Schraegstrich, daher wird er nicht als Kommando erkannt.
Ein /SEND-DATA-Kommando wurde ausgefuehrt, obwohl keine Daten erwartet wurden.

```
SDP0135   DATA FOUND, STATEMENT EXPECTED
SDP0135   DATENSATZ GEFUNDEN, ANWEISUNG ERWARTET
```

Bedeutung
Der Satz beginnt nicht mit zwei Schraegstrichen, daher wird er nicht als eine Anweisung erkannt.
Ein /SEND-DATA-Kommando wurde ausgefuehrt, obwohl eine Anweisung erwartet wurde.

```
SDP0136   STATEMENT FOUND, COMMAND EXPECTED
SDP0136   ANWEISUNG GEFUNDEN, KOMMANDO ERWARTET
```

Bedeutung
Der Satz beginnt mit zwei Schraegstrichen, daher wird er nicht als ein Kommando erkannt.
Ein /SEND-STMT-Kommando wurde ausgefuehrt, obwohl ein Kommando erwartet wurde.

```
SDP0137   STATEMENT FOUND, DATA EXPECTED
SDP0137   ANWEISUNG GEFUNDEN, DATENSATZ ERWARTET
```

Bedeutung
Der Satz beginnt mit zwei Schraegstrichen, daher wird er nicht als Datensatz erkannt.
Ein /SEND-STMT-Kommando wurde ausgefuehrt, obwohl Daten erwartet wurden.

```
SDP0138   ERROR DURING PREANALYSIS OF THE PROCEDURE
SDP0138   FEHLER WAEHREND DER PROZEDUR-VORANALYSE
```

Bedeutung
Die Strukturierung der Prozedur mittels der Kontrollflusskommandos ist fehlerhaft. Die Prozedur wurde nicht ausgefuehrt.

Maßnahme
Aendern Sie die Prozedurstrukturierung.

```
SDP0139   BACK BRANCH LIMIT REACHED
SDP0139   MAXIMALE ANZAHL DER RUECKWAERTSSPRUENGE ERREICHT
```

Bedeutung
Das durch /MODIFY-PROCEDURE-TEST-OPTIONS voreingestellte BACK-BRANCH-LIMIT wurde erreicht.

```
SDP0140   SYNTAX ERROR DURING ESCAPE CHARACTER REPLACEMENT
SDP0140   SYNTAX-FEHLER WAEHREND DER FLUCHTSYMBOL-ERSETZUNG
```

- Warranty: Y

```
SDP0141   SEMANTIC ERROR DURING ESCAPE-CHARACTER REPLACEMENT
SDP0141   SEMANTISCHER FEHLER WAEHREND DER FLUCHTSYMBOL-ERSETZUNG
```

```
SDP0142    LABEL IGNORED IN PROCEDURE HEAD
SDP0142    LABEL IM PROZEDURKOPF IGNORIERT

SDP0143    /CYCLE ONLY ALLOWED IN LOOPS
SDP0143    /CYCLE NUR IN SCHLEIFEN ERLAUBT

SDP0144    ERROR DURING PARAMETER TRANSFER
SDP0144    FEHLER WAEHREND DER PARAMETERUEBERGABE
```

Bedeutung
Wenn die Fehlerursache nicht aus vorangegangenen Fehlermeldungen erkennbar ist, so ist noch folgender Fall moeglich:
Es wurde vergessen die Parameterdeklarationen durch /BEGIN-PARAMETER-DECLARATION und /END-PARAMETER-DECLARATION zu klammern, dadurch wird nur die erste Deklaration ausgewertet.

```
SDP0145    SPECIFICATION OF LIBRARY-ELEMENT VERSION NOT ALLOWED FOR NOT-STRUCTURED
           PROCEDURES
SDP0145    ANGABE VON BIBLIOTHEKSELEMENT VERSION FUER NICHT STRUKTURIERTE PROZEDUREN NICHT
           ERLAUBT
```

Bedeutung
Fuer nicht Strukturierte Prozeduren, nur der Standardwert (*HIGHEST-EXISTING) ist erlaubt.

```
SDP0150    INCORRECT SYMBOL AT THIS POSITION IN EXPRESSION : (&00)
SDP0150    ZEICHEN AN DIESER POSITION IM AUSDRUCK NICHT ERLAUBT: (&00).

SDP0151    INCORRECT SYMBOL IN HEX-STRING: (&00).
SDP0151    NICHT ERLAUBTES ZEICHEN IM HEX-STRING: (&00).
```

Bedeutung
Es sind nur Zahlen und die Buchstaben A-F bzw. a-f erlaubt.

```
SDP0152    INCORRECT SYMBOL FOLLOWING INTEGER: (&00)
SDP0152    NICHT ERLAUBTES ZEICHEN NACH EINER ZAHL: (&00)
```

Bedeutung
Es muss ein Blank zwischen Zahl und Buchstabe sein.

```
SDP0153    ERROR IN NAME: (&00)
SDP0153    FEHLER IM NAMEN: (&00)
```

Bedeutung
Ein Teilname darf nicht mit einem Punkt anfangen. Dem Zeichen "-" darf kein Punkt folgen.

```
SDP0154    ERROR IN NAME PART: (&00).
SDP0154    FEHLER IM TEILNAMEN: (&00)
```

Bedeutung
Einem Punkt darf nur ein Buchstabe oder ein Sonderzeichen folgen.

```
SDP0155    STACK OVERFLOW DURING EXPRESSION ANALYSIS
SDP0155    STACK UEBERLAUF WAEHREND DER AUSDRUCKS-ANALYSE

SDP0156    SYNTACTICAL ERROR IN EXPRESSION: (&OO)
SDP0156    SYNTAKTISCHER FEHLER IM AUSDRUCK: (&OO)

SDP0157    OPERATOR NOT ALLOWED IN SDF-P BASIC-VERSION
SDP0157    OPERATOR NICHT IN SDF-P GRUNDAUSBAU ERLAUBT

SDP0158    ERROR: EXPRESSION EMPTY (LENGTH = 0).
SDP0158    FEHLER: AUSDRUCK LEER (LAENGE = 0).

SDP0200    LABEL '(&OO)' OCCURS MORE THAN ONCE
SDP0200    LABEL '(&OO)' TRITT MEHRFACH AUF

SDP0201    INVALID BLOCK-CLOSING COMMAND USED
SDP0201    FALSCHES BLOCK-ABSCHLIESSENDES KOMMANDO BENUTZT
```

Bedeutung
Der aktuelle offene Block darf nicht mit diesem blockbeendenden Kommando abgeschlossen werden.

```
SDP0202    BLOCK-CLOSING COMMAND EXISTS, BLOCK-OPENING COMMAND MISSING
SDP0202    BLOCK-ABSCHLIESSENDES KOMMANDO VORHANDEN, BLOCK-EROEFFNENDES KOMMANDO FEHLT
```

Bedeutung
Ein blockbeendendes Kommando wurde gefunden, ohne dass ein blockbeginnendes Kommando vorgelegen hat.

```
SDP0203    CURRENT BLOCK NOT AN IF BLOCK
SDP0203    AKTUELLER BLOCK KEIN IF-BLOCK
```

Bedeutung
Ein /ELSE oder /ELSE-IF Kommando wurde ausserhalb eines IF-Blocks gefunden.

```
SDP0204    /ELSE COMMAND ALREADY PRESENT
SDP0204    /ELSE-KOMMANDO BEREITS VORHANDEN
```

Bedeutung
Das /ELSE-Kommando des IF-Blockes wurde bereits gefunden.
Ein weiteres /ELSE oder /ELSE-IF Kommando ist nicht erlaubt.

```
SDP0205    SKIP LABEL NOT ALLOWED
SDP0205    'SKIP LABEL' NICHT ERLAUBT
```

Bedeutung
SKIP LABELs duerfen nicht gebraucht werden:
- innerhalb von Bloecken
- vor SDF-P Kontrollflusskommandos.

SDP0207 Not all control structures closed
SDP0207 Nicht alle Kontrollstrukturen geschlossen

Bedeutung
Es fehlt ein END-BLOCK, END-IF, END-FOR, END-WHILE oder UNTIL Kommando.

SDP0208 TARGET LABEL '(&00)' NOT DEFINED
SDP0208 ZIEL-LABEL '(&00)' NICHT DEFINIERT

SDP0209 INVALID TARGET OF JUMP
SDP0209 UNGUELTIGES SPRUNGZIEL

Bedeutung
Ursachen:
- Sprungziel liegt nicht in der aktuellen Blockhierarchie
- Der Operand des /CYCLE- oder /EXIT-BLOCK-Kommandos bezeichnet keinen Blocknamen.
- Sprungziel des /INCLUDE-BLOCK-Kommandos ist kein /BEGIN-BLOCK-Kommando mit Blocknamen.

SDP0210 /CYCLE OR /EXIT COMMAND NOT WITHIN A BLOCK
SDP0210 /CYCLE ODER /EXIT KOMMANDO AUSSERHALB VON BLOECKEN

Bedeutung
Kein blockbeginnendes Kommando liegt fuer /CYCLE oder /EXIT vor.

SDP0211 Only messages of the class SDP may be suppressed
SDP0211 Nur Meldungen der Klasse SDP duerfen unterdrueckt werden

SDP0212 /LOGON COMMAND DETECTED IN OR BEFORE PROCEDURE HEAD
SDP0212 /LOGON-KOMMANDO LIEGT IM ODER VOR PROZEDUR-KOPF

Bedeutung
Ein /LOGON Kommando ist an dieser Stelle nicht erlaubt.

SDP0213 COMMAND NOT YET IMPLEMENTED
SDP0213 KOMMANDO NOCH NICHT IMPLEMENTIERT

SDP0215 LABEL OF BLOCK-CLOSING COMMAND DOES NOT MATCH LABEL OF BLOCK-OPENING COMMAND
SDP0215 LABEL DES BLOCK-ABSCHLIESSENDEN KOMMANDOS PASST NICHT ZUM LABEL DES BLOCK-EROEFFNENDEN KOMMANDOS

```
SDP0216   THIS PROCEDURE CANNOT BE CALLED WITH THIS COMMAND
SDP0216   DIESE PROZEDUR KANN MIT DIESEM KOMMANDO NICHT AUFGERUFEN WERDEN
```

Bedeutung
/CALL-Kommando durch das /SET-PROCEDURE-OPTIONS-Kommando verboten.
/DO-Kommando wurde verwendet, die Prozedur hat aber Parameter mit 'TRANSFER-TYPE=BY-REFERENCE'. Alte Prozeduren nicht mit /INCLUDE-PROC rufen.

Maßnahme
Falls der Aufruf mit /CALL-PROCEDURE versucht wurde, sollte es nochmals mit /INCLUDE-PROCEDURE versucht werden.
Falls der Aufruf mit /INCLUDE-PROCEDURE versucht wurde, sollte es nochmals mit /CALL-PROCEDURE versucht werden.

```
SDP0217   THIS COMMAND IS NOT ALLOWED WITH THE OPERAND 'STRUCTURE-OUTPUT=VARNAME'
SDP0217   DIESES KOMMANDO IST MIT OPERAND 'STRUCTURE-OUTPUT=VARNAME' NICHT ERLAUBT
```

Bedeutung
Nur die Kommandos /SHOW-FILE-ATTRIBUTES mit INFORMATION=NAME-AND-SPACE/ ALL-ATTRIBUTES und /SHOW-JOB-STATUS sind zusammen mit einer strukturierten Variablen zulaessig.

```
SDP0218   VARIABLE MUST BE A LIST OF TYPE 'STRUCTURE'
SDP0218   VARIABLE MUSS EINE LISTE VOM TYP 'STRUCTURE' SEIN

SDP0219   ERROR DURING PROMPTING
SDP0219   FEHLER WAEHREND DES PROMPTINGS
```

Bedeutung
Fehlermeldung von TIAM beim Schreiben/Lesen.
Moegliche Fehlerursache:
- Prompting im Stapelbetrieb
- Prompt-String zu lang, ...

```
SDP0221   TOO MUCH INFORMATION. /EXEC-CMD NOT PROCESSED
SDP0221   ZUVIEL INFORMATION. /EXEC-CMD NICHT AUSGEFUEHRT

SDP0222   OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR '(&00)'. IN SYSTEM MODE: /HELP-MSG
          (&00)
SDP0222   OPERAND 'CMD' IM /EXEC-CMD FEHLERHAFT, FEHLER '(&00)'. IM SYSTEM-MODUS: /HELP-
          MSG (&00)
```

Bedeutung
Naehere Information ueber den Fehlerschluessel kann ueber /HELP-MSG im Systemmodus erfragt werden.

```
SDP0223   MODIFICATION OF CONTROL FLOW COMMANDS BY 'SYSTEM-EXIT' NOT ALLOWED
SDP0223   VERAENDERUNGEN VON KONTROLLFLUSS-KOMMANDOS DURCH 'RZ-EXIT' NICHT ERLAUBT
```

Bedeutung
Durch ein RZ-EXIT wurde ein Kommando modifiziert, das nicht modifiziert werden durfte. Das gilt vor allem fuer alle SDF-P-Kontrollflusskommandos.

Maßnahme
Systemverwalter verstaendigen, um den RZ-EXIT zu korrigieren oder zu deaktivieren.

```
SDP0224   LOGGING SUPPRESSED; CONTAINER '(&00)' IS READ PROTECTED
SDP0224   PROTOKOLLIERUNG UNTERDRUECKT; BEHAELTER '(&00)' IST LESE-GESCHUETZT

SDP0225   DO YOU WANT TO CANCEL ALL ACTIVATED PROCEDURES? REPLY (Y=YES; N=NO)
SDP0225   MOECHTEN SIE ALLE LAUFENDEN PROZEDUREN ABBRECHEN? ANTWORT (Y=JA; N=NEIN)

SDP0226   COMMAND NOT ALLOWED FOR /EXECUTE-CMD
SDP0226   KOMMANDO FUER /EXECUTE-CMD NICHT ZUGELASSEN

SDP0227   WARNING WHILE EXECUTING THE OPERAND 'CMD'
SDP0227   WARNUNG BEI DER AUSFUEHRUNG DES OPERANDEN 'CMD'

SDP0228   TOO MANY VARIABLES ARE DECLARED. COMMAND IS ABORTED
SDP0228   ZU VIELE VARIABLEN WURDEN ANGELEGT. KOMMANDO WURDE ABGEBROCHEN

SDP0229   ALL OPERANDS OF /EXECUTE-CMD IGNORED FOR COMMAND SPECIFIED BY 'CMD'. PROCESSING
          CONTINUES
SDP0229   ALLE OPERANDEN VON /EXECUTE-CMD WERDEN FUER DAS IN 'CMD' ANGEGEBENEN KOMMANDO
          IGNORIERT. VERARBEITUNG WIRD FORTGESETZT
```

Bedeutung
Das im Operanden 'CMD' angegebene Kommando kann nicht von /execute-cmd bearbeitet werden. Deswegen werden alle Operanden von /execute-cmd ignoriert und das Kommando wird separat ausgefuehrt.

Maßnahme
Fuer solche Kommandos, bitte benutzen Sie prozedurweite Funktionen:
* text-output-> /assign-sysout to=*variable(...)
* struct-output-> /assign-stream sysinf,to=*var(...)
* msg-struct-output -> /assign-stream sysmsg,to=*var(...)
* returncode-> /save-returncode (nach dem Kommando)
* returncode=*none-> /if-cmd-error ;end-if

und geben Sie das Kommando direkt ein.

SDP0230 Output larger than supported by /EXECUTE-CMD. /EXECUTE-CMD repeated; processing continues
SDP0230 Ausgabe groesser als von /EXECUTE-CMD unterstuetzt. /EXECUTE-CMD wiederholt; Verarbeitung fortgesetzt

Bedeutung
Die Ausgabe des Kommandos, das von /EXECUTE-CMD ausgefuehrt wird, ist groesser als die Standardlaenge, die von /EXECUTE-CMD vorgegeben wird (392.6 KBytes). Das Kommando wird von /EXECUTE-CMD mit einem groesseren Systempuffer wiederholt, solange die Ausgabe des Kommandos die angegebene Laenge ueberschreitet.

Maßnahme
Die Menge der Ausgabedaten kann mit einer genaueren Angabe der Operanden INFORMATION und SELECT des ausgefuehrten Kommandos verringert werden, wenn das Kommando diese Moeglichkeiten anbietet.
Wenn die Wiederhohlung des Kommandos auf jeden Fall vermieden werden soll, sollte /EXECUTE-CMD durch die passende Kombination der Kommandos /ASSIGN-SYSOUT, /ASSIGN-STREAM und /IF-CMD-ERROR ersetzt werden.

SDP0231 OUTPUT OF SPECIFIED CMD LARGER THAN ACTUALLY SUPPORTED BY SYSTEM. CMD REJECTED
SDP0231 AUSGABE DES ANGEGEBENEN KDOS GROESSER ALS VOM SYSTEM UNTERSTUETZT. KDO ZURUECKGEWIESEN

Bedeutung
Die Ausgabe des angegebenen Kommandos ist groesser als die fuer /EXECUTE-CMD maximal vorgesehene Laenge (6.3 MBytes).
Deswegen kann die Ausgabe nicht vollstaendig in dem von /EXECUTE-CMD angelegten Systempuffer aufgenommen werden.
Das Kommando wurde ausgefuehrt, aber die Ausgabe kann nicht vollstaendig ausgewertet werden. Deswegen wird das Kommando zurueckgewiesen.

Maßnahme
Wenn das im /EXECUTE-CMD angegebene Kommando Ausgabesteuerungs-Operanden besitzt (z.B. "INFORMATION" bzw. "SELECT"), sollten diese Operanden angegeben werden, im die Ausgabe kleiner zu machen.
Wenn das nicht moeglich ist, sollte das /EXECUTE-CMD durch die passende Kombination der Kommandos /ASSIGN-SYSOUT, /ASSIGN-STREAM und /IF-CMD-ERROR ersetzt werden.

SDP0232 ERROR IN OPERAND '(&OO)'
SDP0232 FEHLER IM OPERANDEN '(&OO)'

SDP0234 OPERAND 'NAME' INVALID
SDP0234 OPERAND 'NAME' UNGUELTIG

SDP0237 OPERAND 'OUTPUT' INVALID
SDP0237 OPERAND 'OUTPUT' UNGUELTIG

```
SDP0239   ERROR DURING EVALUATION OF RIGHT SIDE OF ASSIGNMENT
SDP0239   FEHLER WAEHREND DER AUSWERTUNG DER RECHTEN SEITE DER ZUWEISUNG

SDP0250   Cmd or event not allowed in /INCLUDE-CMD context
SDP0250   Kdo bzw. Ereignis in /INCLUDE-CMD Umgebung nicht erlaubt
```

Bedeutung
Das Kommando oder das Ereignis wird in einer Prozedur bzw. Prozedurhierarchie, die von /INCLUDE-CMD aufgerufen wird, nicht zugelassen, weil es zu Inkonsistenzen fuehren wuerde.
Z.B: /RESUME-PROGRAM, /SEND-MESSAGE, /START-PROGRAM, /CANCEL-PROCEDURE, AID Kommandos... wuerden zu Inkonsistenzen fuehren und werden deswegen abgewiesen.
/INCLUDE-CMD-Kommandos duerfen nicht geschachtelt werden.

Maßnahme
Das Kommando bzw. das Ereignis soll nicht eingegeben werden.
Die Prozedur, die solche Kommandos bzw Ereignisse enthaelt, soll nicht waehrend eines /INCLUDE-CMD-Prozeduraufrufs aufgerufen werden.

```
SDP0251   Cmd or event not allowed in operand 'CMD' of /INCLUDE-CMD
SDP0251   Kdo bzw. Ereignis im /INCLUDE-CMD-Operanden 'CMD' nicht erlaubt
```

Bedeutung
Das Kommando oder das Ereignis ist im Operanden 'CMD' des Kommandos /INCLUDE-CMD nicht erlaubt.

Maßnahme
Das Kommando bzw. Ereignis soll aus der Liste der im /INCLUDE-CMD-Kommandos angegebenen Kommandos entfernt werden.

```
SDP0252   Operand 'CMD' invalid
SDP0252   Operand 'CMD' ungueltig
```

Bedeutung
Der Wert des Operanden 'CMD' soll in Klammern angegeben werden und mindestens einen Buchstaben enthalten.

Maßnahme
Operandenwert korrigieren und die Eingabe wiederholen.

SDP0253 /INCLUDE-CMD not allowed in this mode
SDP0253 /INCLUDE-CMD in diesem Modus nicht erlaubt

Bedeutung
/INCLUDE-CMD darf nur im Programm-Modus mit einem CMD-Makro oder einer //EXECUTE-SYSTEM-CMD-Anweisung angegeben werden.

Maßnahme
Einschraenkungen bei der Angabe des Kommandos /INCLUDE-CMD bitte beachten!

SDP0254 All procedures cancelled due to program termination; processing continues
SDP0254 Alle Prozeduren werden wegen Programmbeendigung geloescht; Verarbeitung wird fortgesetzt

Bedeutung
Waehrend der Ausfuehrung der Prozeduren, die von /INCLUDE-CMD aufgerufen werden, wird das aufrufende Programm beendet.
Um Inkonsistenzen zu vermeiden, die wegen des Loeschen des aufrufenden Programms verursacht werden koennen, werden alle aktiven Prozeduren geloescht.

Maßnahme
Die Ursache der Programmbeendigung soll vom Programmierer festgestellt werden, um ein solches Ereignis zu vermeiden.

SDP0255 /RESUME-PROGRAM replaced by /RESUME-PROCEDURE; processing continues
SDP0255 /RESUME-PROGRAM durch /RESUME-PROCEDURE ersetzt; Verarbeitung wird fortgesetzt

Bedeutung
Nach der Unterbrechung einer Prozedur, die von /INCLUDE-CMD aufgerufen wird, soll nicht das aufrufende Programm fortgesetzt werden, sondern die unterbrochene Prozedur.

Maßnahme
Machen Sie sich keine Sorgen, das System laeuft einfach weiter.

SDP0256 K2 request ignored in /INCLUDE-CMD context
SDP0256 K2-Taste in /INCLUDE-CMD-Umgebung nicht zugelassen

Bedeutung
Die Aufforderung zur Eingabe von Prozedurparametern darf in Prozeduren in der /INCLUDE-CMD-Umgebung nicht mit K2 geloescht werden, da /CANCEL-PROCEDURE in dieser Umgebung nicht unterstuetzt wird.

Maßnahme
Bitte geben Sie einen ungueltigen Wert an, um den Abbruch der Prozedur zu erzwingen.

SDP0257 Error in /INCLUDE-CMD cmd
SDP0257 Fehler in /INCLUDE-CMD-Kdo

Bedeutung
Die Ausfuehrung eines Kommandos, das von /INCLUDE-CMD angegeben wird, meldet einen Fehler.

Maßnahme
Fehlerursache beheben und/oder Fehler durch Einfuehrung eines Fehlerbearbeitungsblocks abfangen (IF-CMD-ERROR).

SDP0258 Warning in /INCLUDE-CMD cmd
SDP0258 Warnung im /INCLUDE-CMD-Kdo

Bedeutung
Eine Warnung wurde von einem Kommando gemeldet, das im /INCLUDE-CMD angegeben wurde.

Maßnahme
Die Ursache der Warnung sollte gegebenenfalls entfernt werden.

SDP0259 Operation aborted: selection ignored
SDP0259 Vorgang abgebrochen: Auswahl ignoriert

Bedeutung
Die Abbruchtaste wurde am Bildschirm gedrueckt.
Die getroffene Auswahl wird ignoriert, kein Element wird geliefert.

SDP0260 OPERAND MESSAGE IGNORED. PROCESSING CONTINUES
SDP0260 OPERAND MESSAGE IGNORIERT. VERARBEITUNG WIRD FORTGESETZT

Bedeutung
Der Operand MESSAGE kann nicht mit der gegenwaertigen SDF Version verarbeitet werden.

Maßnahme
Verwenden Sie eine neuere Version von SDF, die diesen Operanden verarbeiten kann.

SDP0300 OPERAND(S) IN EXPRESSION DO NOT MATCH OPERATOR
SDP0300 OPERAND(EN) IM AUSDRUCK PASSEN NICHT ZUM OPERATOR

Bedeutung
Ein oder mehrere Operanden des Ausdrucks haben einen zum Rechenoperator nicht passenden Typ.

SDP0301 INVALID VALUE IN EXPRESSION
SDP0301 FALSCHER WERT IM AUSDRUCK

SDP0302 ILLEGAL DIVISION BY ZERO
SDP0302 DIVISION DURCH NULL UNZULAESSIG

```
SDP0303   ERROR IN VARIABLE NAME '(&00)'
SDP0303   FEHLER IM VARIABLEN-NAMEN '(&00)'

SDP0304   OVERFLOW, NUMBER OUT OF RANGE
SDP0304   UEBERLAUF, ZAHL AUSSERHALB DES MOEGLICHEN WERTEBEREICHS
```

Bedeutung
Es sind nur Werte zwischen -2**31 und 2**31-1 zugelassen.
Builtin-Funktion USER-SWITCH: es gibt nur 32 Schalter (Index 0-31),
der gewuenschte Index ist zu gross oder zu klein.

```
SDP0305   VALUE WAS TRUNCATED BECAUSE BUFFER TOO SMALL
SDP0305   WERT WURDE GEKUERZT, DA PUFFER ZU KLEIN
```

Bedeutung
Der intern definierte Puffer ist zu klein.

```
SDP0306   BUFFER TOO SMALL. OPERATION OR ASSIGNMENT NOT EXECUTED
SDP0306   PUFFER ZU KLEIN. OPERATION BZW. ZUWEISUNG NICHT AUSGEFUEHRT

SDP0307   Error in variable '(&00)'
SDP0307   Fehler in Variablen '(&00)'
```

Bedeutung
Ein Fehler wurde waehrend der Bearbeitung der angegebenen Variablen
durch SDF-P entdeckt.

Maßnahme
Die Variable und/oder die weiteren Bearbeitungsschritte muessen korrigiert werden.

```
SDP0310   EVALUATION ERROR
SDP0310   FEHLER WAEHREND DER AUSDRUCKS-AUSWERTUNG

SDP0373   DESCRIPTION OF SYSTEM INTERFACE MODIFIED IN SYNTAX FILE
SDP0373   BESCHREIBUNG DER SYSTEMSCHNITTSTELLE IN SYNTAXDATEI MODIFIZIERT
```

Bedeutung
Moegliche Ursachen:
- ein Kommandoattribut wurde in der Benutzer-Syntaxdatei geaendert.
- das Kommando wird durch ein in der Benutzer-Syntaxdatei definiertes Kommando ausser Kraft gesetzt.

```
SDP0402   INVALID FUNCTION NAME. FUNCTION '(&00)' DOES NOT EXIST
SDP0402   FALSCHER FUNKTIONSNAME. FUNKTION '(&00)' EXISTIERT NICHT

SDP0403   Specified string is too long (more than 4 characters)
SDP0403   Spezifizierte Zeichenkette zu lang (mehr als 4 Buchstaben)

SDP0410   INCONSISTENCY BETWEEN 'WHEN' AND 'THEN' PARAMETERS
SDP0410   INKONSISTENZ ZWISCHEN 'WHEN' UND 'THEN' PARAMETERN

SDP0411   STRING EMPTY
SDP0411   STRING LEER
```

SDP0412 START POSITION OUT OF RANGE
SDP0412 START-POSITION AUSSERHALB MOEGLICHEN BEREICHS

Bedeutung
Die Startposition ist groesser als die Stringlaenge bzw. <= Null.

SDP0413 ILLEGAL LENGTH
SDP0413 NICHT ZULAESSIGE LAENGENANGABE

SDP0414 Warning: *REST-LENGTH value used for LENGTH operand
SDP0414 Warnung: *REST-LENGTH Werte ist fuer Operand LENGTH verwendet

SDP0415 SYNTAX ERROR: INTEGER EXPECTED IN STRING. CONVERSION NOT POSSIBLE
SDP0415 SYNTAX-FEHLER: ZAHL IM STRING ERWARTET. KONVERTIERUNG NICHT MOEGLICH

SDP0416 NUMBER OUT OF RANGE
SDP0416 ZAHL AUSSERHALB ZULAESSIGEN WERTEBEREICHS

Bedeutung
INTEGER-TO-CHARACTER() :
- Es sind nur Werte zwischen 0 und 255 zugelassen.
INTEGER() :
- Es sind nur Werte zwischen -2**31 und 2**31-1 zugelassen.

SDP0417 SPECIFIED STRING EMPTY. FUNCTION NOT EXECUTED
SDP0417 ANGEGEBENER STRING LEER. FUNKTION NICHT AUSGEFUEHRT

SDP0418 INVALID MSG-IDENTIFICATION
SDP0418 UNGUELTIGE MELDUNGSSCHLUESSEL

SDP0419 JV: JOB VARIABLE '(&OO)' NOT ACCESSIBLE
SDP0419 JV: AUF JOBVARIABLE '(&OO)' KANN NICHT ZUGEGRIFFEN WERDEN

Bedeutung
Moegliche Ursachen:
- Die Jobvariable ist nicht shareable.
- Die Jobvariable ist im Zugriff geschuetzt.

SDP0420 JV: JOB VARIABLE '(&OO)' DOES NOT EXIST
SDP0420 JV: JOBVARIABLE '(&OO)' EXISTIERT NICHT

SDP0421 JV: DMS ERROR '(&OO)' WHILE ACCESSING JOB VARIABLE. IN SYSTEM MODE: /HELP-MSG DMS(&OO)
SDP0421 JV: DMS-FEHLER '(&OO)' BEIM ZUGRIFF AUF JOBVARIABLE. IM SYSTEM-MODUS: /HELP-MSG DMS(&OO)

Bedeutung
Naehere Information ueber den DMS-Fehlerschluessel kann ueber /HELP-MSG im Systemmodus erfragt bzw. dem BS2000-Handbuch 'Systemmeldungen' entnommen werden.

SDP0422 KEYWORD '(&OO)' UNKNOWN FOR THIS FUNCTION
SDP0422 SCHLUESSELWORT '(&OO)' UNBEKANNT FUER DIESE FUNKTION

```
SDP0423   VARIABLE '(&00)' NOT AN ARRAY
SDP0423   VARIABLE '(&00)' KEIN ARRAY

SDP0424   NO CONTAINER ASSIGNED TO VARIABLE '(&00)'
SDP0424   VARIABLER '(&00)' KEIN BEHAELTER ZUGEWIESEN

SDP0425   BUILTIN FUNCTION VAR-ATTRIBUTES: VARIABLE NOT A STRUCTURE
SDP0425   BUILTIN-FUNKTION VAR-ATTRIBUTES: VARIABLE IST KEINE STRUKTUR
```

Bedeutung
Die Variable ist keine Struktur, deshalb kann die gewuenschte Information nicht gegeben werden (2.Parameter: *STRUCTURE-INFORMATION).

```
SDP0426   VARIABLE '(&00)' NOT A LIST
SDP0426   VARIABLE '(&00)' KEINE LISTE

SDP0427   ATTRIBUTE '(&00)' UNKNOWN
SDP0427   ATTRIBUT '(&00)' UNBEKANNT

SDP0428   COMMAND RETURN CODE NOT AVAILABLE IN DIALOG
SDP0428   KOMMANDO-RETURN-CODE KANN IM DIALOG NICHT ABGEFRAGT WERDEN

SDP0429   CONVERSION NOT POSSIBLE
SDP0429   KONVERTIERUNG NICHT MOEGLICH

SDP0430   VAR-ATTRIBUTE: CONTAINER HAS NO SCOPE
SDP0430   VAR-ATTRIBUTE: BEHAELTER HAT KEINEN SCOPE

SDP0431   ERROR '(&00)' IN BUILTIN FUNCTION '(&01)'
SDP0431   FEHLER '(&00)' IN BUILTIN-FUNKTION '(&01)'

SDP0432   FUNCTION NOT ALLOWED FOR LISTS
SDP0432   FUNKTION FUER LISTEN NICHT ERLAUBT

SDP0433   GIVEN STRING NOT A C-LITERAL
SDP0433   EINGEGEBENER STRING KEIN C-LITERAL

SDP0434   GIVEN STRING NOT A X-LITERAL
SDP0434   EINGEGEBENER STRING KEIN X-LITERAL

SDP0435   DESIRED INFORMATION NOT AVAILABLE
SDP0435   GEWUENSCHTE INFORMATION NICHT VERFUEGBAR

SDP0436   GIVEN LENGTH NOT BETWEEN ZERO AND MAXIMUM POSSIBLE STRING LENGTH
SDP0436   ANGEGEBENE LAENGE LIEGT NICHT ZWISCHEN NULL UND MAXIMAL MOEGLICHER STRING-LAENGE

SDP0437   LENGTH OF PARAMETER 'FILL-BYTE' EQUAL TO ZERO
SDP0437   LAENGE DES PARAMETERS 'FILL-BYTE' GLEICH NULL

SDP0438   LENGTH OF PARAMETER 'SEPARATOR' EQUAL TO ZERO
SDP0438   LAENGE DES PARAMETERS 'SEPARATOR' GLEICH NULL

SDP0439   LENGTH OF FILE NAME ZERO OR GREATER THAN 54
SDP0439   LAENGE DES DATEI-NAMENS NULL ODER GROESSER ALS 54
```

SDP0440 Name '(&00)' not a file name or not a specific file name
SDP0440 Name '(&00)' kein Dateiname oder kein spezifischer Dateiname

SDP0441 DMS ERROR '(&00)' WHEN CALLING FSTAT MACRO. IN SYSTEM MODE: /HELP-MSG (&00)
SDP0441 DVS-FEHLER '(&00)' BEIM AUFRUF DES FSTAT-MAKROS. IM SYSTEM-MODUS: /HELP-MSG (&00)

Bedeutung
Naehere Information ueber den DVS-Fehlerschluessel kann ueber /HELP-MSG im Systemmodus erfragt bzw. dem BS2000-Handbuch 'Systemmeldungen' entnommen werden.

SDP0442 LAYOUT DOES NOT EXIST
SDP0442 LAYOUT EXISTIERT NICHT

SDP0443 SYNTAX OF PATTERN IS NOT A CORRECT WILDCARD SYNTAX
SDP0443 SYNTAX DES MUSTERS IST KEINE KORREKTE WILDCARD-SYNTAX

SDP0444 INTERFACE ERROR CONCERNING BUILTIN FUNCTION. CONTACT SYSTEM ADMINISTRATOR
SDP0444 SCHNITTSTELLENFEHLER ZUR BUILTIN-FUNKTION. SYSTEMVERWALTER VERSTAENDIGEN

SDP0445 USERID WITH MORE THAN 8 CHARACTERS IS NOT POSSIBLE
SDP0445 USERID MIT MEHR ALS 8 ZEICHEN IST NICHT MOEGLICH

SDP0446 USERID UNKNOWN
SDP0446 USERID UNBEKANNT

SDP0447 THE GIVEN STRING IS NO SDF-LIST
SDP0447 DER ANGEGEBENE STRING IST KEINE SDF-LISTE

SDP0448 THE PARAMETER 'NUMBER' OUT OF RANGE
SDP0448 DER PARAMETER 'NUMBER' AUSSERHALB DES MOEGLICHEN WERTEBEREICHS

Bedeutung
Es sind weniger Unterlisten vorhanden als 'NUMBER' angibt.
'NUMBER' ist kleiner gleich 0.

SDP0449 ARRAY ELEMENTS ARE NOT ALLOWED
SDP0449 ARRAYELEMENTE SIND NICHT ZUGELASSEN

SDP0450 USERID LOCKED
SDP0450 BENUTZERKENNUNG GESPERRT

SDP0451 NOT ENOUGH SPACE FOR STORING RESULT
SDP0451 NICHT GENUEGEND SPEICHERPLATZ FUER DAS ERGEBNIS VORHANDEN

Bedeutung
Es wurde nicht genuegend Speicherplatz zum Speichern des Ergebnisses zugewiesen.

```
SDP0452   INVALID DATE
SDP0452   UNGUELTIGES DATUM
```

Bedeutung
Das Eingabe-/Ausgabedatum ist fehlerhaft, niedriger als 1582-10-15 oder groesser als 9999-12-31.

```
SDP0453   (&00) PARAMETER IS EMPTY OR ITS LENGTH IS GREATER THAN (&01) CHARACTER(S) OR
          CONTAINS ONE OR MORE SPACES
SDP0453   PARAMETER (&00) IST LEER ODER LAENGER ALS (&01) ZEICHEN ODER ENTHAELT
          LEERZEICHEN
```

```
SDP0454   INVALID PARAMETER : '(&00)'
SDP0454   UNGUELTIGER PARAMETER: '(&00)'
```

Bedeutung
Der Parameter enthaelt einen Syntaxfehler.

```
SDP0455   INVALID LENGTH OF INPUT STRING (ALLOWED : 1..256)
SDP0455   EINGABESTRING MIT UNZULAESSIGER LAENGE (ERLAUBT: 1..256)
```

Bedeutung
Der Eingabestring ist entweder leer oder laenger als 256 Zeichen.

Maßnahme
Entweder den leeren String auffuellen oder den zu Langen String kuerzen.

```
SDP0456   LENGTH PARAMETER IS OUT OF RANGE (ALLOWED : 1..256)
SDP0456   PARAMETER LENGTH AUSSERHALB DES GUELTIGEN WERTEBEREICHS
```

Bedeutung
Der Parameter LENGTH hat entweder den Wert Null oder einen Wert groesser 256.

Maßnahme
Parameterwert zwischen 1 und 256 angeben.

```
SDP0457   LENGTH PARAMETER IS NOT A NUMERIC VALUE
SDP0457   PARAMETER LENGTH HAT NICHT-NUMERISCHEN WERT
```

Bedeutung
Wahrscheinlich enthaelt der Parameter LENGTH nicht-numerische Zeichen.

Maßnahme
Numerischen Parameterwert angeben.

```
SDP0458   BEGIN-DATE GREATER THAN END-DATE
SDP0458   ANFANGSDATUM GROESSER ALS ENDEDATUM
```

Maßnahme
Datumsangaben vertauschen.

```
SDP0459   Parameter error or invalid parameters combination. Additional information:
          '(&00)'
SDP0459   Parameter Fehler oder ungueltig Parameter Kombination. Weitere Information:
          '(&00)'
```

Bedeutung
Erklaerung:
Weitere Information =
1 falscher INPUT Parameter
2 interner Fehler
3 ungueltige Kombination von DATA-TYPE und/oder PATTERN mit einem von den folgenden Parameter CAT-ID, USER-ID, VERSION, GENERATION, WILDCARD, KEYSTAR, SEPARATORS, UNDERSCORE, ODD, CORRECTION-STATE, USER-INTERFACE, ALIAS, VOLUME-ONLY oder ungueltige Kombination von den Parametern DEVICE-CLASS, EXCEPT-DISKS, EXCEPT-TAPES
4 falsche LOWEST-LENGTH/HIGHEST-LENGTH Grenze (HIGHEST < LOWEST, >SDF-A Grenze, falscher Wert, keine dezimal Grenze, keine Grenze erlaubt, ...)
5 falscher PATTERN (Laenge = 0, Syntax)
6 falscher Wert (VALUE Parameter).

Maßnahme
Eingabe korrigieren und wiederholen.

```
SDP0460   THE GIVEN STRING IS NOT A STRUCTURE
SDP0460   DER EINGEGEBENE STRING IST KEINE STRUKTUR

SDP0461   THE NUMERIC VALUE FOR THE OPERAND MUST BE GREATER THAN ZERO
SDP0461   NUMERISCHER OPERANDENWERT MUSS GROSSER NULL SEIN

SDP0462   '(&00)' IS NOT A STRUCTURED-NAME
SDP0462   '(&00)' IST KEIN STRUKTURNAME

SDP0463   The given operand '(&00)' is unknown
SDP0463   Der angegebene Operand '(&00)' ist unbekannt

SDP0464   TOO MANY AMBIGUITIES FOR THE GIVEN OPERAND
SDP0464   OPERANDENANGABE ENTHAELT ZU VIELE MEHRDEUTIGKEITEN

SDP0465   OPERAND OF TYPE BOOLEAN NOT ALLOWED
SDP0465   OPERAND VOM TYP BOOLEAN HIER UNZULAESSIG

SDP0467   NO NAME FOUND; PROCESSING CONTINUES
SDP0467   KEINEN NAMEN GEFUNDEN: VERARBEITUNG WIRD FORTGESETZT

SDP0468   NO NAME FOUND
SDP0468   KEINEN NAMEN GEFUNDEN
```

```
SDP0469   Invalid parameter '(&00)' specified
SDP0469   Der angegebene Parameter '(&00)' ist ungueltig
```

Bedeutung
Die Beschreibung der GETINSP-Schnittstelle im IMON Handbuch enthaelt naehere Angaben zu diesem Parameter.

Maßnahme
Eingabe korrigieren und wiederholen.

```
SDP0470   Internal error returned by GETINSP/GETINSV interface. Return code '(&00)'
          received
SDP0470   Interner Fehler beim GETINSP-/GETSINV-Schnittstelleaufruf. Returncode '(&00)'
```

Bedeutung
Der GETINSP/GETINSV Schnittstelle hat einen unerwarteten Returncode geliefert.

Maßnahme
Verstaendigen Sie den Systemverwalter.

```
SDP0471   SDF-P VERSION NOT SUPPORTED BY SDF-P-BIF
SDP0471   SDF-P VERSION WIRD VON SDF-P-BIF NICHT UNTERSTUETZT
```

Bedeutung
Die Version der Systemschnittstelle, die von SDF-P benutzt wird, wird von SDF-P-BIF nicht unterstuetzt. Alle built-in Funktionen, die mit SDF-P-BIF geladen wurden, werden ignoriert.

Maßnahme
Bitte benutzen Sie eine andere Version des SDF-P-BIF Kernes, die die laufende SDF-P Version unterstuetzt. Siehe SDF-P Manual fuer gueltige SDF-P/SDF-P-BIF Versionsraster.

```
SDP0472   Null byte (x'00') not allowed in STRING and FIELD-SEPARATOR operands
SDP0472   DIE OPERANDEN STRING UND FIELD-SEPARATOR DUERFEN KEIN NULLBYTE (X'00') ENTHALTEN

SDP0473   Internal error during builtin function
SDP0473   Interner Fehler beim Ausfuehren einer Builtin-Funktion

SDP0474   Syntax error in regular expression for operand FIELD-SEPARATOR
SDP0474   Syntaxfehler in regulaerem Ausdruck fuer Operand FIELD-SEPARATOR

SDP0475   Variable must be a STRUCTURE or a LIST/ARRAY
SDP0475   Variable muss Struktur, Liste oder Array sein

SDP0476   Result string too long
SDP0476   Ergebnisstring zu lang
```

Bedeutung
Der Ergebnisstring darf maximal 4096 Zeichen lang sein.

```
SDP0477    Incorrect SDF structure
SDP0477    Fehlerhafte SDF-Struktur

SDP0478    Invalid STRING syntax in *STRING-TO-VARIABLE value
SDP0478    Syntaxfehler im STRING-Wert fuer *STRING-TO-VARIABLE

SDP0479    Invalid STRING syntax in *STRING-TO-VARIABLE value near '(&00)'
SDP0479    Syntaxfehler im STRING-Wert for *STRING-TO-VARIABLE bei '(&00)'

SDP0480    Incorrect SDF structure for variable '(&00)'
SDP0480    Fehlerhafte SDF-Struktur fuer Variable '(&00)'

SDP0481    Value of operand 'POSITION' must be greater than zero
SDP0481    'POSITION' muss groesser Null sein

SDP0482    An input string is too long (1..255)
SDP0482    Ein Eingabe String ist zu lang (1..255)
```

Bedeutung
Die Operanden
- INPUT-NAME
- WILDCARD-PATTERN
- CONSTRUCTION-WILDCARD

duerfen nicht groesser als 255 Zeichen sein.

```
SDP0483    Incorrect CONSTRUCTION-WILDCARD value
SDP0483    Falscher CONSTRUCTION-WILDCARD Wert

SDP0484    Too long result string (1..255)
SDP0484    Zu groesser Ausgabe String (1..255)
```

Bedeutung
Der Ausgabe String darf nicht groesser als 255 Zeichen sein.

```
SDP0485    Value of operand 'FIELD-NUMBER' must be greater than zero
SDP0485    'FIELD-NUMBER' muss groesser Null sein

SDP0486    Odd number of apostrophes in STRING value
SDP0486    Ungerade Anzahl von Apostrophen im STRING Wert

SDP0487    Invalid type of expression in *STRING-TO-VARIABLE() operand
SDP0487    Ausdruck hat falschen typ fuer operand *STRING-TO-VARIABLE()

SDP0488    Spaces not allowed in input strings
SDP0488    Blanks nicht erlaubt im Eingabe String
```

Bedeutung
Die Operanden
- INPUT-NAME
- WILDCARD-PATTERN
- CONSTRUCTION-WILDCARD

duerfen kein Blank (' ') enthalten.

SDP0489 Warning: Installation-Unit '(&00)' not found in IMON Software Inventory. Default value assumed
SDP0489 Warnung: Installation-Unit '(&00)' nicht gefunden im IMON Software Inventory. Standardwert wird benutzt

SDP0490 Installation-Unit '(&00)' version '(&01)' not found
SDP0490 Installation-Unit '(&00)' Version '(&01)' nicht gefunden

SDP0491 Warning: Logical-id '(&00)' not found in Installation-Unit '(&01)' version '(&02)'. Default value assumed
SDP0491 Warnung: Logical-Id '(&00)' nicht gefunden im Installation-Unit '(&01)' Version '(&02)'. Standard Wert wird benutzt

SDP0492 Null byte (x'00') not allowed in PATTERN operand and list elements
SDP0492 Der Operand PATTERN und die Listenelemente duerfen kein Nullbyte (x'00') enthalten

SDP0493 Value of operands BEGIN-INDEX, END-INDEX, BEGIN-COLUMN and END-COLUMN must be greater than zero
SDP0493 BEGIN-INDEX, END-INDEX, BEGIN-COLUMN und END-COLUMN muessen groesser Null sein

SDP0494 Syntax error in regular expression for operand PATTERN
SDP0494 Syntaxfehler in regulaerem Ausdruck fuer operand PATTERN

SDP0495 '(&00)' not a correct JV name
SDP0495 '(&00)' ist kein korrekter JV-Name

Bedeutung
Der JV-Name (&00) muss ein full-filename_1..54 sein.

SDP0496 User is not privileged to see installation information
SDP0496 Benutzer hat keine Berechtigung, um auf alle Informationen zuzugreifen

SDP0497 No path name assigned to logical-id '(&00)'
SDP0497 Fuer logischer Name '(&00)' kein zugeordeneter Pfadname vorhanden

SDP0498 BEGIN-COLUMN must not be greater than END-COLUMN.
SDP0498 BEGIN-COLUMN darf nicht als END-COLUMN groesser sein.

SDP0499 FIRST-RECORD must not be higher than LAST-RECORD.
SDP0499 FIRST-RECORD darf nicht als LAST-RECORD hoeher sein.

SDP0510 Stream name '(&00)' invalid
SDP0510 Ungueltiger Stromname '(&00)'

Bedeutung
Es wurde ein reservierter Stromname verwendet (SYSDTA, SYSCMD, SYSOUT, SYSIPT, SYSOPT oder mit '$' beginnender Name).

Maßnahme
Stromnamen ueberpruefen und Eingabe wiederholen.

```
SDP0511   Assignment invalid for stream '(&00)'
SDP0511   Ungueltige Zuweisung des Stromes '(&00)'
```

Bedeutung
Der angegebene Strom konnte nicht zugewiesen werden:
- Zuweisung des Vaterstroms fuehrt zu Schleife ueber vorher zugewiesene Stroeme
- ...

Maßnahme
Anderen Vaterstrom zuweisen
Fehlerursache beheben und Eingabe wiederholen.

```
SDP0512   Server no longer active, S-stream reset to *DUMMY. /TRANSMIT-BY-STREAM command
          ignored, processing continues
SDP0512   Server nicht mehr aktiv, Strukturierter-Strom auf *DUMMY gesetzt. /TRANSMIT-BY-
          STREAM Kommando wird ignoriert, Verarbeitung wird fortgesetzt
```

Bedeutung
Der Server ist zum Sendezeitpunkt in keinem gueltigen Zustand. Der Server hat die zu diesem Strom gehoerenden Informationen geloescht. Daher ist keine Uebertragung dieses Stroms mehr moeglich.

Maßnahme
Server-Umgebung ueberpruefen und Strom in gueltiger Umgebung zuweisen.

```
SDP0513   System stream '(&00)' cannot be deleted
SDP0513   SYSTEM STREAM '(&00)' KANN NICHT GELOESCHT WERDEN

SDP0514   Stream '(&00)', assigned to another one, cannot be deleted
SDP0514   STREAM '(&00)' IST EINEM ANDEREN STREAM ZUGEWIESEN UND DARF DESHALB NICHT
          GELOESCHT WERDEN
```

Bedeutung
Der Strom kann nicht geloescht werden, weil der einem anderen Strom (Sohnstrom) zugewiesen worden ist. Das Loeschen wuerde den Sohn Strom ohne Bestimmung lassen.

Maßnahme
Bitte weisen Sie den Sohnstrom auf eine andere Bestimmung oder *dummy (der Sohnstrom kann mit /show-stream-assignment gefunden werden). Wiederholen Sie dann das /assign-stream.

```
SDP0515   Stream not created at primary level
SDP0515   STREAM NICHT AUF PRIMARY EBENE GESCHAFFEN
```

Bedeutung
Stroeme duerfen nur auf Primary Ebene geloescht werden. Das vermeidet, dass die Wirkung eines Loeschens von der Prozedur Umgebung abhaengt.

Maßnahme
Bitte geben Sie /delete-stream nach Ausgang aller Prozeduren oder wenn alle aufrufenden Prozeduren seit Primary Ebene system-file-context=*same gesetzt haben.

```
SDP0516   Stream '(&00)' does not exist, processing continues
SDP0516   STREAM '(&00)' IST NICHT GESCHAFFEN; VERARBEITUNG WIRD FORTGESETZT

SDP0517   SPECIFIED STREAM '(&00)' DOES NOT EXIST
SDP0517   ANGEGEBENER STROM '(&00)' EXISTIERT NICHT
```

- Warranty: Y

Bedeutung
Der Strom ist im laufenden Arbeitskontext unbekannt (Prozedur oder Dialog).

Maßnahme
Der Strom muss vor Bearbeitung mit /ASSIGN-STREAM geschaffen werden.

```
SDP0518   NO MATCH FOR SPECIFIED WILDCARD PATTERN, PROCESSING CONTINUES
SDP0518   KEIN TREFFER FUER WILDCARD-MUSTER, VERARBEITUNG WIRD FORTGESETZT

SDP0519   NO MATCH FOR WILDCARD PATTERN
SDP0519   KEIN TREFFER FUER WILDCARD-MUSTER

SDP0520   Specified variable '(&00)' invalid
SDP0520   Angegebene Variable '(&00)' ist ungueltig
```

Bedeutung
Die Variable muss den Typ STRUCTURE haben.

Maßnahme
Variablentyp ueberpruefen.

```
SDP0522   Transmitted data incompatible with format processed by server
SDP0522   Sendedaten nicht kompatibel mit von Server verarbeitetem Format
```

Bedeutung
Der Server konnte den aktuellen Auftrag aus einem der folgenden Gruende nicht ausfuehren:
- der Server erwartet einen anderen Namen der Sendedaten
- in den gesendeten Variablen fehlen Daten
- die Attribute der Sendedaten entsprechen nicht den vom Server erwarteten Attributen.

```
SDP0524   Server no longer active, S-stream reset to *DUMMY
SDP0524   Server nicht mehr aktiv, S-Strom auf *DUMMY zurueckgesetzt
```

Bedeutung
Der Server-Status ist zum Uebertragungszeitpunkt nicht mehr gueltig.
Der Server hat die fuer diesen Strom relevanten Informationen geloescht.
Daher ist keine weitere Uebertragung ueber diesen Strom moeglich.

Maßnahme
Serverumgebung ueberpruefen und Strom in einer fuer diesen Strom gueltigen Umgebung zuweisen!

```
SDP0530   Server '(&00)' does not exist
SDP0530   Server '(&00)' existiert nicht
```

```
SDP0531   Warning returned by server
SDP0531   Server sendet Warnung
```

Bedeutung
Der Server hat eine Warnung gesendet.
Naehere Angaben werden in der Variablen RET-CONTROL-VAR-NAME abgelegt, sofern dies in den Kommandos /ASSIGN-STREAM und /TRANSMIT-BY-STREAM angegeben wurde.

Maßnahme
Inhalt der Variablen RET-CONTROL-VAR-NAME ueberpruefen.

```
SDP0532   Error returned by server
SDP0532   Server meldet Fehler
```

Bedeutung
Der Server hat einen Fehler gemeldet.
Naehere Angaben werden in der Variablen RET-CONTROL-VAR-NAME abgelegt, sofern dies in den Kommandos /ASSIGN-STREAM und /TRANSMIT-BY-STREAM angegeben wurde.

Maßnahme
Inhalt der Variablen RET-CONTROL-VAR-NAME ueberpruefen.

```
SDP0533   Server linkage error
SDP0533   Bindefehler des Servers
```

```
SDP0534   INTERNAL ERROR RETURNED BY SERVER
SDP0534   INTERNER FEHLER VOM SERVER ZURUECKGEGEBEN
```

Bedeutung
Server ist wegen unerwarteten Ereignisses oder Mangels an- oder nicht Vorhandenseins von- Systemmitteln abgebrochen.

```
SDP0535   WARNING RETURNED BY SERVER AT DELETION OF STREAM '(&00)', PROCESSING CONTINUES
SDP0535   LOESCHEN DES STREAMS '(&00)' MIT WARNUNG VOM SERVER, VERARBEITUNG WIRD
          FORTGESETZT

SDP0536   Error returned by server, delete rejected for stream '(&00)'
SDP0536   LOESCHEN DES STREAMS '(&00)' ZURUECKGEWIESEN WEGEN FEHLERS VOM SERVER

SDP0537   Internal error returned by server, delete rejected for stream '(&00)'
SDP0537   LOESCHEN DES STREAMS '(&00)' ZURUECKGEWIESEN WEGEN SERVER INTERNEN FEHLERS

SDP0538   Recursive call of stream server rejected
SDP0538   Rekursiver Aufruf vom Server zurueckgewiesen
```

Bedeutung
Der Server von strukturierten Stroeme, der durch
/ASSIGN-STREAM ...TO=*VARIABLE(...) gesetzt wurde,
hat einen ruecklaeufigen Aufruf mit einem Fehler Zustand entdeckt
und bricht diesen Aufruf durch Loeschen jeder neuen Operation ab.
Der erste Aufruf meldet einen internen Fehler.

Maßnahme
Der ruecklaeufige Aufruf wurde wahrscheinlich von der laufenden Benutzer-Umgebung verursacht.
Der Server benutzt MIP- und SDF-P-Dienste, die ueber die strukturierten Standardstroeme (SYSINF, SYSMSG) Daten uebergeben, dabei wurden diese Stroeme diesem Server zugewiesen.
Wird die Ursache des ersten Fehlers beseitigt, so kann der ruecklaeufige Aufruf vermieden werden.

```
SDP1006   INTERNAL ERROR IN VARIABLE HANDLER. CONTACT SYSTEM ADMINISTRATOR
SDP1006   INTERNER FEHLER IM VARIABLEN-HANDLER. SYSTEMVERWALTER VERSTAENDIGEN

SDP1007   No variable declared
SDP1007   Noch keine Variable angelegt
```

Bedeutung
Der Variablen-Pool ist leer.

```
SDP1008   VARIABLE/LAYOUT '(&00)' DOES NOT EXIST
SDP1008   VARIABLE BZW LAYOUT '(&00)' EXISTIERT NICHT
```

- Warranty: Y

```
SDP1010   VARIABLE '(&00)' HAS NO VALUE
SDP1010   VARIABLE '(&00)' HAT KEINEN WERT

SDP1014   NO STRUCTURE OR LAYOUT DECLARATION INITIATED
SDP1014   KEINE STRUKTUR- BZW. KEINE LAYOUT-DEKLARATION EINGELEITET
```

```
SDP1015    STRUCTURE CLOSED INTERNALLY
SDP1015    STRUKTUR INTERN GESCHLOSSEN
```

Bedeutung
Am Ende der Prozedur waren noch nicht alle mit /BEGIN-STRUCTURE deklarierten Strukturen durch /END-STRUCTURE geschlossen.

```
SDP1017    WARNING: VARIABLE '(&00)' ALREADY EXISTS
SDP1017    WARNUNG: VARIABLE '(&00)' EXISTIERT BEREITS
```

```
SDP1018    VARIABLE '(&00)' ALREADY EXISTS BUT WITH OTHER ATTRIBUTES
SDP1018    VARIABLE '(&00)' EXISTIERT BEREITS MIT ANDEREN ATTRIBUTEN
```

- Warranty: Y

Maßnahme
Variable umbenennen.

```
SDP1019    ARRAY BOUND OR LIMIT OUT OF RANGE
SDP1019    ARRAY-GRENZE ODER LIMIT AUSSERHALB DES MOEGLICHEN WERTEBEREICHS
```

```
SDP1020    UPPER LIMIT OF ARRAY OR LIST ELEMENTS OF VARIABLE '(&00)' REACHED
SDP1020    MAXIMALE ANZAHL DER ARRAY- ODER LISTENELEMENTE DER VARIABLEN '(&00)' ERREICHT
```

```
SDP1022    JOB VARIABLE '(&00)' NOT ACCESSIBLE
SDP1022    AUF DIE JOBVARIABLE '(&00)' KANN NICHT ZUGEGRIFFEN WERDEN
```

Bedeutung
Moegliche Ursachen:
- Password wurde nicht angegeben.
- Nur Lese-Zugriff moeglich.

```
SDP1023    JOB VARIABLE '(&00)' ALREADY EXISTS
SDP1023    JOBVARIABLE '(&00)' EXISTIERT BEREITS
```

Bedeutung
Es wurde versucht, eine bereits existierende Jobvariable als Behaelter neu anzulegen.

Maßnahme
STATE-Angabe des Behaelters ueberpruefen und den Wert STATE=OLD oder STATE=ANY angeben.

```
SDP1024    JOB VARIABLE '(&00)' DOES NOT EXIST
SDP1024    JOBVARIABLE '(&00)' EXISTIERT NICHT
```

Maßnahme
STATE=NEW oder STATE=ANY angeben bzw. die Jobvariable anlegen.

```
SDP1026    STRING TOO LONG FOR JOB VARIABLE '(&00)'
SDP1026    ZEICHENKETTE FUER JOBVARIABLE '(&00)' ZU LANG
```

Bedeutung
Die Zeichenkette hat mehr als 256 Zeichen.

```
SDP1027    VALUE FOR JOB VARIABLE '(&00)' IS NOT A STRING
SDP1027    WERT FUER JOBVARIABLE '(&00)' IST KEIN STRING
```

Bedeutung
Variablen, die eine Jobvariable als Behaelter haben, duerfen nur als Strings definiert werden.

```
SDP1029    VALUE TYPE NOT ALLOWED FOR VARIABLE 'SYSSWITCH'
SDP1029    WERTE TYP NICHT FUER VARIABLE 'SYSSWITCH' ERLAUBT
```

Bedeutung
Der Wert muss vom Typ BOOLEAN sein.

```
SDP1030    CONTAINER / VARIABLE-CONTAINER '(&00)' DOES NOT EXIST
SDP1030    BEHAELTER / VARIABLENBEHAELTER '(&00)' EXISTIERT NICHT
```

- Warranty: Y

Bedeutung
- Variablen muessen existieren, bevor sie als Behaelter verwendet werden koennen.
- Variablenbehaelter ist geschlossen worden.

```
SDP1031    ATTRIBUTES OF CONTAINER DO NOT MATCH THOSE OF DECLARED VARIABLE
SDP1031    ATTRIBUTE DES BEHAELTERS STIMMEN NICHT MIT DENEN DER ZU DEKLARIERENDEN VARIABLEN
           UEBEREIN

SDP1032    VARIABLE '(&00)' ALREADY EXISTS
SDP1032    VARIABLE '(&00)' EXISTIERT BEREITS

SDP1033    Scope of container is smaller than scope of variable '(&00)'
SDP1033    Scope des Behaelters ist kleiner als der Scope der Variablen '(&00)'

SDP1034    TYPE OF CONTAINER '(&00)' DOES NOT MATCH TYPE OF VARIABLE
SDP1034    TYP DES BEHAELTERS '(&00)' STIMMT NICHT MIT DEM DER VARIABLE UEBEREIN

SDP1035    THIS ELEMENT CANNOT BE DECLARED IMPLICITLY
SDP1035    DIESES ELEMENT KANN NICHT IMPLIZIT ANGELEGT WERDEN

SDP1036    VARIABLE TYPE AND VALUE TYPE DO NOT MATCH
SDP1036    VARIABLEN-TYP UND WERT-TYP PASSEN NICHT ZUSAMMEN

SDP1037    Variable '(&00)' cannot have or get a value
SDP1037    Werte fuer Variable '(&00)' nicht erlaubt
```

Bedeutung
Die Variable (&00) bezeichnet eine Struktur bzw. ein Array oder eine Liste.
Ein einfacher Wert (d.h. String, Integer oder Boolean) oder der Name einer Variablen, die einen einfachen Wert enthaelt, wurde erwartet.
Wenn die Variable (&00) als Parameter zu einer vordefinierten Funktion angegeben wird, der einen Name einen Struktur, Array order Liste erwartet, muss der Variablename (&00) zwischen Hochkommas geschrieben werden (z.B. '(&00)').

```
SDP1038   /FREE-VARIABLE NOT POSSIBLE FOR VARIABLE 'SYSSWITCH'
SDP1038   AUF VARIABLE 'SYSSWITCH' DARF KEIN /FREE-VARIABLE ANGEWENDET WERDEN

SDP1039   '#' MISSING OR MISUSED
SDP1039   '#' FEHLT ODER WIRD NICHT RICHTIG VERWENDET
```

Bedeutung
Es wurde ein '#' im Variablennamen angegeben, obwohl die Variable kein Array oder keine Liste ist.
Es wurde das '#' vergessen, obwohl die Variable ein Array oder eine Liste ist.

```
SDP1040   VARIABLE '(&OO)' MUST BE A VARIABLE OR LIST OF TYPE 'STRING'
SDP1040   VARIABLE '(&OO)' MUSS EINE VARIABLE ODER LISTE VOM TYP 'STRING' SEIN
```

Bedeutung
Die Variable hat einen Jobvariablen-Behaelter und muss vom Typ String sein.
Fuer /SHOW-VARIABLE muss die OUTPUT- bzw. INPUT-Variable eine Liste vom Typ String sein.
Fuer /READ-VARIABLE muss die INPUT-Variable eine Liste vom Typ String sein.
Fuer /EXEC-CMD muss die TEXT-OUTPUT-Variable eine Liste vom Typ String sein
Fuer /CALL-PROCEDURE muss die Variable eine Liste vom Typ String sein.

```
SDP1041   STRUCTURE/ARRAY/LIST '(&OO)' DOES NOT EXIST
SDP1041   STRUKTUR/ARRAY/LISTE '(&OO)' EXISTIERT NICHT

SDP1042   AGGREGATE NODE '(&OO)' DOES NOT EXIST
SDP1042   AGGREGAT-KNOTEN '(&OO)' EXISTIERT NICHT

SDP1043   VARIABLE '(&OO)' MUST NOT BE AN ARRAY ELEMENT OR A LIST ELEMENT
SDP1043   VARIABLE '(&OO)' DARF KEIN ARRAY- ODER LISTEN-ELEMENT SEIN

SDP1044   POOL ID INVALID, VARIABLES POOL DOES NOT EXIST. CONTACT SYSTEM ADMINISTRATOR
SDP1044   POOL-ID FALSCH, VARIABLEN-POOL EXISTIERT NICHT. SYSTEMVERWALTER VERSTAENDIGEN

SDP1045   VARIABLE OR ELEMENT OF VARIABLE '(&OO)' CANNOT BE DECLARED IMPLICITLY
SDP1045   VARIABLE ODER ELEMENT DER VARIABLEN '(&OO)' KANN NICHT IMPLIZIT DEKLARIERT
          WERDEN

SDP1046   NO IMPLICIT DECLARATION ALLOWED
SDP1046   IMPLIZITE DEKLARATION NICHT ERLAUBT

SDP1047   STRUCTURE '(&OO)' NOT COMPLETE. MEMORY SPACE SHORTAGE
SDP1047   STRUKTUR '(&OO)' NICHT VOLLSTAENDIG. ADRESSRAUM-MANGEL
```

Bedeutung
Waehrend der Strukturdeklaration war kein adressraum mehr vorhanden.

```
SDP1048   LAYOUT OF VARIABLE '(&OO)' DOES NOT EXIST
SDP1048   LAYOUT DER VARIABLE '(&OO)' EXISTIERT NICHT

SDP1049   LAYOUT ALREADY EXISTS BUT WITH OTHER ATTRIBUTES
SDP1049   LAYOUT EXISTIERT BEREITS MIT ANDEREN ATTRIBUTEN
```

```
SDP1050    LAYOUT DOES NOT EXIST
SDP1050    LAYOUT EXISTIERT NICHT

SDP1052    AGGREGATE ELEMENT NOT PRESENT
SDP1052    AGGREGATELEMENT NICHT VORHANDEN
```

Bedeutung
Von einem Aggregat wird das erste, letzte oder naechste Element gesucht
und
nicht gefunden.

```
SDP1054    JOB VARIABLE ERROR: JVS ERROR CODE '(&00)' WHILE ACCESSING JOB VARIABLE '(&01)'.
           IN SYSTEM MODE: /HELP-MSG JVS(&00)
SDP1054    JOBVARIABLEN-FEHLER: JVS-FEHLERCODE '(&00)' BEI ZUGRIFF AUF JOBVARIABLE
           '(&01)'. IN SYSTEM-MODUS: /HELP-MSG JVS(&00)

SDP1056    SYNTAX ERROR IN SET-VARIABLE COMMAND
SDP1056    SYNTAX-FEHLER IM SET-VARIABLE KOMMANDO
```

Bedeutung
Syntaxfehler beim SET-VARIABLE-Kommando:
- fehlerhafter Variablenname oder
- fehlerhafter Wert im WRITE-MODE-Operanden oder
- ...

Maßnahme
Kommandosyntax korrigieren.

```
SDP1057    NAME '(&00)' TOO LONG
SDP1057    NAME '(&00)' ZU LANG
```

Bedeutung
Die Variable existiert, aber es kann nicht auf sie zugegriffen werden.
Moegliche Ursachen:
- Auf Grund der Indexauswertung (bei Arrays) wurde der Name zu lang.
- Der Name des Behaelters ist laenger als der Name der Variablen.

```
SDP1058    NOT ENOUGH MEMORY SPACE FOR VALUE BUFFER
SDP1058    ADRESSRAUM FUER WERTE-PUFFER ZU KLEIN
```

Bedeutung
Es ist nicht genuegend Adressraum fuer den Wert vorhanden.

```
SDP1059    NOT ENOUGH MEMORY SPACE FOR NAME BUFFER AND VALUE BUFFER
SDP1059    ADRESSRAUM FUER NAME- UND WERTE-PUFFER ZU KLEIN
```

Bedeutung
Es ist nicht genuegend Adressraum fuer Name und Wert vorhanden.

```
SDP1060    WARNING: VARIABLE OR LAYOUT '(&00)' IMPORTED
SDP1060    WARNUNG: VARIABLE BZW. LAYOUT '(&00)' IMPORTIERT
```

```
SDP1061    ERROR WHILE IMPORTING LAYOUT OF VARIABLE '(&00)'
SDP1061    FEHLER BEI IMPORT DES LAYOUTS DER VARIABLEN '(&00)'

SDP1063    STRUCTURE NOT OPENED
SDP1063    STRUKTUR NICHT GEOEFFNET
```

Bedeutung
Die Struktur wurde noch nicht geoeffnet.
Ein /END-STRUCTURE ist zuviel.

```
SDP1064    LAYOUT OF VARIABLE '(&00)' NOT CLOSED
SDP1064    LAYOUT DER VARIABLEN '(&00)' NICHT GESCHLOSSEN
```

Bedeutung
Moegliche Ursachen:
- Vor dem ersten Zugriff muss das Layout geschlossen sein.
- Ein oder mehrere /END-STRUCTURE fehlen.

```
SDP1065    STEM '(&00)' OF AGGREGATE ELEMENT IS NOT A STRUCTURE
SDP1065    STAMM '(&00)' DES AGGREGAT-ELEMENTS IST KEINE STRUKTUR

SDP1066    VARIABLE '(&00)' CANNOT BE DECLARED WITH A CONTAINER
SDP1066    VARIABLE '(&00)' KANN NICHT MIT EINEM BEHAELTER DEKLARIERT WERDEN
```

Bedeutung
Statische Strukturen, die mit TYPE=STRUCT(*BY-SYSCMD) deklariert werden, duerfen nie einen Container haben.

```
SDP1067    STRUCTURE NOT DECLARED
SDP1067    STRUKTUR NICHT DEKLARIERT
```

Bedeutung
Moegliche Ursachen:
- Der Name des Layouts wurde bereits vergeben.
- Ein /BEGIN-STRUCTURE ohne Strukturdeklaration mit TYPE=STRUCT(*BY-SYSCMD) wurde angegeben.
- Ein /BEGIN-STRUCTURE-Kommando ist zuviel.

```
SDP1068    LAYOUT DECLARATION NOT POSSIBLE
SDP1068    LAYOUT-DEKLARATION NICHT MOEGLICH
```

Bedeutung
Eine Struktur oder ein Layout ist noch offen.
Es wird ein /BEGIN-STRUCTURE-Kommando ohne Operanden erwartet, welches die Elementdeklarationen der mit DEF=*BY-SYSCMD deklarierten Struktur einleitet.

Maßnahme
Struktur oder Layout mit /END-STRUCTURE schliessen.

SDP1069 STRUCTURE NOT CLOSED
SDP1069 STRUKTUR NICHT GESCHLOSSEN

Bedeutung
Moegliche Ursachen:
Ein /END-STRUCTURE-Kommando fehlt.
Die Zuweisung fuer das Strukturelement wurde ignoriert, da die Struktur noch nicht geschlossen ist.

Maßnahme
Struktur mit der noetigen Anzahl /END-STRUCTURE-Kommandos schliessen und Zuweisung wiederholen.

SDP1070 STRUCTURE NAME UNKNOWN
SDP1070 STRUKTUR-NAME NICHT BEKANNT

Bedeutung
Im /END-STRUCTURE-Kommando wurde ein falscher Name angegeben.

SDP1071 FURTHER STRUCTURE DECLARATION NOT POSSIBLE
SDP1071 WEITERE STRUKTUR-DEKLARATION NICHT MOEGLICH

Bedeutung
Eine Struktur oder ein Layout mit TYPE=STRUCT(*BY-SYSCMD)) wurde noch nicht geschlossen. Eine neue Strukturdeklaration ist deshalb nicht moeglich.

SDP1072 OPERAND 'WRITE-MODE=(&OO)' INVALID IN /SET-VARIABLE COMMAND
SDP1072 OPERAND 'WRITE-MODE=(&OO)' IM /SET-VARIABLE-KOMMANDO FEHLERHAFT

Bedeutung
WRITE-MODE=EXTEND und WRITE-MODE=PREFIX ist nur fuer Listen zugelassen.
WRITE-MODE=MERGE ist nur fuer Strukturen zugelassen.

SDP1073 THE TYPES OF TARGET AND SOURCE OF THE /SET-VARIABLE COMMAND DO NOT MATCH
SDP1073 DIE VARIABLEN-TYPEN VON ZIEL UND QUELLE DES /SET-VARIABLE-KOMMANDOS PASSEN NICHT ZUSAMMEN

SDP1074 /DECLARE PARAMETER ONLY ALLOWED IN PROCEDURE HEAD
SDP1074 /DECLARE-PARAMETER NUR IM PROZEDURKOPF ERLAUBT

SDP1075 WARNING: /SHOW-VARIABLE FOR OUTPUT LIST IGNORED
SDP1075 WARNUNG: /SHOW-VARIABLE AUF AUSGABE-LISTE IGNORIERT

Bedeutung
Die Daten der Ausgabeliste werden nicht in dieselbe ausgegeben.
Zum Beispiel ist /SHOW-VARIABLE A,OUTPUT=*LIST(A) nicht moeglich.

```
SDP1076    INVALID VALUE OF OPERAND 'LIMIT'
SDP1076    UNGUELTIGER WERT VON OPERAND 'LIMIT'
```

Bedeutung
Limit-Angabe fuer Liste ist Null bzw. Negativ.
Der Operand UPPER-BOUND einer Arraydeklaration ist kleiner als LOWER-BOUND.

```
SDP1077    INPUT LIST EQUAL TO OUTPUT LIST IS NOT ALLOWED. COMMAND REJECTED
SDP1077    DIESELBE LISTE FUER INPUT UND OUTPUT IST NICHT ERLAUBT. KOMMANDO ABGEWIESEN

SDP1078    NO ASSIGNMENT ALLOWED FOR THIS VARIABLE
SDP1078    DIESER VARIABLEN KANN KEIN WERT ZUGEWIESEN WERDEN
```

Bedeutung
READ-VAR: Die Variable ist ein Array oder eine Liste.

```
SDP1079    CONVERSION ERROR WHILE READING THE INPUT
SDP1079    KONVERTIERUNGS-FEHLER BEIM EINLESEN DER EINGABEN
```

Maßnahme
Die Typen von Wert und Variablen, in die eingelesen werden soll, stimmen nicht ueberein.

```
SDP1080    INVALID INPUT FORMAT ENTERED FOR OPERAND '*BY-INPUT' IN /READ-VARIABLE COMMAND
SDP1080    FALSCHES EINGABE-FORMAT, WENN OPERAND '*BY-INPUT' IM /READ-VARIABLE-KOMMANDO
           ANGEGEBEN

SDP1081    MISSING INPUT RECORD
SDP1081    EINGABE-SATZ FEHLT
```

Bedeutung
In der Eingabedatei oder -liste des /READ-VARIABLE-Kommandos sind nicht genuegend Werte angegeben.

```
SDP1082    FILE '(&OO)' NOT AN ISAM FILE
SDP1082    DATEI '(&OO)' KEINE ISAM-DATEI

SDP1083    LIST OF AGGREGATES NOT ALLOWED
SDP1083    LISTE VON AGGREGATEN NICHT ERLAUBT

SDP1084    WARNING: INITIALIZATION OF LAYOUT ELEMENTS IGNORED
SDP1084    WARNUNG: INITIALISIERUNG VON LAYOUT-ELEMENTEN IGNORIERT

SDP1085    NO SDF-P COMMANDS PERMITTED IN NOT STRUCTURED PROCEDURES
SDP1085    KEINE SDF-P-KOMMANDOS IN NICHT STRUKTURIERTEN PROZEDUREN ZUGELASSEN

SDP1086    SCOPE OF LAYOUT SMALLER THAN SCOPE OF VARIABLE '(&OO)'
SDP1086    SCOPE DES LAYOUTS KLEINER ALS SCOPE DER VARIABLEN '(&OO)'

SDP1087    VARIABLE '(&OO)' ON THE LEFT SIDE OF THE ASSIGNMENT MUST BE AN AGGREGATE
SDP1087    VARIABLE '(&OO)' DER LINKEN SEITE DER ZUWEISUNG MUSS AGGREGAT SEIN
```

SDP1088 ELEMENT NAME '(&OO)' IN INVALID CONTEXT
SDP1088 ELEMENT-NAME '(&OO)' IN FALSCHEM KONTEXT

Bedeutung
/DECLARE-ELEMENT vom Datentyp 'composed name' ist nur fuer dynamische Strukturen moeglich.

SDP1089 OPERAND '*PROMPT' OF /DECLARE-PARAMETER IN COMBINATION WITH 'TRANSFER-TYPE=BY-REFERENCE' NOT ALLOWED
SDP1089 OPERAND '*PROMPT' DES /DECLARE-PARAMETER IN KOMBINATION MIT 'TRANSFER-TYPE=BY-REFERENCE' NICHT ERLAUBT

SDP1090 JOB VARIABLES NOT POSSIBLE AS CONTAINERS
SDP1090 JOBVARIABLEN ALS BEHAELTER NICHT MOEGLICH

SDP1091 ADDITION OF ELEMENTS TO STATIC STRUCTURES NOT POSSIBLE
SDP1091 EINER STATISCHEN STRUKTUR DARF KEIN ELEMENT MEHR HINZUGEFUEGT WERDEN

Bedeutung
Die Struktur darf nicht vergroessert werden:
Es ist kein /DECLARE-ELEMENT mehr moeglich (weder implizit noch explizit).

SDP1092 MULTIPLE DECLARATION OF STRUCTURE ELEMENTS NOT POSSIBLE
SDP1092 MEHRFACHDEKLARATION VON STRUKTURELEMENTEN NICHT MOEGLICH

SDP1093 OK, VARIABLE EXISTS, STRUCTURE IS ALREADY CLOSED
SDP1093 OK, VARIABLE EXISTIERT, DIE STRUKTUR IST ABER BEREITS GESCHLOSSEN

SDP1094 THE CREATED ELEMENT NAME IS TOO LONG (> 255)
SDP1094 DER ERZEUGTE ELEMENT-NAME IST ZU LANG (> 255 ZEICHEN)

Bedeutung
/SET-VAR <name> = <expr>,MERGE erzeugt Element-Namen, die zu lang sind.
Eine rekursive Zuweisung kann einen Element-Namen erzeugen, der zu lang ist.

SDP1095 WARNING: STRUCTURE IS EMPTY
SDP1095 WARNUNG: STRUKTUR IST LEER

SDP1096 VARIABLE '(&OO)' MUST BE A LIST OF TYPE STRING OR ANY CONTAINING ONLY STRING VALUES
SDP1096 VARIABLE '(&OO)' MUSS EINE LISTE VOM TYP STRING ODER ANY SEIN UND NUR STRING WERTE ENTHALTEN

SDP1097 ERROR IN PROMPT OPERAND
SDP1097 FEHLER IM PROMPT-OPERANDEN

Bedeutung
Der Prompt-Wert ist nicht vom Typ String oder zu lang.

```
SDP1098   /DELETE-VARIABLE not allowed for the variable '(&00)'
SDP1098   Auf Variable '(&00)' darf kein /DELETE-VARIABLE angewendet werden
```

Bedeutung
/DELETE-VARIABLE nicht erlaubt fuer :
- SYSSWITCH Variable
- Prozedurparameter
- Element einer zusammengesetzten Variable.

```
SDP1099   Cannot free list element '(&00)'
SDP1099   Listenelement '(&00)' kann nicht geloescht werden
```

Bedeutung
Nur das erste und das letzte Element einer Liste koennen mit dem Kommando FREE-VARIABLE geloescht werden.

```
SDP1100   Creation of gaps not allowed for a list in the variable '(&00)'
SDP1100   Liste in Variable '(&00)' darf keine Luecken enthalten
```

Bedeutung
Neue Listenelemente duerfen nur am Listenende angelegt werden; dabei duerfen keine Luecken entstehen, d.h. wenn die Liste 'n' Elemente enthaelt, darf nur ein Element 'n+l' angelegt werden.

Maßnahme
Mit dem Kommando /SET-VARIABLE WRITE-MODE=*PREFIX koennen neue Listenelemente vor dem Listenkopf eingefuegt werden.

```
SDP1101   SYNTAX ERROR IN VARIABLE NAME
SDP1101   SYNTAX-FEHLER IM VARIABLEN-NAMEN

SDP1102   Task variable '(&00)' has been deleted
SDP1102   Taskglobale Variable '(&00)' geloescht

SDP1103   Value of constant variable '(&00)' cannot be modified
SDP1103   Wert der konstanten Variablen '(&00)' kann nicht veraendert werden
```

Bedeutung
Der Wert der konstanten Variablen (&00) kann nicht mit den Kommandos SET-VARIABLE, READ-VARIABLE oder FREE-VARIABLE veraendert werden. Die Deklaration der Variablen kann mit DELETE-VARIABLE geloescht werden.

```
SDP1104   VARIABLE '(&00)' MUST BE A LIST OF ELEMENTS WITH SIMPLE VALUES
SDP1104   VARIABLES '(&00)' MUSS EINE LISTE DER ELEMENTE MIT EINFACHEN WERTEN SEIN

SDP1105   Constant variable already declared with another value
SDP1105   Konstante Variable schon mit einem anderen Wert deklariert
```

Bedeutung
Mehrfache Definitionen derselben konstanten Variablen sind erlaubt, wenn die Werte gleich sind.

```
SDP1106   Error while reading on SYSDTA
SDP1106   Fehler beim Lesen aus SYSDTA

SDP1107   VARIABLE '(&00)' CAN NOT BE SORTED. IT IS A LIST OF ELEMENTS WITH MIXED TYPE
          VALUE
SDP1107   VARIABLE '(&00)' KANN NICHT SORTIERT WERDEN. ES IST EINE LISTE DER ELEMENTE MIT
          MISCHART WERT

SDP1120   Variable '(&00)' must be a list of simple types or a list of structures
SDP1120   Variable '(&00)' muss eine Liste einfacher Typen oder eine Strukturliste sein
```

Bedeutung
Die Eingabevariable muss eine Liste einfacher Typen (string, integer oder boolean) oder eine Strukturliste sein.

```
SDP1121   Variable '(&00)' is not a list of structures
SDP1121   Variable '(&00)' ist keine Strukturliste
```

Bedeutung
Die Namen von Strukturelementen duerfen im Operanden DISPLAYED-ELEMENTS nicht angegeben werden,wenn die Eingabevariable keine Strukturliste ist.

```
SDP1122   Variable '(&00)' must be a list of dynamic structures
SDP1122   Variable '(&00)' mub eine dynamische Struckturliste sein
```

Bedeutung
Ein Element SELECTION-CODE wird in die Struktur eingefuegt. Das ist nur moeglich wenn TO-VARIABLE eine dynamische Struckturliste ist.

```
SDP1123   Value too long for variable '(&00)'
SDP1123   Werte fuer Variable '(&00)' zu lang
```

Bedeutung
Werte fuer StringVariablen sind nur bis 4096 Zeichen erlaubt.

```
SDP1124   Variable '(&00)' is not initialized
SDP1124   Variable '(&00)' ist nicht initialisiert

SDP1131   INDEX EVALUATION ERROR
SDP1131   FEHLER BEI INDEX-AUSWERTUNG

SDP1132   VARIABLE NAME '(&00)' OR SUBNAME TOO LONG
SDP1132   VARIABLEN-NAME '(&00)' ODER TEILNAME ZU LANG
```

Bedeutung
Variablennamen duerfen maximal 255 Zeichen lang sein.
Teilnamen von Variablen duerfen maximal 20 Zeichen lang sein.

```
SDP1133   INDEX OF VARIABLE '(&00)' OUT OF RANGE
SDP1133   INDEX-WERT DER VARIABLEN '(&00)' AUSSERHALB ZULAESSIGEN WERTEBEREICHS
```

Bedeutung
Es sind nur Werte zwischen -2**31 und 2**31-1 zugelassen.

```
SDP1134    VARIABLE NAME RESERVED FOR TPR APPLICATIONS
SDP1134    VARIABLEN-NAME FUER TPR-ANWENDUNGEN RESERVIERT

SDP1135    NO ARRAY ELEMENTS ALLOWED HERE
SDP1135    ARRAY-ELEMENTE HIER NICHT ERLAUBT

SDP1136    NO LIST ELEMENTS ALLOWED
SDP1136    KEINE LISTEN-ELEMENTE ERLAUBT

SDP1137    NO LIST DECLARED
SDP1137    KEINE LISTE DEKLARIERT

SDP1138    RESERVED KEYWORD USED
SDP1138    RESERVIERTES SCHLUESSELWORT VERWENDET
```

Bedeutung
Der Name darf werden als Variablenname noch als Funktionsname benutzt werden.
Folgende Namen sind reserviert:
- AND, LE, OFF, DIV, LT, ON, EQ, MOD, OR, FALSE, NE, TRUE, GE,
- NO, OR, GT, NOT, YES.

```
SDP1139    Parameter already declared
SDP1139    Parameter bereits deklariert
```

Bedeutung
Der Parameter darf nicht mehrfach deklariert werden.

```
SDP1140    Not a simple variable name '(&00)'
SDP1140    Kein einfacher Variablenname '(&00)'
```

Bedeutung
Der Variablenname (&00) darf kein '.' oder '#' enthalten.

```
SDP1141    Attributes of variable '(&00)' to be declared not allowed in SDF-P basic version
SDP1141    Attribute der Variablen '(&00)' duerfen im SDF-P-Grundausbau nicht deklariert
           werden
```

Bedeutung
Nur Variablen mit TYPE=*ANY und MULTIPLE-ELEMENTS=*NO koennen automatisch deklariert werden.

Maßnahme
Bitte Systemverwalter verstaendigen, um das Subsystem SDF-P zu starten.

```
SDP1201    Variable container '(&00)' already exists but with other attributes
SDP1201    Variablenbehaelter '(&00)' existiert bereits, aber mit anderen Attributen

SDP1202    WARNING: Variable Container '(&00)' already exists
SDP1202    Variablenbehaelter '(&00)' existiert bereits, Verarbeitung wird fortgesetzt

SDP1203    Variable Container '(&00)' does not exist
SDP1203    Variablenbehaelter '(&00)' existiert nicht
```

SDP1204 Contained variable '(&00)' cannot be processed by this version of SDF-P
SDP1204 Mit Behaelter verknuepfte Variable '(&00)' kann mit dieser SDF-P-Version nicht bearbeitet werden

SDP1205 Variable Container '(&00)' cannot be processed by this version of SDF-P
SDP1205 Variablenbehaelter '(&00)' kann mit dieser SDF-P-Version nicht bearbeitet werden

SDP1206 Variable container '(&00)' is corrupted
SDP1206 Variablenbehaelter '(&00)' verfaelscht

SDP1207 Contained variable '(&00)' does not match the specified structure
SDP1207 Mit Behaelter verknuepfte Variable '(&00)' passt nicht zur angegebenen Struktur

SDP1208 Version *INCREMENT not allowed for Variable Container '(&00)'
SDP1208 Version *INCREMENT nicht erlaubt fuer Variablenbehaelter '(&00)'

Bedeutung
Wenn der Variablenbehaelter mit LOCK-ELEMENT=*YES geoeffnet wurde, darf nicht Version *INCREMENT angegeben werden.

SDP1209 Value of SCOPE parameter not allowed for OPEN-VARIABLE-CONTAINER in procedure head
SDP1209 Unzulaessiger SCOPE-Parameterwert for OPEN-VARIABLE-CONTAINER im Prozedurkopf

SDP1210 Variable Container has been closed. Information not available
SDP1210 Variablenbehaelter wurde geschlossen. Informationen nicht verfuegbar

SDP1211 Variable '(&00)' already declared but refer to a closed or a no more visible variable-container
SDP1211 Variable '(&00)' existiert bereits aber bezieht auf einen geschlossenen oder auf einen nicht mehr sichtbaren Variablenbehaelter

Maßnahme
Die Deklaration der Variable kann mit /DELETE-VARIABLE geloescht werden.

SDP1300 Procedure compiler version '(&00)' started
SDP1300 Prozedur-Compiler Version '(&00)' gestartet

SDP1301 Procedure compiler terminated normally
SDP1301 Prozedur-Compiler normal beendet

SDP1302 Procedure compiler terminated abnormally
SDP1302 Prozedur Uebersetzer abnormal beendet

SDP1303 Command '(&00)' cannot be processed by the current SDF-P version
SDP1303 Kommando '(&00)' kann nicht durch die laufende SDF-P Version bearbeitet werden

Bedeutung
Das Kommando (&00) kommt von einer Prozedur, die durch eine hoehere SDF-P-Version uebersetzt wurde. Es kann nicht durch die laufende SDF-P-Version ausgefuehrt werden.

Maßnahme
Die neueste SDF-P-Version soll verwendet werden.

SDP1304 Compiled procedure corrupted at command '(&00)'
SDP1304 Kompilierte Prozedur verfaelscht ab Kommando '(&00)'

SDP1305 Error when writing result of compilation
SDP1305 Fehler beim Schreiben des Compilierungsergebnisses

SDP1306 Internal error in compiler component. Contact system administrator
SDP1306 Interner Fehler in der Compiler-Komponente. Systemverwalter verstaendigen

SDP1307 *SAME cannot be specified for incompatible FROM-FILE and TO-FILE operands
SDP1307 *SAME kann fuer die inkompatiblen Operanden FROM-FILE und TO-FILE nicht angegeben werden

SDP1308 Not structured procedures cannot be compiled
SDP1308 Nicht-strukturierte Prozeduren koennen nicht Kompiliert werden

SDP1401 Compiled procedure '(&00)' corrupted
SDP1401 Kompilierte Prozedur '(&00)' fehlerhaft

SDP1402 Procedure '(&00)' is not a S-procedure. Only elements of type 'J' supported
SDP1402 Prozedur '(&00)' ist keine S-prozedur. Nur ElementenTyp 'J' ist erlaubt

SDP1403 Procedure '(&00)' cannot be processed by this version of SDF-P
SDP1403 Prozedur '(&00)' kann mit dieser SDF-P-Version nicht bearbeitet werden

SDP1404 Invalid operand TYPE specified. Only '*STD' is processed by this version of SDF-P
SDP1404 Ungueltiger Operand TYPE angegeben. Nur '*STD' kann mit dieser SDF-P-Version bearbeitet werden

SDP1405 This procedure cannot be processed by this version of SDF-P
SDP1405 Diese Prozedur kann mit dieser SDF-P-Version nicht bearbeitet werden

Bedeutung
Element mit angegebenen Namen vom Typ SYSJ existiert in Bibliothek. Diese Funktion wird nur ab SDF-P-BASYS V02.0B unterstuetzt.

SDP1406 Library element '(&00)' not found
SDP1406 Bibliothekselement '(&00)' nicht gefunden

SDP2000 Warning: Not all list elements could be treated successfully
SDP2000 Warnung: Nicht alle Listenelemente konnten erfolgreich behandelt werden

SDP2001 None of the list elements could be treated successfully
SDP2001 Keines der Listenelemente konnte erfolgreich behandelt werden

SDP2002 Error when treating list element '(&00)'. Processing continues
SDP2002 Fehler bei Behandlung des Listenelements '(&00)'. Die Verarbeitung wird fortgesetzt

SDP2003 Error when treating list element '(&00)'. Processing aborted
SDP2003 Fehler bei Behandlung des Listenelements '(&00)'. Die Verarbeitung wird abgebrochen

Fachwörter

Im folgenden werden zentrale Begriffe des Prozedurkonzepts von SDF-P erläutert. Innerhalb der Definitiionstexte sind die Begriffe kursiv dargestellt, die an anderer Stelle in diesem Verzeichnis definiert werden.

aufgerufene Prozedur

Prozedur, die aus einer anderen Prozedur heraus aufgerufen wird und dieser untergeordnet ist. Die aufgerufene Prozedur wird vollständig abgearbeitet, bevor sie die Steuerung an die *aufrufende Prozedur* zurückgibt.

aufrufende Prozedur

Prozedur, die eine untergeordnete Prozedur aufruft. Die aufrufende Prozedur übergibt die Steuerung an die untergeordnete Prozedur, während diese abgearbeitet wird.

ELSE-Zweig

Bestandteil eines *IF-Blocks*, der eine alternative Kommandofolge enthält, die ausgeführt wird, wenn die Bedingung im IF-Kommando nicht erfüllt wurde. Ein ELSE-Zweig wird vom Kommando ELSE eingeleitet und ausgeführt.

Fehlerbehandlung

Basiert in *S-Prozeduren* darauf, dass BS2000-Kommandos einen definierten Returncode liefern. Anhand dieses Returncodes kann SDF-P feststellen, ob bei der Kommandoausführung ein Fehler auftrat. Die Fehlerbehandlung wird automatisch angestoßen, wenn eine Kommandoausführung einen Returncode mit einer Fehleranzeige zurückliefert. Die Fehlerbehandlung selbst wird in sogenannten Fehlerbehandlungsblöcken durchgeführt. Zwei solcher Fehlerbehandlungsblöcke sind zu unterscheiden: IF-BLOCK-ERROR und IF-CMD-ERROR. Bei einem Fehler im *Prozedurablauf* springt SDF-P zum nächsten IF-BLOCK-ERROR- oder IF-CMD-ERROR-Kommando.

FOR-Block

Schleife in *S-Prozeduren*, in der eine Kommandofolge so lange ausgeführt wird, wie einer Laufvariablen Werte zugewiesen werden und wahlweise eine bestimmte Bedingung erfüllt ist. Der FOR-Block beginnt mit dem Kommando FOR und endet mit dem Kommando END-FOR. Das Kommando FOR enthält die Laufvariable.

formaler Prozedurparameter

Variable in einer Prozedur, an die ein im Prozeduraufruf angegebener aktueller Parameter übergeben wird. Formale Prozedurparameter werden im Prozedurkopf deklariert.

Hintergrund-Prozedur

Prozedur, die im Hintergrund unabhängig vom aufrufenden Auftrag abläuft und eine eigene Auftragsnummer (TSN) erhält. Sowohl S- als auch Nicht-S-Prozeduren können als Hintergrund-Prozeduren ablaufen.

IF-Block

Bedingte *Verzweigung* in *S-Prozeduren*. Ein IF-Block wird mit dem Kommando IF eingeleitet und mit dem Kommando END-IF abgeschlossen. Die Bedingung für die Verzweigung wird im Kommando IF angegeben, eine alternative Bedingung im Kommando ELSE-IF. Ein IF-Block besteht immer aus einem *THEN-Zweig*, der ausgeführt wird, wenn die Bedingung erfüllt ist, und wahlweise einem *ELSE-Zweig*, der ausgeführt wird, wenn die Bedingung nicht erfüllt ist.

Kommandoblock

Bestandteil einer *S-Prozedur*, der zusammengehörende Prozedurteile zu einer logischen Einheit zusammenfasst. Ein Kommandoblock beginnt mit einem Einleitungskommando und endet mit einem Abschlusskommando. Dazwischen stehen die Kommandos, die in diesem Block ausgeführt werden sollen. Als Kommandoblock werden bezeichnet: *Schleifen*, *Verzweigungen* und einfache Kommandoblöcke.

Kontrollflusskommando

Kommandos in *S-Prozeduren*, die den *Prozedurablauf* steuern. Als Kontrollflusskommandos werden Sprungkommandos bezeichnet sowie paarweise zusammengehörende Einleitungs- und Abschlusskommandos, die Kommandoblöcke einleiten und abschließen. Kontrollflusskommandos in SDF-P sind: BEGIN-BLOCK, BEGIN-PARAMETER-DECLARATION, CYCLE, ELSE, ELSE-IF, END-BLOCK, END-FOR, END-IF, END-WHILE, END-PARAMETER-DECLARATION, EXIT-BLOCK, FOR, GOTO, IF, IF-BLOCK-ERROR, IF-CMD-ERROR, REPEAT, UNTIL, WHILE.

Kontrollstruktur

Bestandteil von *S-Prozeduren* zur Steuerung des *Prozedurablaufs*. Zu den Kontrollstrukturen zählen *Schleifen*, *Verzweigungen* und einfache *Kommandoblöcke*. Sie werden jeweils durch paarweise zusammengehörende Kontrollflusskommandos eingeleitet und abgeschlossen. Zu den *Kontrollflusskommandos* gehören außerdem die Sprungkommandos.

Nicht-S-Prozedur

*Prozedu*r im BS2000, die nicht nach SDF-P-Regeln erstellt ist. Nicht-S-Prozeduren können als *Vordergrund*- und als *Hintergrund-Prozeduren* ablaufen.

Prozedur

Häufig benutzte Folgen von Kommandos, Anweisungen und Datensätzen, die in einem *Prozedurbehälter* gespeichert werden. Die Ausführung dieser Kommandofolgen kann mit einem einzigen Kommando im Dialog- bzw. im Stapelbetrieb veranlasst werden.

Prozedurabschlusskommando

Kommando, mit dem die *Prozedurausführung* an jeder beliebigen Stelle beendet werden kann. Über das Prozedurabschlusskommando können Fehlerinformationen an den Aufrufer der Prozedur zurückgeliefert werden.

Prozeduraufruf

Einleitung des *Prozedurstarts*, bei der die zu startende Prozedur benannt wird und ggf. *Prozedurparamete*r übergeben werden. Beim Aufruf von *S-Prozeduren* werden zunächst die Kommandos des *Prozedurkopfs* ausgeführt und so die *Prozedureigenschaften* eingestellt, bevor die Kommandos des *Prozedurrumpfs* analysiert und verarbeitet werden.

Prozedurausführung

Ablaufteil des *Prozedurstarts*, in dem die Kommandos entsprechend den BS2000-Regeln ausgeführt werden. Dabei werden zunächst die Ausdrücke innerhalb der Kommandos ersetzt. Danach wird das Kommando analysiert und ausgeführt.

Prozedurbeendigung

Prozeduren können mit folgenden *Prozedurabschlusskommandos* beendet werden: mit dem SDF-P-Kommando EXIT-PROCEDURE, mit dem Kommando END-PROCEDURE, mit dem Kommando CANCEL-PROCEDURE.

Prozedurbehälter

Datei, Bibliothekselement oder Listenvariable, in der eine *Prozedur* gespeichert ist.

Prozedureigenschaften

Werden im Kommando SET-PROCEDURE-OPTIONS festgelegt und beeinflussen Aufruf, Ablauf und *Fehlerbehandlung* der *Prozedur*. Zu den Prozedureigenschaften gehören: Kommando für den Prozeduraufruf, Systemdatei-Umgebung, Länge der Prozedursätze, Art der Protokollierung, Unterbrechbarkeit der Prozedur, Art der Fehlerbehandlung, Escape-Zeichen, Art der Variablendeklaration, Jobvariablenersetzung. Prozedureigenschaften können im Kommando MODIFY-PROCEDURE-OPTIONS geändert werden.

Prozedur-Interpreter

Prüft den *Prozedurkopf*, führt ihn aus und analysiert dann den *Prozedurrumpf*. Bei der Ausführung der Prozedur erkennt der Prozedur-Interpreter die *Kontrollflusskommandos* und führt sie aus.

Prozedurkopf

Bestandteil einer *S-Prozedur*, in dem die globalen *Prozedureigenschaften* definiert und die *Prozedurparameter* deklariert werden. Der Prozedurkopf steht am Anfang einer S-Prozedur.

Prozedurparameter

In SDF-P Oberbegriff für aktuelle Parameter und *formale Parameter*. Wird als Synonym für aktuelle und formale Parameter verwendet, wenn deren Unterscheidung nicht relevant ist. Prozedurparameter sind durch folgende Eigenschaften gekennzeichnet: Parametername, Anfangswert (falls angegeben), Datentyp und Art der Parameterübergabe.

Prozedurrumpf

Bestandteil einer *S-Prozedur*, der den Ablauf der Prozedur festlegt. Der Prozedurrumpf folgt auf den P*rozedurkopf* . Er besteht aus einer Folge von Kommandos, Anweisungen und Daten. Die S-Prozedur kann gesteuert werden über die verschiedenen Arten von *Kommandoblöcken*, die SDF-P unterstützt, zum Beispiel über *Verzweigungen*, *Schleifen* oder Fehlerbehandlungsblöcke.

Prozedursatz

Verarbeitungseinheit einer *Prozedur* bestehend aus Kommandos, Anweisungen oder Daten, die während des Ablaufs auf einmal vom System verarbeitet werden. Mehrere Prozedursätze, die aus Kommandos oder Anweisungen bestehen, können durch Semikolon getrennt in eine *Prozedurzeile* geschrieben werden.

Prozedurschachtelung

Bezeichnet den Aufruf einer Prozedur aus einer anderen Prozedur heraus, wobei jeder *Prozedurbehälter* nur eine *Prozedur* enthalten darf.

Prozedurstart

Umfasst Aufruf, Analyse und Ausführung der *Prozedur*.

Prozedurumgebung

Besteht aus allen Systemdaten und Systemeigenschaften, die den Prozedurablauf beeinflussen, z.B. Taskvariablen, *Prozedureigenschaften*, Variablen der *aufrufenden Prozedur*.

Prozedurunterbrechung

Unterbrechung des Prozedurlaufs einer *Prozedur*, die im Dialog als Vordergrund-Prozedur aufgerufen wurde. Zu unterscheiden sind Unterbrechungen von der Systemebene und prozedurinterne Unterbrechungen. Prozedurintern kann der Prozedurlauf jederzeit mit dem Kommando HOLD-PROCEDURE unterbrochen werden. Von der Systemebene aus kann eine Prozedur mit der Funktionstaste K2 unterbrochen werden.

Prozedurzeile

Datensatz des *Prozedurbehälters*, der die Daten, Kommandos und Anweisungen der Prozedur enthält. Kommando- und Anweisungsfolgen können sich über mehrere Prozedurzeilen erstrecken.

REPEAT-Block

Schleife in *S-Prozeduren*, in der eine Kommandofolge so lange ausgeführt wird, bis eine definierte Bedingung nicht mehr erfüllt ist. Der REPEAT-Block wird mindestens einmal durchlaufen. Er beginnt mit dem Kommando REPEAT und endet mit dem Kommando UNTIL.

Schleife

Kontrollstruktur in *S-Prozeduren*, die in Abhängigkeit einer Bedingung mehrmals durchlaufen werden kann. Eine Schleife wird von einem Einleitungskommando eingeleitet und mit einem Abschlusskommando beendet.

SDF-P

Prozedursprache, die die Kommandosprache des BS2000 zu einer Programmiersprache erweitert, in der strukturiertes Programmieren analog zu höheren Programmiersprachen möglich ist. Ein wesentliches Merkmal von Prozeduren unter SDF-P ist der Aufbau durch sogenannte *Kommandoblöcke*.

Sprachelemente in S-Prozeduren

Sind Kommandos, Anweisungen, Datensätze, Variablen, Funktionen und Ausdrücke.

S-Prozedur

Strukturierte Prozedur im BS2000, die dem Prozedurformat von SDF-P entspricht. Hauptbestandteile einer S-Prozedur sind der *Prozedurkopf* und der *Prozedurrumpf*. Zusammengehörende Prozedurteile können zu *Kommandoblöcken* zusammengefasst werden. Die *Fehlerbehandlung* in S-Prozeduren ist blockorientiert.

THEN-Zweig

Bestandteil eines *IF-Blocks*, der ausgeführt wird, wenn die Bedingung im IF-Kommando erfüllt ist. Der THEN-Zweig wird vom Kommando IF eingeleitet und ausgeführt.

Verzweigung

Kontrollstruktur in *S-Prozeduren*, in der Kommandofolgen abhängig vom Ergebnis einer Bedingung ausgeführt werden. Eine Verzweigung wird von einem Einleitungskommando eingeleitet und mit einem Abschlusskommando beendet. Verzweigungen werden auch als *IF-Blöcke* bezeichnet.

Vordergrund-Prozedur

Prozedur, die unter Steuerung des Auftrags abläuft, in dem sie aufgerufen wurde; es wird kein neuer Auftrag erzeugt.

WHILE-Block

Schleife in *S-Prozeduren*, in der eine Kommandofolge so lange ausgeführt wird, bis eine definierte Bedingung nicht mehr erfüllt ist. Ist die Bedingung bei der ersten Ausführung bereits nicht erfüllt, wird die Kommmandofolge nie durchlaufen. Der WHILE-Block beginnt mit dem Kommando WHILE und endet mit dem Kommando END-WHILE.

Literatur

Die Handbücher sind online unter *http://manuals.fujitsu-siemens.com* zu finden oder in gedruckter Form gegen gesondertes Entgelt unter *http://FSC-manualshop.com* zu bestellen.

[1] **BS2000/OSD-BC**
Einführung in das DVS
Benutzerhandbuch

[2] **BS2000/OSD-BC**
DVS-Makros
Benutzerhandbuch

[3] **BS2000/OSD-BC**
Kommandos Band 1 - 5
Benutzerhandbuch

[4] **BS2000/OSD-BC**
Kommandos Band 6, Ausgabe in S-Variablen und SDF-P-BASYS
Benutzerhandbuch

[5] **JV** (BS2000/OSD)
Jobvariablen
Benutzerhandbuch

[6] **AID V2.1A** (BS2000/OSD)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

[7] **BS2000/OSD-BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

[8] **BS2000/OSD-BC**
Einführung in die Systembetreuung
Benutzerhandbuch

[9] **HIPLEX MSCF** (BS2000-OSD)
Mehrrechnersystem
Benutzerhandbuch

[10] **SDF** (BS2000/OSD)
SDF-Verwaltung
Benutzerhandbuch

[11] **LMS** (BS2000/OSD)
SDF-Format
Benutzerhandbuch

[12] **IMON** (BS2000/OSD)
Installationsmonitor
Benutzerhandbuch

[13] **SECOS** (BS2000/OSD)
Security Control System
Benutzerhandbuch

[14] **BS2000/OSD-BC**
Systeminstallation
Benutzerhandbuch

[15] **BS2000/OSD-BC**
Systemmeldungen
Benutzerhandbuch

[16] **SDF-A** (BS2000/OSD)
Benutzerhandbuch

[17] **SDF-CONV** (BS2000/OSD)
Benutzerhandbuch

[18] **POSIX** (BS2000/OSD)
Kommandos
Benutzerhandbuch

[19] **FHS** (BS2000/OSD, TRANSDATA)
Dialogerweiterung für TIAM und SDF-P
Benutzerhandbuch

[20] **SDF** (BS2000/OSD)
Einführung in die Dialogschnittstelle SDF
Benutzerhandbuch

[21] **MSGMAKER** (BS2000/OSD)
Erstellen und Bearbeiten von BS2000 Meldungsdateien
Benutzerhandbuch

[22] **DSSM / SSCM** (BS2000/OSD)
Verwaltung von Subsystemen
Benutzerhandbuch

[23] **CALENDAR** (BS2000/OSD)
Benutzerhandbuch

[24] **BS2000/OSD-BC**
System-Exits
Benutzerhandbuch

[25] **BS2000/OSD**
Softbooks Deutsch
CD-ROM

Internet-Adresse
http://manuals.fujitsu-siemens.com

Stichwörter

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.