1 Einleitung

SDF-A (System Dialog Facility-Administration) ist ein Softwareprodukt zur Verwaltung und Anpassung der Benutzeroberfläche des BS2000, die durch SDF realisiert ist.
SDF (System Dialog Facility) benötigt zur internen Verarbeitung von Eingaben Syntaxbeschreibungen, die in eigenen Dateien, den Syntaxdateien, abgelegt werden. Diese Syntaxdateien können in einer maximal dreistufigen Hierarchie angeordnet werden (siehe Abschnitt „Dateihierarchie“ auf Seite 33).

Mit der Vorgabe der Benutzeroberfläche wird der Leistungsumfang des Systems festgelegt. Durch Anpassung der Benutzeroberfläche können daher z.B. Benutzer gezielt privilegiert oder in ihren Rechten und Befugnissen eingeschränkt werden. Mit SDF-A kann die Benutzeroberfläche dementsprechend bearbeitet werden.

Das vorliegende Handbuch beschreibt, wie Sie mit SDF-A neue Syntaxdateien erstellen und vorhandene Syntaxdateien ändern können.

Die mit SDF-A vorgenommene Bearbeitung einer Syntaxdatei wird wirksam, sobald diese Syntaxdatei aktiviert wird.

1.1 Zielsetzung und Zielgruppen

Das Handbuch wendet sich vor allem an die Systembetreuung und an erfahrene BS2000-Benutzer. Es dient zum einen der Beschreibung der Leistung und Anwendung von SDF-A, zum andern dient es als Nachschlagewerk für alle SDF-A-Anweisungen, Makros und für die Schnittstelle zwischen SDF und höheren Programmiersprachen.

Das Handbuch enthält auch die Beschreibung von SDF-SIM (System Dialog Facility-Simulator). SDF-SIM ermöglicht das Testen von Kommandos und Anweisungen in einer definierten Umgebung von Syntaxdateien.

Als vorbereitende Lektüre und zur Vertiefung und Ergänzung des Themas SDF sind folgende Handbücher empfehlenswert:

– „Einführung in die Dialogschnittstelle SDF“ [1]
  Dieses Handbuch beschreibt die mit SDF realisierte Dialogschnittstelle, die Möglichkeiten für die Eingabe von SDF-Kommandos und Anweisungen sowie die direkt zu SDF gehörenden Kommandos und Standardanweisungen.

– „SDF-Verwaltung“ [2]
  Dieses Handbuch beschreibt die Kommandos zum Steuern von SDF, die Installation und Verwaltung von Syntaxdateien, sowie die Anweisungen an SDF-I, SDF-U und SDF-PAR.

– „Kommandos, Band 1 bis 5“ [4]
  Diese Handbücher beschreiben alle BS2000/OSD-Kommandos im SDF-Format, auch die Kommandos, die der Systembetreuung vorbehalten sind.

– „Einführung in die Systembetreuung“ [6]
  Dieses Handbuch beschreibt die Maßnahmen zur Systembedienung und Systemverwaltung.

Zu SDF-A gibt es ein Tabellenheft [3]. Es enthält die SDF-A-Anweisungen, die Makros und die Funktionsaufrufe der Schnittstelle zwischen SDF und höheren Programmiersprachen sowie die SDF-SIM-Anweisungen. Das Tabellenheft ist als Nachschlagewerk für erfahrene Benutzer gedacht.

1.2 Konzept des Handbuchs

Als Einführung in das Konzept von SDF und in die grundsätzliche Vorgehensweise beim Arbeiten mit SDF-A ist das Kapitel „Syntaxdateien und ihre Bearbeitung mit SDF-A“ gedacht.

Wie man mit SDF-A arbeitet und was SDF-A leistet, erfahren Sie anhand von Beispielen in den Kapiteln „Kommando- und Anweisungsdefinitionen ändern“ und „Eigene Kommandos und Anweisungen definieren und implementieren“.

Die Anweisungen an SDF-A, die SDF-Programmschnittstelle und SDF-SIM sind jeweils in eigenen Kapiteln und alphabetisch geordnet beschrieben. Dieser Teil des Handbuchs kann als Referenzteil verwendet werden.

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-A V4.1 also unter SYSRME.SDF-A.041.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 INSTALL-UNIT=SDF-A,LOGICAL-IDENTIFIER=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
```

1.3 Änderungen gegenüber der vorigen Version

Das Handbuch zu SDF-A V4.1E enthält gegenüber dem Vorgängerhandbuch (SDF-A V4.1A, Ausgabe Juni 1996) folgende Änderungen:

- Ab SDF V4.4A steht Ihnen die neue Standardanweisung SHOW-STMT zur Verfügung.
- Die Standardanweisungen werden nicht mehr in diesem Handbuch beschrieben. Sie finden eine ausführliche Beschreibung der Standardanweisungen im Benutzerhandbuch „Einführung in die Dialogschnittstelle SDF“ [1].
- Bei den folgenden Anweisungen gibt es Änderungen:

Anweisung	**Änderungen**
ADD-CMD, MODIFY-CMD	Struktur IMPLEMENTOR=*PROCEDURE(...): – in der Struktur NAME=*BY-IMON(...): neuer Operand ELEMENT – neuer Operand UNLOAD-PROGRAM In den Strukturen *YES(...) bzw. *NO(...) der Operanden DIALOG-, DIALOG-PROC-, GUIDED-, BATCH-, BATCH-PROC- und CMD-ALLOWED: – PRIVILEGE=*ALL entfällt (entspricht *SAME); wird aber noch kompatibel unterstützt
ADD-STMT, MODIFY-STMT	neuer Operand STMT-VERSION
DEFINE- ENVIRONMENT	Struktur PARAMETER-FILE=*NO(...): – neuer Operandenwert SYSTEM=*CURRENT

- Die Beschreibung der Schnittstelle zu höheren Programmiersprachen wurde überarbeitet. Insbesondere ist zu beachten, dass die Schnittstelle zu höheren Programmiersprachen nur den normierten Übergabebereich im alten Format (vgl. Abschnitt „Änderungen der SDF-Programmschnittstelle“ auf Seite 605ff) unterstützt, d.h. die Funktionalität der „alten“ Makroaufrufe RDSTMT, CORSTMT und TRSTMT. Die erweiterte Funktionalität der entsprechenden „neuen“ Makroaufrufe CMDRST, CMDCST und CMDTST (z.B. Statement-Returncode) für den normierten Übergabebereich im neuen Format ist nur über die Assembler-Schnittstelle verfügbar. Das Beispiel für die Nutzung der C-Schnittstelle wurde entsprechend aktualisiert.

SDF-A V4.1E ist ab BS2000/OSD V1.0 (mit SDF ab V4.1B) ablauffähig.
Bei voller Nutzung des Funktionsumfangs sind die erzeugten Syntaxdateien jedoch erst ab BS2000/OSD V3.0 einsetzbar.

1.4 Hinweise zur Benutzeroberfläche

Die in diesem Handbuch beschriebenen Anweisungen werden vom Kommandoprozessor SDF (System Dialog Facility) verarbeitet. Das Produkt bietet damit verschiedene Formen des geführten oder ungeführten Dialogs mit der Möglichkeit, Hilfsmenüs zu den Anweisungen und Kommandos anzufordern. Bei fehlerhaften Eingaben kann ein Korrekturdialog geführt werden. Ausführliche Informationen zu den Möglichkeiten, die SDF bietet, finden Sie im Handbuch „Einführung in die Dialogschnittstelle SDF“ [1].

Abkürzungsmöglichkeiten von Namen

SDF ermöglicht, Eingaben sowohl im Dialog- und Stapelbetrieb als auch in Prozeduren abzukürzen. Die Abkürzungen müssen in der zugehörigen Syntax-Umgebung eindeutig sein. Bei Funktionserweiterungen kann eine heute bestehende Eindeutigkeit wieder aufgehoben werden. Es wird deshalb empfohlen, in Prozeduren keine oder allenfalls die garantierten Abkürzungen zu verwenden, die in den Anweisungsformaten durch Fettdruck hervorgehoben sind. Kommando- und Anweisungsnamen, Operanden und Schlüsselwortwerte können wie folgt abgekürzt werden:

– von rechts nach links können ganze Namensteile weggelassen werden. Mit einem Namensteil entfällt auch der vorangehende Bindestrich.
– bei jedem Namensteil können von rechts nach links einzelne Zeichen weggelassen werden.
– ein Stern vor einem Schlüsselwortwert als Abkürzung für diesen Wert ist keine zulässige Abkürzung. Ab SDF V4.0A werden Schlüsselwortwerte immer mit führendem Stern dargestellt. Der Stern kann weggelassen werden, wenn alternativ kein variabler Operandenwert möglich ist, dessen Wertebereich den Namen des Schlüsselwortwerts beinhaltet. Diese Abkürzungsmöglichkeit kann durch Erweiterungen in einer Folgeversion eingeschränkt werden. Aus Kompatibilitätsgründen werden Operandenwerte, die bisher ohne führenden Stern dargestellt wurden, auch ohne Stern akzeptiert.

Beispiel für die Eingabe

Kommando in der Langform:

```
/MODIFY-SDF-OPTIONS SYNTAX-FILE=*NONE,GUIDANCE=*MINIMUM
```

Kommando in Kurzform:

```
/MOD-SDF-OPT SYN-F=*NO,GUID=*MIN
```

Die garantierten Abkürzungen sind als Vorschlag für eine verkürzte Eingabe zu verstehen, sie sind aber nicht unbedingt die in Ihrer Syntax-Umgebung kürzest möglichen. Sie sind jedoch aussagefähig und werden langfristig eindeutig gehalten.

Neben dem Kommando- oder Anweisungsnamen kann im Handbuch zusätzlich ein Kurzname dokumentiert sein. Der Kurzname ist als Aliasname des Kommandos oder der Anweisung implementiert und wird langfristig garantiert. Der Kurzname besteht aus maximal 8 Buchstaben (A...Z), die aus dem Kommando- oder Anweisungsnamen abgeleitet sind. Ein Kurzname kann nicht abgekürzt werden.

Default-Werte

Die Angabe der meisten Operanden ist wahlfrei. Wahlfreie Operanden sind bereits mit einem Operandenwert vorbesetzt, dem so genannten Default-Wert. Die Default-Werte sind in der Syntaxdarstellung durch Unterstreichung gekennzeichnet. Wird der wahlfreie Operand nicht explizit angegeben, so wird zur Ausführung des Kommandos oder der Anweisung für den Operanden der Default-Wert eingesetzt. Wenn einige Default-Werte nicht den Werten entsprechen, die Sie oft brauchen, dann können Sie ab SDF V4.1A für Ihre Task temporär eigene Default-Werte definieren.

Stellungsoperanden

SDF erlaubt die wahlweise Angabe von Operanden als Schlüsselwort- oder als Stellungsoperanden. Es kann jedoch nicht völlig ausgeschlossen werden, dass sich bei einem Versionswechsel eine Operandenposition ändert. Insbesondere in Prozeduren sollten deshalb Stellungsoperanden vermieden werden.

XHCS-Unterstützung

Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Bei Einsatz des Softwareproduktes XHCS können erweiterte Zeichensätze verwendet werden. Eingaben werden von SDF gemäß der Standard-Codetabelle **EBCDIC.DF.03** interpretiert (z.B. bei der Umsetzung Groß-/Kleinschreibung).
Die Codierung folgender Zeichen darf in einem erweiterten Zeichsatz nicht von der Codierung der Standard-Codetabelle abweichen:

$, # , @ , ! , ” , ? , ^ , = , : , / , * , - , (,) , [,] , < , > , Komma, Semikolon, Hochkomma

Zusätzliche Zeichen aus einem erweiterten Zeichensatz werden von SDF nur innerhalb der Datentypen <c-string> und <text> ausgewertet, d.h. zur Umsetzung von Groß-/Kleinschreibung wird eine von XHCS für den erweiterten Zeichensatz bereitgestellte Codetabelle verwendet. Zusätzliche Zeichen innerhalb der anderen Datentypen werden mit Syntaxfehler abgewiesen.

Bei der Eingabe von Anweisungen wird das CCS verwendet, das im jeweiligen Makroaufruf (RDSTMT/CMDRST, TRSTMT/CMDTST oder CORSTMT/CMDCST) angegeben wurde. Wurde kein CCS angegeben, wird das CCS wie bei der Eingabe von Kommandos verwendet.

Weitere Informationen zur XHCS-Unterstützung finden Sie in den Handbüchern „Einführung in die Dialogschnittstelle SDF“ [1] und „XHCS“ [11].

1.5 Verwendete Metasprache

1.5.1 Darstellungsmittel

In diesem Handbuch werden folgende Darstellungsmittel verwendet:

für Hinweise

für Warnhinweise

Ablaufbeispiele sind mit `dicktengleicher` Schrift dargestellt. Eingaben des Benutzers sind zusätzlich **`halbfett`** hervorgehoben.

Literaturhinweise werden im Text in Kurztiteln und eckigen Klammern [ ] angegeben. Alle folgenden Literaturhinweise beziehen sich auf BS2000/OSD-BC V5.0. Der vollständige Titel jeder Druckschrift, auf die verwiesen wird, ist im Literaturverzeichnis aufgeführt.

1.5.2 SDF-Syntaxdarstellung

Diese Syntaxbeschreibung basiert auf der SDF-Version 4.5A. 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 = ***YES** (...) / ***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 = ***YES** (...) ***YES**(...) **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 18).
	-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)

2 Syntaxdateien und ihre Bearbeitung mit SDF-A

Zusammen mit dem Grundausbau des BS2000 und mit den jeweiligen Softwareprodukten werden so genannte Syntaxdateien ausgeliefert. Diese enthalten:

- Informationen über die Syntax der jeweiligen Kommandos sowie der Programmanweisungen
- Informationen darüber, wie die Kommandos im BS2000 implementiert sind, z.B.
  - die Namen der System-Entries, über die die Kommandoausführung veranlasst wird
  - die Festlegungen für die Parameterübergaben an die ausführenden System-Module
- allgemeine und kommandospezifische Festlegungen für die Dialogführung
- erläuternde Texte zu den Kommandos bzw. Anweisungen und ihren Operanden (Hilfe-Texte)
- Privilegien, die für den Zugriff auf Anwendungsbereiche, Kommandos, Anweisungen, Operanden und Operandenwerte notwendig sind.

Die Syntaxdateien werden, je nach Bedarf, zu Gruppen- oder Systemsyntaxdateien gemischt und im System entsprechend zugewiesen (siehe Handbuch „SDF-Verwaltung“ [2]). Wenn SDF eine Kommandoeingabe verarbeitet, so holt es sich die dazu benötigten Informationen aus den aktivierten Syntaxdateien (siehe Abschnitt „Dateihierarchie“ auf Seite 33).

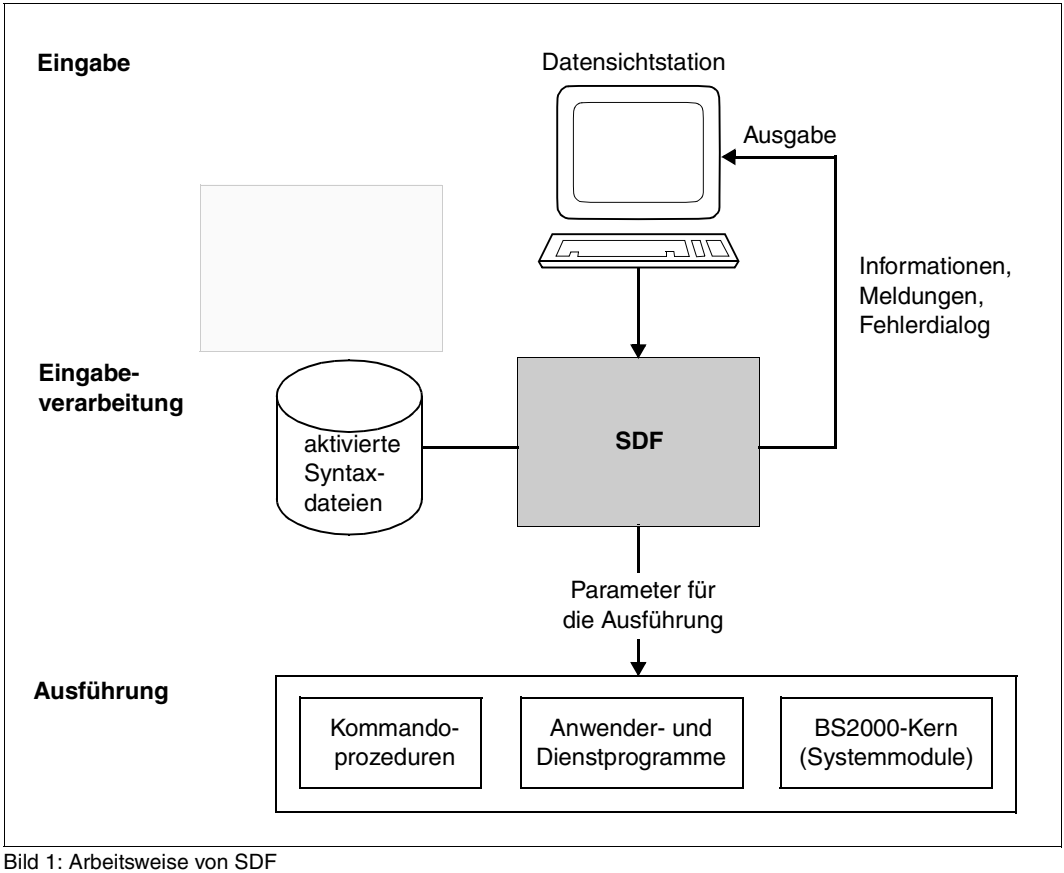


Bild 1: Arbeitsweise von SDF

2.1 Dateiarten

Es gibt drei Arten von Syntaxdateien:

– Systemsyntaxdateien

– Gruppensyntaxdateien

– Benutzersyntaxdateien

Neben den Systemsyntaxdateien können für eine Task eine Gruppen- und eine oder mehrere Benutzersyntaxdateien aktiviert sein. Für diesen Fall gibt es eine feste Dateihierarchie, die festlegt, wie SDF mit mehreren Syntaxdateien arbeitet (siehe Abschnitt „Dateihierarchie“ auf Seite 33ff).

SDF-A unterstützt Syntaxdateien mit und ohne PAM-Key.

2.1.1 Systemsyntaxdateien

Mit dem BS2000 Grundausbau und den jeweiligen Softwareprodukten werden Syntaxdateien vom Typ SYSTEM ausgeliefert. Diese enthalten die Definitionen von Kommandos und Programmen, die systemweit angeboten werden sollen. Sie werden von Fujitsu Siemens Computers geliefert und können mit SDF-A verändert werden.
Ab SDF V3.0 können mehrere Systemsyntaxdateien gleichzeitig aktiviert sein. Dabei muss mindestens die Basis-Systemsyntaxdatei vorhanden sein und es kann mehrere Subsystem-Syntaxdateien geben.

Basis-Systemsyntaxdatei

Um mit SDF arbeiten zu können, benötigen Sie zumindest die Basis-Systemsyntaxdatei. Sie enthält die Definitionen der Anwendungsbereiche, der Standardanweisungen und der SDF-spezifischen Kommandos (aus dem Anwendungsbereich SDF) so wie die systemweiten Einstellungen der Globalinformation. Standardmäßig wird hierfür die Syntaxdatei SYSSDF.SDF.045 aktiviert. Die Systembetreuung kann mit den Dienstprogrammen SDF-A (bzw. SDF-U) Modifizierungen (z.B. Funktionseinschränkungen) vornehmen. Modifizierungen gelten für alle Benutzer des Systems.

Beim Starten von SDF wird der Name der Basis-Systemsyntaxdatei aus der SDF-Parameterdatei gelesen und die Basis-Systemsyntaxdatei wird automatisch aktiviert.

Die Systembetreuung kann während des Systemlaufs mit dem Kommando MODIFY-SDF-PARAMETERS die aktivierte Basis-Systemsyntaxdatei durch eine andere ersetzen. Ein Wechsel der Basis-Systemsyntaxdatei während des Systemlaufs wirkt sofort für alle bestehenden und künftigen Tasks.

Subsystem-Syntaxdateien

Neben der Basis-Systemsyntaxdatei können mehrere Subsystem-Syntaxdateien aktiviert sein. Eine Subsystem-Syntaxdatei enthält die Definitionen von Kommandos und Programmen, die zu einem von DSSM verwalteten Subsystem oder auch zu einer beliebigen Installation-Unit (incl. BS2CP) gehören. Sie kann mit SDF-A modifiziert werden, wobei Modifikationen, wie z.B. Funktionseinschränkungen, für alle Benutzer des Systems gelten.

Die Aktivierung einer Subsystem-Syntaxdatei während des Systemlaufs wirkt sofort für alle bestehenden und künftigen Tasks.

Eine Subsystem-Syntaxdatei kann auf zwei Arten aktiviert werden:

- Der Name der Subsystem-Syntaxdatei wurde bei Deklaration des Subsystems festgelegt. Beim Laden des Subsystems wird die Subsystem-Syntaxdatei automatisch aktiviert bzw. beim Entladen wieder deaktiviert.
- Der Name der Subsystem-Syntaxdatei ist in der SDF-Parameterdatei eingetragen. Bei der Systemeinleitung wird sie automatisch aktiviert. Ist der Pubset, auf dem sie abgelegt ist, bei der Systemeinleitung nicht verfügbar, kann sie erst durch das Importieren des Pubsets aktiviert werden.
  Eine Änderung (Deaktivierung bzw. Austausch) kann nur über die SDF-Parameterdatei erfolgen. Die Bereitstellung der Subsystem-Syntaxdatei ist hier unabhängig von der Verfügbarkeit des zugehörigen Subsystems.

Ist eine Kommando- bzw. Anweisungsdefinition in mehreren aktivierten Subsystem-Syntaxdateien definiert, so sind folgende Fälle zu unterscheiden:

- Von dem Subsystem kann immer nur eine Version aktiviert sein. In diesem Fall wird die Kommando- bzw. Anweisungsdefinition benutzt, die in der Syntaxdatei der zuletzt aktivierten Version des Subsystems enthalten ist.
- Von dem Subsystem können mehrere Versionen aktiviert sein (Koexistenz). Dabei sind folgende Fälle zu unterscheiden:
  - Für Tasks, die einer bestimmten Subsystem-Version zugeordnet sind, wird die Kommando- bzw. Anweisungsdefinition aus der Syntaxdatei der entsprechenden Subsystem-Version benutzt.
    Die Syntaxanalyse eines START-<utility>-Kommandos erfolgt immer vor dem Laden des Programms, d.h. vor der Verbindung der Task mit einer Subsystem-Version des aufgerufenen Programms. Deshalb wird dafür die Syntaxdatei der zuerst aktivierten Version des Subsystems ausgewertet.
  - Für Tasks, die keiner Subsystem-Version zugeordnet sind, wird die Kommando- bzw. Anweisungsdefinition benutzt, die in der Syntaxdatei der zuerst aktivierten Version des Subsystems enthalten ist.

2.1.2 Gruppensyntaxdatei

Mit Softwareprodukten können auch Syntaxdateien vom Typ GROUP ausgeliefert werden. Diese enthalten die Definitionen von Kommandos und Programmen, die nur für bestimmte Benutzerkennungen angeboten werden sollen.

Diese Gruppensyntaxdateien können mit SDF-A verändert werden. Mit SDF-A kann sich die Systembetreuung Syntaxdateien vom Typ GROUP auch selbst erstellen. Mit SDF-I (siehe Handbuch „SDF-Verwaltung“ [2]) werden sie in die jeweilige Gruppensyntaxdatei eingemischt. Die Veränderungen, z.B. Funktionserweiterungen, gelten für die Benutzerkennungen, denen die jeweilige Gruppensyntaxdatei zugewiesen wird.

Eine Gruppensyntaxdatei kann Erweiterungen, Einschränkungen und sonstige Änderungen gegenüber den Systemsyntaxdateien enthalten. Die Systembetreuung hat durch die Zuweisung einer Gruppensyntaxdatei die Möglichkeit, den Funktionsvorrat gezielt auf bestimmte Benutzerkennungen abzustimmen.

Hat die Systembetreuung z.B. den von Fujitsu Siemens Computers angebotenen Funktionsumfang in der Basis-Systemsyntaxdatei systemweit eingeschränkt, kann sie über eine Gruppensyntaxdatei diese Einschränkung für bestimmte Benutzerkennungen aufheben. Andererseits kann sie systemweit angebotene Funktionen über eine Gruppensyntaxdatei für bestimmte Benutzerkennungen einschränken.

Eine Gruppensyntaxdatei kann, muss aber nicht aktiviert sein. Die Systembetreuung ordnet sie einer PROFILE-ID über den Dateinamen zu. Eine PROFILE-ID kann mehreren Benutzerkennungen zugeordnet sein. Benutzerkennungen mit gleicher PROFILE-ID arbeiten mit der gleichen Gruppensyntaxdatei (siehe Abschnitt „Namenskonventionen“ auf Seite 36).

Mit dem Kommando MODIFY-SDF-PARAMETERS kann die Systembetreuung
- eine bestehende Zuordnung ändern
- eine bestehende Zuordnung aufheben
- einer PROFILE-ID erstmals eine Gruppensyntaxdatei zuordnen.

In Abhängigkeit vom Operanden SCOPE ist diese Zuordnung entweder permanent, permanent ab der nächsten Session oder nur während der laufenden Session gültig. Eine mit MODIFY-SDF-PARAMETERS getroffene Festlegung gilt nur für solche Tasks, die *nach* dieser Festlegung erzeugt werden. Bereits existierende Tasks sind davon nicht betroffen. Ist eine Gruppensyntaxdatei einer PROFILE-ID zugeordnet, so wird sie nach dem LOGON-Vorgang automatisch aktiviert. Sie bleibt aktiviert bis zum Taskende.

Je Task kann immer nur eine Gruppensyntaxdatei aktiviert sein. Beim STARTUP erzeugt SDF eine Klasse-4-Liste, in der die Namen der Gruppensyntaxdateien und die zugeordneten PROFILE-IDs - wie in der SDF-Parameterdatei enthalten - stehen.

Falls bei der LOGON-Verarbeitung die zu aktivierende Gruppensyntaxdatei nicht verfügbar ist, z.B. weil sie gerade bearbeitet wird, so wird eine Meldung ausgegeben und die Task ohne Gruppensyntaxdatei erzeugt.

Bei nicht vorhandener oder ungültiger SDF-Parameterdatei wird eine Warnmeldung ausgegeben. Ist die zu aktivierende TSOS-Gruppensyntaxdatei fehlerhaft, so fordert SDF an der Konsole zur Eingabe einer fehlerfreien TSOS-Gruppensyntaxdatei auf. Beim nächsten LOGON-Vorgang unter der Benutzerkennung TSOS wird diese Gruppensyntaxdatei aktiviert.

Ab BS2000/OSD-BC V1.0 ist nur die Basis-Systemsyntaxdatei notwendig, in der auch die Kommando- und Programmdefinitionen für die Benutzerkennungen TSOS, SYSPRIV und SYSAUDIT enthalten sind. Da diesen Definitionen die Privilegien TSOS, SECURITY-ADMINISTRATION, SAT-FILE-MANAGEMENT oder SAT-FILE-EVALUATION zugeordnet sind, steht dieser besondere Funktionsumfang nur den Benutzern zur Verfügung, die das erforderliche Privileg besitzen.

Gruppensyntaxdateien werden im Rahmen bestimmter Produkte ausgeliefert oder vom Benutzer mit SDF-A selbst erstellt.

2.1.3 Benutzersyntaxdateien

Eine Benutzersyntaxdatei kann Erweiterungen, Einschränkungen und sonstige Änderungen gegenüber der Basis-Systemsyntaxdatei, gegenüber den Subsystem-Syntaxdateien und ggf. gegenüber der Gruppensyntaxdatei enthalten. Erweiterungen und sonstige funktionelle Änderungen sind beschränkt auf Anweisungen an Benutzer-Programme und auf Kommandos, die durch Prozeduren implementiert sind. Funktionelle Einschränkungen, die in den Systemsyntaxdateien oder in der Gruppensyntaxdatei für BS2000-Kommandos (durch System-Module implementiert) festgelegt sind, können in einer Benutzersyntaxdatei nicht aufgehoben werden.

Benutzersyntaxdateien können, müssen aber nicht aktiviert sein. Ab SDF V4.1 können pro Task gleichzeitig mehrere Benutzersyntaxdateien aktiviert sein. Ist ein Kommando oder eine Anweisung in mehreren gleichzeitig aktivierten Benutzersyntaxdateien definiert, dann gilt die Definition aus der zuletzt aktivierten Benutzersyntaxdatei.

Wenn eine Benutzersyntaxdatei den Namen $<userid>.SDF.USER.SYNTAX hat und eine Task unter der Benutzerkennung <userid> gestartet wird, dann wird diese Benutzersyntaxdatei nach der Abarbeitung des Kommandos SET-LOGON-PARAMETERS automatisch aktiviert. Falls zu diesem Zeitpunkt die zu aktivierende Benutzersyntaxdatei nicht verfügbar ist, z.B. weil sie gerade bearbeitet wird, wird eine Meldung ausgegeben und die Task ohne Benutzersyntaxdatei erzeugt.

Während der Task kann der Benutzer die Benutzersyntaxdatei(en) mit dem Kommando bzw. der Anweisung MODIFY-SDF-OPTIONS aktivieren. Mit MODIFY-SDF-OPTIONS kann er sie auch wieder deaktivieren.

Benutzersyntaxdateien sind mit SDF-A selbst zu erstellen.

2.2 Privilegienkonzept

Ab BS2000 V10.0 können die bisherigen Privilegien der Benutzerkennung TSOS auf verschiedene andere Benutzerkennungen verteilt werden, wenn das Softwareprodukt SECOS (siehe Handbuch „SECOS“ [10]) im System geladen ist. Den Benutzerkennungen können dabei so genannte systemglobale Privilegien zugeordnet werden, die sie zu bestimmten Aufgaben, z.B. zur systemglobalen Benutzerverwaltung (USER-ADMINISTRATION) oder zur Sicherheitsverwaltung (SECURITY-ADMINISTRATION), berechtigen. Besitzt eine Benutzerkennung ein solches Privileg, dann darf sie privilegierte Operationen ausführen und dazu bestimmte, privilegierte Kommandos benutzen.

2.2.1 Privilegien in Syntaxdateien

Privilegierte Objekte (Anwendungsbereiche, Programme, Kommandos, Anweisungen, Operanden und Operandenwerte) wurden früher in Gruppensyntaxdateien definiert, die den Benutzerkennungen mit entsprechendem Privileg zugewiesen werden. So wurde sichergestellt, dass keine andere Benutzerkennung Zugriff auf privilegierte Objekte hatte.

Mit SDF-A ab Version 2.0 kann für die Objekte in den Syntaxdateien (Anwendungsbereiche, Programme, Kommandos, Anweisungen, Operanden und Operandenwerte) einzeln festgelegt werden, welche Privilegien eine Benutzerkennung besitzen muss, um mit diesen Objekten arbeiten zu können (z.B. TSOS, USER-ADMINISTRATION oder SECURITY-ADMINISTRATION). Alle Objekte können dann in einer einzigen Syntaxdatei definiert sein, auf die alle Benutzer Zugriff haben. Ein Benutzerauftrag kann nur dann auf ein Objekt zugreifen, wenn er mindestens eines der dem Objekt zugeordneten Privilegien besitzt. Ein Benutzerauftrag kann auch mehrere Privilegien besitzen.

Privilegien sind Schlüsselwörter. Mögliche Privilegien sind im Benutzerhandbuch „SECOS“ [10] zu finden und außerdem von den installierten Softwareprodukten abhängig (z.B. die Privilegien HSMS-ADMINISTRATION und VM2000-ADMINISTRATION).

Privilegien werden einem Objekt mittels SDF-A mit der entsprechenden ADD-Anweisung zugeordnet und mit der entsprechenden MODIFY-Anweisung verändert.

Im geführten Dialog werden nur die Objekte angezeigt, auf die der Benutzer auf Grund seiner Privilegien tatsächlich zugreifen darf.

Die Angabe und Auswertung von Privilegien für Objekte der Syntaxdatei wird folgendermaßen unterstützt:
- von SDF-A ab Version 2.0
- von SDF ab Version 2.0
- in BS2000 ab BS2000/OSD-BC V1.0 in Verbindung mit SRPM (siehe Handbuch „SECOS“ [10]).

2.2.2 Hinweise und Regeln zur Privilegienvergabe

- Wird eine Privilegienliste verändert, so müssen die Privilegien, die erhalten bleiben, in der MODIFY-Anweisung wiederholt werden.
- Wird einer Struktur ein neues Privileg zugeordnet, so erhalten nur die abhängigen Strukturen, die mit PRIVILEGE=*SAME definiert sind, das neue Privileg. Das bedeutet, dass die abhängigen Strukturen, die eigene Privilegien besitzen, bei Änderungen separat (mit EDIT und MODIFY-...) verändert werden müssen.
- Die gleiche Prüfung wird in allen MODIFY- und ADD-Anweisungen durchgeführt. Damit wird verhindert, dass eine abhängige Struktur mehr Privilegien erhält als die ihr übergeordneten Strukturen.
- Wenn in einer Struktur ein Privileg gelöscht wird, so wird das Privileg in den abhängigen Strukturen, die mit eigenen Privilegien ausgestattet sind, ebenfalls gelöscht. Diese Maßnahme verhindert Inkonsistenzen im Syntaxbaum.
  In abhängigen Strukturen, die mit PRIVILEGE=*SAME definiert sind, wird das entsprechende Privileg auch gelöscht.
  Das letzte Privileg in einer abhängigen Struktur kann nicht gelöscht werden. Der Versuch wird mit einer Fehlermeldung abgewiesen. In diesem Fall muss die Struktur mit der Anweisung REMOVE entfernt werden.
- Hat ein Operand einen Default-Wert, dann muss dieser Default-Wert die gleichen Privilegien besitzen wie der Operand.
- Hat ein Operand keinen Default-Wert, so muss er die gleichen Privilegien besitzen wie seine übergeordnete Struktur (Kommando oder Wert). Für jedes Privileg des Operanden muss mindestens ein Wert mit diesem Privileg vorhanden sein. Diese beiden Regeln wurden festgelegt, damit jeder Benutzerauftrag unabhängig von seinen Privilegien dem Operanden einen Wert zuweisen kann.

2.3 Dateihierarchie

Die Benutzung von SDF setzt das Vorhandensein einer Basis-Systemsyntaxdatei voraus. Daneben können mehrere Subsystem-Syntaxdateien und pro Task eine Gruppen- und mehrere Benutzersyntaxdateien aktiviert sein. Ist ein Kommando oder eine Anweisung in mehreren aktivierten Syntaxdateien definiert, so gilt die Definition aus der zuletzt aktivierten Syntaxdatei.

Für die Fälle, in denen mehrere Typen von Syntaxdateien gemeinsam für eine Benutzerkennung aktiviert sind, ist festgelegt, aus welcher Syntaxdatei SDF die notwendige Information erhält. Daraus ergibt sich eine Hierarchie von Syntaxdateien, die für eine Task unter einer bestimmten Benutzerkennung gültig ist. Folgende Konstellationen sind möglich:

- nur die Systemsyntaxdateien sind aktiviert
- die Systemsyntaxdateien und eine Gruppensyntaxdatei sind aktiviert
- die Systemsyntaxdateien und die Benutzersyntaxdateien sind aktiviert
- die Systemsyntaxdateien, eine Gruppensyntaxdatei und die Benutzersyntaxdateien sind aktiviert
- nur eine Gruppensyntaxdatei ist aktiviert
- eine Gruppensyntaxdatei und die Benutzersyntaxdateien sind aktiviert

Bei allen Konstellationen ist zu beachten, dass ab BS2000/OSD-BC V1.0 die möglichen Kommando- und Anweisungseingaben auch von den Privilegien abhängig sind, die diesen Objekten zugeordnet wurden.

Nur die Systemsyntaxdateien sind aktiviert

Im einfachsten Fall sind lediglich die Systemsyntaxdateien aktiviert. Welche Kommando- und Anweisungseingaben möglich sind, hängt dann allein vom Inhalt der Systemsyntaxdateien ab.

Systemsyntaxdateien und Gruppensyntaxdatei sind aktiviert

Ist neben den Systemsyntaxdateien für die Benutzerkennung auch eine Gruppensyntaxdatei aktiviert, so haben die in der Gruppensyntaxdatei enthaltenen Definitionen (einschließlich Globalinformation) Vorrang vor denen in den Systemsyntaxdateien. Als Definition gilt auch die Sperrung eines Kommandos oder einer Anweisung in der Gruppensyntaxdatei. Nur wenn ein Kommando oder eine Anweisung ausschließlich in den Systemsyntaxdateien definiert ist, nimmt SDF die Information von dort. Welche Kommando- und Anweisungseingaben möglich sind, hängt vom Inhalt beider Dateiarten ab.

Systemsyntaxdateien und Benutzersyntaxdateien sind aktiviert

Sind neben den Systemsyntaxdateien für die Task auch Benutzersyntaxdateien aktiviert, so müssen die in den Benutzersyntaxdateien enthaltenen Definitionen der BS2000-Kommandos (durch System-Module implementiert) voll durch die Systemsyntaxdateien abgedeckt sein. Ansonsten haben die in den Benutzersyntaxdateien enthaltenen Definitionen (einschließlich Sperrungen, einschließlich Globalinformation) Vorrang vor denen in den Systemsyntaxdateien.

System-, Gruppen- und Benutzersyntaxdateien sind aktiviert

Sind neben den Systemsyntaxdateien sowohl eine Gruppensyntaxdatei als auch Benutzersyntaxdateien aktiviert, so gilt grundsätzlich:

Der in den Benutzersyntaxdateien zugelassene Kommandovorrat wird durch die Definitionen bestimmt, die in der Gruppensyntaxdatei und in den Systemsyntaxdateien enthalten sind. Das bedeutet, dass eine Benutzersyntaxdatei keine Systemkommandos enthalten kann,

- die in der Gruppensyntaxdatei oder in den Systemsyntaxdateien gesperrt sind,
- die in der Gruppensyntaxdatei oder in den Systemsyntaxdateien nicht vorhanden sind,
- für die der Benutzer nicht mindestens eines der erforderlichen Privilegien hat.

Sperrungen, Veränderungen und Globalinformation in Benutzersyntaxdateien haben jedoch Vorrang vor den in Gruppen- und Systemsyntaxdateien enthaltenen Definitionen.

Ist ein Systemkommando (durch Systemmodule implementiert) in allen drei Syntaxdateiarten definiert, dann muss die Definition in der Benutzersyntaxdatei durch die Gruppensyntaxdatei voll abgedeckt sein.

Folgende Fälle sind zu unterscheiden:

- Eine Benutzersyntaxdatei enthält vom Benutzer selbst erstellte Kommandos (durch Prozeduren implementiert):
  Alle in der Prozedur enthaltenen Systemkommandos (durch Systemmodule implementiert) müssen durch Definitionen in den Systemsyntaxdateien oder in der Gruppensyntaxdatei abgedeckt sein.
- Eine Benutzersyntaxdatei enthält eine Sperrung für ein Kommando:
  Das Kommando kann nicht ausgeführt werden, auch wenn seine Definition durch die Gruppensyntaxdatei oder die Systemsyntaxdateien abgedeckt ist.
- Eine Benutzersyntaxdatei enthält ein Systemkommando, bei dem ein Operandenwert geändert wurde:
  Die Ausführung des Kommandos erfolgt wie in der Benutzersyntaxdatei definiert.

- Die Gruppensyntaxdatei oder die Systemsyntaxdateien enthalten ein Kommando, das in keiner Benutzersyntaxdatei definiert ist:
  Der Benutzer darf das Kommando verwenden, wenn es nicht in einer Benutzersyntaxdatei gesperrt ist.
- Die Gruppensyntaxdatei enthält eine Sperrung für ein Kommando:
  Das Kommando kann nicht ausgeführt werden, auch wenn es in der Benutzersyntaxdatei definiert und durch eine Definition in den Systemsyntaxdateien abgedeckt ist.

Nur eine Gruppensyntaxdatei ist aktiviert

Mit dem Kommando MODIFY-SDF-PARAMETERS (siehe Handbuch „SDF-Verwaltung“ [2]) kann einer PROFILE-ID eine Gruppensyntaxdatei zugeordnet werden. Wird der Operand HIERARCHY=*NO angegeben, so wird für alle Tasks der Benutzerkennungen mit dieser PROFILE-ID bei der LOGON-Verarbeitung *nur* die festgelegte Gruppensyntaxdatei aktiviert (keine Systemsyntaxdateien). In diesem Fall hängt es allein vom Inhalt dieser Gruppensyntaxdatei ab, welche Kommando- und Anweisungseingaben möglich sind.

Eine Gruppensyntaxdatei und Benutzersyntaxdateien sind aktiviert

Sind für eine Task keine Systemsyntaxdateien aktiviert (s.o.) und neben der Gruppensyntaxdatei sind eine oder mehrere Benutzersyntaxdateien aktiviert, so müssen die in den Benutzersyntaxdateien enthaltenen Definitionen der BS2000-Kommandos (durch System-Module implementiert) voll durch die Gruppensyntaxdatei abgedeckt sein. Ansonsten haben die Definitionen in den Benutzersyntaxdateien (einschließlich Sperrungen und Globalinformation) Vorrang vor den Definitionen in der Gruppensyntaxdatei.

2.4 Namenskonventionen

Neben den Systemsyntaxdateien können pro Benutzerauftrag jeweils eine Gruppen- und mehrere Benutzersyntaxdateien aktiviert sein. Für diesen Fall ist eine Dateihierarchie festgelegt, die bestimmt, wie SDF mit mehreren Syntaxdateien arbeitet. Systemsyntaxdateien und Gruppensyntaxdateien können mit dem Kommando MODIFY-SDF-PARAMETERS aktiviert werden, Benutzersyntaxdateien mit dem Kommando MODIFY-SDF-OPTIONS. Die Namen der zu aktivierenden System- und Gruppensyntaxdateien werden in einer SDF-Parameterdatei eingetragen.

Die Basis-Systemsyntaxdatei wird nach dem Laden von SDF automatisch aktiviert. Dabei wird die in der Parameterdatei festgelegte Datei aktiviert. Dasselbe gilt für die Subsystem-Syntaxdateien, deren Namen in der SDF-Parameterdatei eingetragen sind.
Ist in den DSSM-Deklarationen für SDF keine Parameterdatei festgelegt, wird die Datei $TSOS.SYSPAR.SDF als Parameterdatei verwendet. Hat die Parameterdatei keinen gültigen Inhalt, so wird über die Konsolmeldung CMD0691 ein neuer Name angefordert. Bei Antwort „*STD“ wird $TSOS.SYSSDF.SDF.045 als Basis-Systemsyntaxdatei aktiviert (und zusätzlich $TSOS.SYSSDF.BS2CP.<bs2vers> als Subsystem-Syntaxdatei).
Während des Systemlaufs kann die Systembetreuung die aktivierte Basis-Systemsyntaxdatei wechseln (mit dem Kommando MODIFY-SDF-PARAMETERS). Die während des Systemlaufs aktivierte Basis-Systemsyntaxdatei kann einen beliebigen Namen haben.

Die Systembetreuung weist einer PROFILE-ID eine Gruppensyntaxdatei zu. Während einer Session kann diese Zuweisung mit dem Kommando MODIFY-SDF-PARAMETERS geändert werden. In Abhängigkeit vom Operanden SCOPE ist diese Änderung entweder permanent, permanent ab der nächsten Session oder nur während der laufenden Session wirksam.
Eine so zugewiesene Gruppensyntaxdatei wird nach der LOGON-Verarbeitung für jede Task unter einer Benutzerkennung mit der entsprechenden PROFILE-ID automatisch aktiviert.

Benutzersyntaxdateien sind an eine Benutzerkennung gekoppelt. Wenn eine Benutzersyntaxdatei nach der LOGON-Verarbeitung automatisch aktiviert werden soll, muss sie unter dem Namen $<userid>.SDF.USER.SYNTAX katalogisiert sein. Der Benutzer kann Benutzersyntaxdateien im laufenden Benutzerauftrag aktivieren oder deaktivieren (MODIFY-SDF-OPTIONS). Wird eine Benutzersyntaxdatei mit MODIFY-SDF-OPTIONS aktiviert, kann sie einen beliebigen Namen haben.

2.5 Syntaxdateien bearbeiten

Mit SDF-A können Sie neue Syntaxdateien erstellen oder bestehende ändern. SDF-A arbeitet mit den Syntaxdateien ausschließlich auf der logischen Ebene, d.h. die physische Struktur der Syntaxdateien bleibt verborgen.

Ab SDF-A V3.0 verfügt jeder Benutzer über den vollen Funktionsumfang von SDF-A. Damit kann er zwar (mit Rücksicht auf Dateiattribute und Privilegien) Gruppen- und Systemsyntaxdateien bearbeiten, er selbst darf aber keine Gruppen- und Systemsyntaxdateien aktivieren (siehe auch Seite 36). Nur die Systembetreuung kann steuern, welche Systemsyntaxdateien für das System und welche Gruppensyntaxdatei für eine Benutzerkennung aktiviert wird.

2.5.1 Übersicht über die Leistungen von SDF-A

Mit SDF-A können im Einzelnen folgende Operationen vorgenommen werden:

- Definieren eigener Kommandos, die durch Kommandoprozeduren implementiert sind, und Ändern dieser Definitionen
- Definieren eigener Programme und der Anweisungen an diese sowie Ändern dieser Definitionen
- Sperren von Kommandos, Programmen, Anweisungen, Operanden und Operandenwerten
- Ändern von Standardwerten
- Umbenennen von Anwendungsbereichen, Kommando-, Anweisungs- und Operandennamen sowie Operandenwerten, die Schlüsselwörter sind
- Ändern oder Austauschen von Hilfetexten für den geführten Dialog, z.B. Ersetzen von deutschen Hilfetexten durch solche in der Landessprache
- Ändern der Kommandozuordnung zu Anwendungsbereichen und Definieren neuer Anwendungsbereiche
- Ändern der Voreinstellungen der SDF-Optionen, z.B. für den geführten Dialog
- Löschen und Wiederherstellen von Kommando- und Anweisungsdefinitionen

2.5.2 Syntaxdatei öffnen

Mit der SDF-A-Anweisung OPEN-SYNTAX-FILE öffnen Sie eine Syntaxdatei für die folgende Bearbeitung.

Systemsyntaxdatei

Beim Öffnen einer zu bearbeitenden Systemsyntaxdatei können Sie mit dem Operanden SYSTEM-CONTROL als Referenz eine weitere Systemsyntaxdatei (beliebigen Namens) zuweisen. Diese Systemsyntaxdatei wird im Folgenden als SYSTEM-CONTROL-Datei bezeichnet und sie sollte die Definitionen von allen systemweit angebotenen Kommandos in der Fassung enthalten, die von der Implementierung geliefert wurde. Insbesondere sollte diese Datei die von Fujitsu Siemens Computers gelieferten Definitionen der Benutzerkommandos in ihrer Originalfassung enthalten. Im einfachsten Fall ist diese Datei lediglich eine Kopie der Basis-Systemsyntaxdatei, die Sie von Fujitsu Siemens Computers erhalten haben.

Anhand der SYSTEM-CONTROL-Datei prüft SDF-A, ob die vorzunehmenden Änderungen mit der Implementierung vereinbar sind. Wenn Sie die SYSTEM-CONTROL-Datei nicht zuweisen, kann SDF-A die Syntaxdatei in unzulässiger Weise ändern. Das führt dann später, wenn die geänderte Systemsyntaxdatei aktiviert ist, zu Schwierigkeiten.

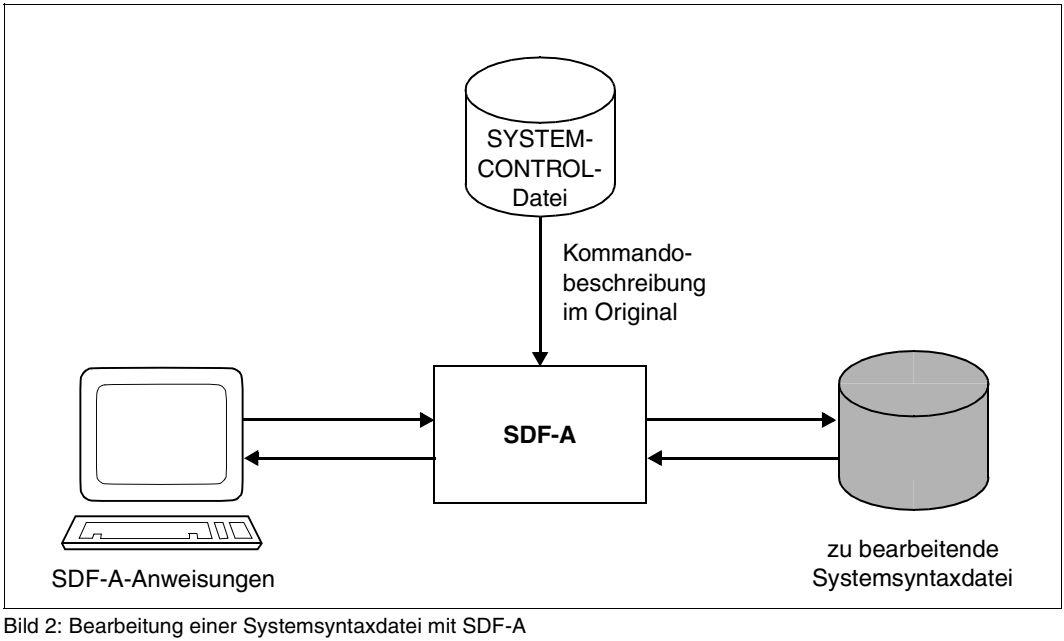


Bild 2: Bearbeitung einer Systemsyntaxdatei mit SDF-A

Gruppensyntaxdatei

Beim Eröffnen einer zu bearbeitenden Gruppensyntaxdatei können Sie mit dem Operanden SYSTEM-CONTROL als Referenz eine Systemsyntaxdatei (beliebigen Namens) zuweisen. Für sie gilt das Gleiche wie für die SYSTEM-CONTROL-Datei, die beim Eröffnen einer Systemsyntaxdatei zugewiesen werden kann.

Eine besondere SYSTEM-CONTROL-Datei brauchen Sie, wenn Sie mit BS2000 V10 arbeiten und die Gruppensyntaxdatei für die Systembetreuung bearbeiten wollen. Sie erhalten diese Datei, indem Sie in eine Kopie der vorhandenen SYSTEM-CONTROL-Datei die Definitionen der Kommandos und Programme, die der Systembetreuung ganz oder teilweise vorbehalten sind, mit COPY...,OVERWRITE-POSSIBLE=*YES kopieren (siehe Anweisung COPY, Seite 212).

Außerdem können Sie mit den Operanden SYSTEM-DESCRIPTIONS eine weitere Systemsyntaxdatei beliebigen Namens zuweisen. Dies sollte die Datei sein, die später zusammen mit der bearbeiteten Gruppensyntaxdatei aktiviert sein wird.

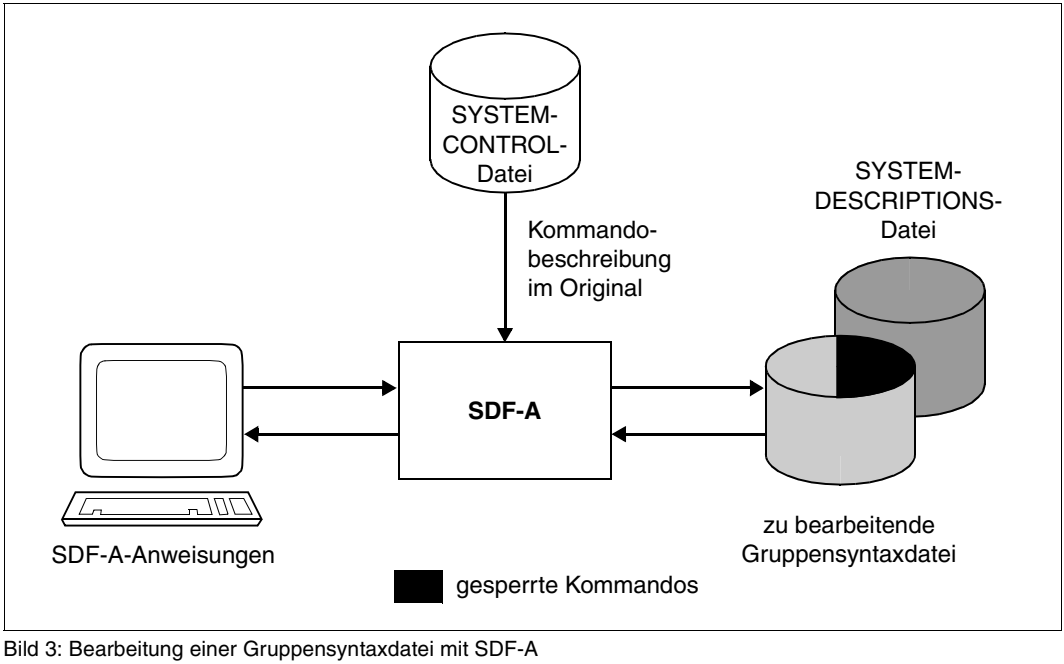


Bild 3: Bearbeitung einer Gruppensyntaxdatei mit SDF-A

Wenn Sie die Gruppensyntaxdatei mit Angabe der SYSTEM-DESCRIPTIONS-Datei öffnen, so können Sie auch die Kommandos und Anweisungen bearbeiten, die ausschließlich in der SYSTEM-DESCRIPTIONS-Datei definiert sind. Wird eine Definition geändert, so schreibt SDF-A die geänderte Fassung in die Gruppensyntaxdatei. In der SYSTEM-DESCRIPTIONS-Datei bleibt die Definition unverändert.

Wenn Sie ein Kommando sperren wollen, das in der SYSTEM-DESCRIPTIONS-Datei definiert ist, so müssen Sie diese beim Eröffnen der Gruppensyntaxdatei unbedingt angeben. Andernfalls löscht SDF-A lediglich die Definition in der Gruppensyntaxdatei (sofern dort vorhanden), ohne dort einen entsprechenden Sperrvermerk einzutragen.

Wollen Sie eine in der Gruppensyntaxdatei vorgenommene Kommandomodifizierung aufheben, so dürfen Sie die SYSTEM-DESCRIPTIONS-Datei nicht angeben. Sie können dann die modifizierte Definition in der Gruppensyntaxdatei löschen. SDF benutzt dann die (nicht modifizierte) Definition aus der Systemsyntaxdatei.

Benutzersyntaxdatei

Beim Öffnen einer zu bearbeitenden Benutzersyntaxdatei können Sie mit dem Operanden USER-CONTROL als Referenz eine Benutzersyntaxdatei (beliebigen Namens) zuweisen. Diese sollte die Definitionen von benutzereigenen Kommandos in ihrer ursprünglichen Fassung enthalten. Ansonsten gilt für sie das Gleiche wie für die SYSTEM-CONTROL-Datei, die beim Eröffnen einer Gruppen- oder Systemsyntaxdatei zugewiesen werden kann.

Außerdem können Sie mit den Operanden GROUP- bzw. SYSTEM-DESCRIPTIONS eine Gruppen- bzw. Systemsyntaxdatei zuweisen. Dies sollten die Dateien sein, die später zusammen mit der bearbeiteten Benutzersyntaxdatei aktiviert sein werden.

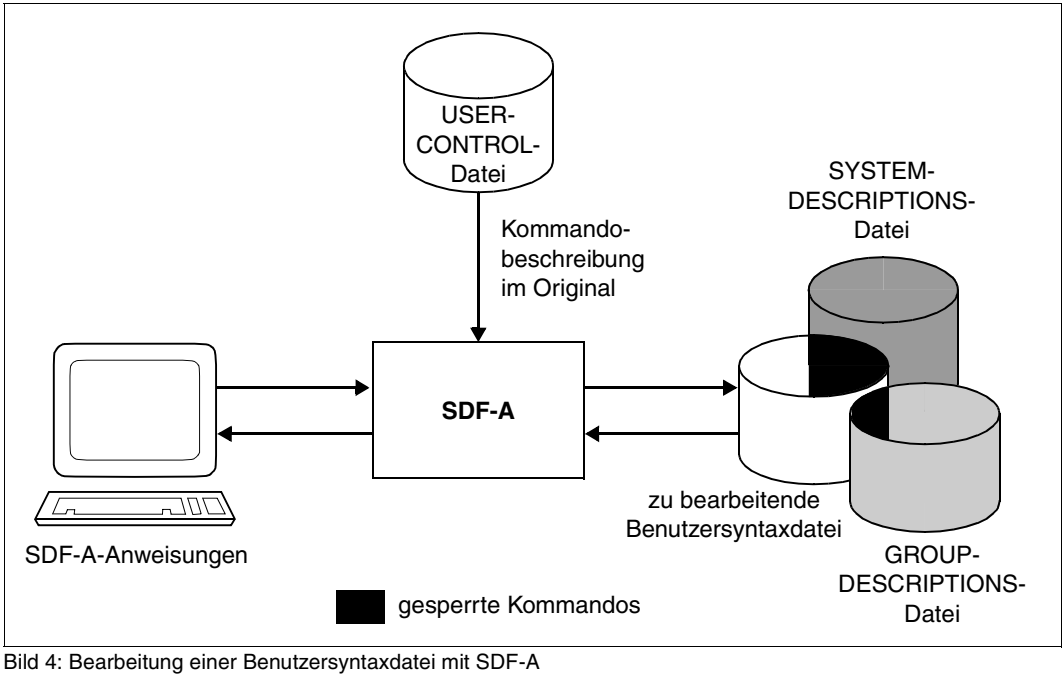


Bild 4: Bearbeitung einer Benutzersyntaxdatei mit SDF-A

Für die GROUP- und SYSTEM-DESCRIPTIONS-Datei, die Sie beim Öffnen einer Benutzersyntaxdatei angeben, gilt prinzipiell das Gleiche wie für die SYSTEM-DESCRIPTIONS-Datei beim Eröffnen einer Gruppensyntaxdatei. Anders als dort muss jedoch der in der Benutzersyntaxdatei definierte Funktionsumfang der BS2000-Kommandos (durch System-Module implementiert) voll durch die GROUP- und SYSTEM-DESCRIPTIONS-Datei abgedeckt sein. Zusätzlich haben diese Dateien hier noch die Funktion der SYSTEM-CONTROL-Datei. In der Benutzersyntaxdatei können Sie Definitionen von Kommandos, die durch System-Module implementiert sind, nur verändern, aber nicht neu definieren.

2.5.3 Bearbeiten von Kommando- und Anweisungsdefinitionen

Kommando- und Anweisungsdefinitionen bestehen aus

– einer kommando- bzw. anweisungsglobalen Definition (Kopf),

– einer Definition für jeden Operanden und

– einer Definition für jeden Operandenwert (Eingabealternative).

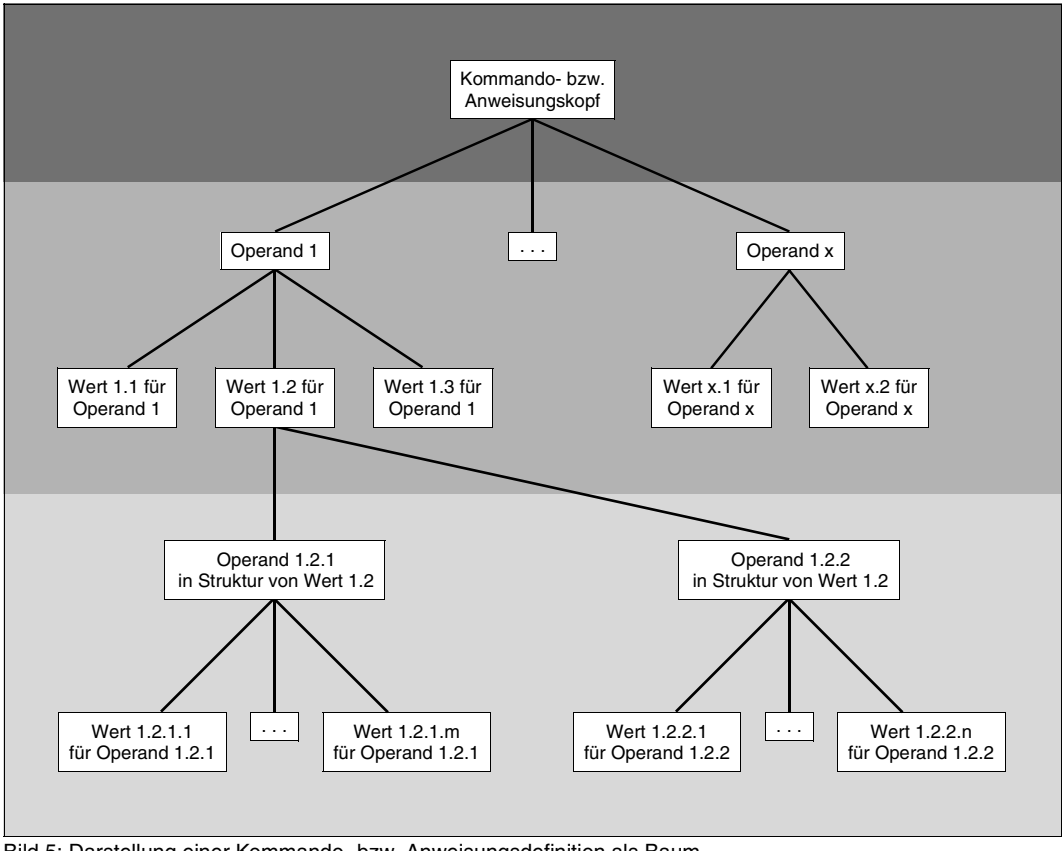


Bild 5: Darstellung einer Kommando- bzw. Anweisungsdefinition als Baum

Für jeden Operanden muss mindestens ein Operandenwert definiert sein. Das gilt auch für den Fall, dass ein Operand mit einem Default-Wert an der Benutzeroberfläche des Kommandos bzw. der Anweisung ausgeblendet ist. Hängt an einem Operandenwert eine Struktur (siehe „Fachwörter“ auf Seite 637), so ist diese global in der Definition des Operandenwerts beschrieben. Zusätzlich gibt es

– für jeden einzelnen Operanden der Struktur eine Definition und

– für jeden Wert, der zu einem dieser Operanden gehört, eine Definition.

Abgeschlossen wird die Struktur mit der Anweisung CLOSE-STRUCTURE.

Mit der Anweisung CLOSE-CMD-OR-STMT wird die Kommando- bzw. Anweisungsdefinition abgeschlossen.

Anweisungen an SDF-A können mit Kommentaren versehen werden; diese müssen in Anführungszeichen (") stehen.

```
//ADD-CMD ... bzw. //ADD-STMT
//ADD-OPERAND <Operand 1>,...
//ADD-VALUE <Wert 1.1>,...
//ADD-VALUE <Wert 1.2>,...,STRUCTURE=*YES(...),...
//ADD-OPERAND <Operand 1.2.1>,...
//ADD-VALUE <Wert 1.2.1.1>,...
. . .
//ADD-VALUE <Wert 1.2.1.m>,...
//ADD-OPERAND <Operand 1.2.2>,...
//ADD-VALUE <Wert 1.2.2.1>,...
. . .
//ADD-VALUE <Wert 1.2.2.n>,...
//CLOSE-STRUCTURE
//ADD-VALUE <Wert 1.3>,...
. . .
//ADD-OPERAND <Operand x>,...
//ADD-VALUE <Wert x.1>,...
//ADD-VALUE <Wert x.2>,...
//CLOSE-CMD-OR-STMT
```

Bild 6: Folge der SDF-A-Anweisungen, mit denen die in Bild 5 gezeigte Definition erstellt wird

Eine Kommando- oder Anweisungsdefinition können Sie

– neu erstellen,

– ändern,

– löschen oder

– auf SYSOUT oder SYSLST ausgeben.

Das Gleiche können Sie mit der Definition eines Operanden oder Operandenwerts tun.

Wenn Sie eine Kommando- oder Anweisungsdefinition neu erstellen, so müssen Sie nicht unbedingt auch alle zugehörigen Operanden- und Operandenwertdefinitionen neu erstellen. Häufig können Sie auf vorhandene Operanden- bzw. Operandenwertdefinitionen zurückgreifen, die Sie in die neu zu erstellende Kommando- bzw. Anweisungsdefinition kopieren (siehe COPY). Die vorhandene Definition muss nicht unbedingt exakt die benötigte Definition sein. Wenn sich eine vorhandene Definition nur geringfügig von der benötigten unterscheidet, so ist es sinnvoll, sie zu kopieren und anschließend die Kopie entsprechend zu ändern (siehe unten).

Folgendes Verfahren zur Neuerstellung einer Kommando- oder Anweisungsdefinition ist sinnvoll:

– *1. Schritt*

  Festlegung des Kommando- bzw. Anweisungsgerüsts (Kommando- bzw. Anweisungsname, Operanden, Operandenwerte, Strukturen) mit sehr einfachen ADD-Anweisungen, die nur wenige Angaben (Standardwerte) enthalten.

– *2. Schritt*

  Ausgabe der vorläufigen Kommando- bzw. Anweisungsdefinition mit der SHOW-Anweisung und Überprüfung.

– *3. Schritt*

  Vervollständigung der vorläufigen Kommando- bzw. Anweisungsdefinition mit MODIFY-Anweisungen im geführten Dialog (Einbringen von Hilfetexten und Implementierungsangaben). SDF-A führt dabei durch den Kommando- bzw. Anweisungsbaum.

Sie ändern eine Kommando- oder Anweisungsdefinition, indem Sie entweder die globale Definition ändern (siehe MODIFY-CMD bzw. MODIFY-STMT) oder Operanden- bzw. Operandenwertdefinitionen

– hinzufügen durch Neuerstellen (siehe ADD-OPERAND bzw. ADD-VALUE) oder durch Kopieren (siehe COPY),

– ändern (siehe MODIFY-OPERAND bzw. MODIFY-VALUE) oder

– löschen (siehe REMOVE).

Das Hinzufügen oder Ändern von Operanden- bzw. Operandenwertdefinitionen erfordert, dass Sie zuvor explizit mit der Anweisung EDIT oder implizit bei der vorangegangenen Bearbeitung auf die zu bearbeitende Stelle in der Kommando- bzw. Anweisungsdefinition positionieren.

Das Löschen einer Kommando- oder Anweisungsdefinition geschieht mit der Anweisung REMOVE.

Das Sperren eines Kommandos, einer Anweisung, eines Programms, eines Operanden oder eines Operandenwerts geschieht ebenfalls mit der Anweisung REMOVE. Eine systemweite Sperre wird in der Basis-Systemsyntaxdatei definiert. Kommandos, die durch System-Module implementiert sind, sowie Operanden und Operandenwerte dieser Kommandos lassen sich auch kennungsspezifisch in einer Gruppensyntaxdatei sperren. Beim Definieren einer Sperre in der Gruppensyntaxdatei muss die Basis-Systemsyntaxdatei als Referenzdatei zugewiesen sein. Eine Sperre kann auch in einer Benutzersyntaxdatei definiert werden. Da aber der Benutzer eine Benutzersyntaxdatei selbst aktiviert und deaktiviert, ist eine in ihr definierte Sperre nur für den Benutzer selbst von Bedeutung.

Aufheben können Sie eine Sperre mit der Anweisung RESTORE für Kommandos, Programme und Anweisungen. Dazu müssen deren Definitionen noch in einer Syntaxdatei vorhanden sein, die höher in der Dateihierarchie steht als die gerade zu bearbeitende Syntaxdatei.

Bei der Ausgabe von Definitionen (siehe SHOW) können Sie bestimmen,

– wie ausführlich Sie diese haben wollen und
– ob Sie sie in einer handbuchähnlichen Form erhalten möchten oder als Folge der SDF-A-Anweisungen, mit denen sie definiert werden.

2.6 Sicherheit von Syntaxdateien

Während der Bearbeitung einer Syntaxdatei mit SDF-A kann es zu Unterbrechungen kommen. Eine Unterbrechung kann z.B. durch die K2-Taste oder durch die Anweisungen HOLD-PROGRAM oder EXECUTE-SYSTEM-CMD ausgelöst werden. In besonders kritischen Abschnitten wird die Betätigung der K2-Taste ignoriert.

Eine Syntaxdatei, die nur zum Lesen geöffnet wurde (OPEN-SYNTAX-FILE ....,MODE= *READ) ist durch Unterbrechungen nicht gefährdet.

Eine Syntaxdatei, die erzeugt oder geändert werden soll, muss jedoch mit der Angabe MODE=*CREATE / *UPDATE geöffnet werden. Solange die Verarbeitung eines Objektes nicht abgeschlossen ist, kann eine Unterbrechung das Objekt oder auch die Syntaxdatei gefährden. SDF-A gibt bei Gefahr von Datenverlust Warnmeldungen aus, wenn Unterbrechungen auftreten.
SDF-A löst nur dann einen Schreibvorgang auf Platte aus, wenn ein Objekt explizit geschlossen wird. Das ist bei folgenden Anweisungen der Fall:

- CLOSE-CMD-OR-STMT schließt die Kommando- oder Anweisungsdefinition
- OPEN öffnet eine neue Syntaxdatei und speichert vorher die zurzeit geöffnete Syntaxdatei (sofern vorhanden).
- END speichert die aktuelle Syntaxdatei und beendet SDF-A.

Achtung!
Tritt während des Abspeicherns eine Unterbrechung auf, so kann die Syntaxdatei möglicherweise zerstört werden.

Das Erzeugen, Löschen oder Ändern eines Kommandos bzw. einer Anweisung erfordert meist mehrere Schreibvorgänge, da auch die Kommando- oder Anweisungstabellen aktualisiert werden müssen. Unterbrechungen sollten gerade bei solchen Operationen unbedingt vermieden werden, da sonst Inkonsistenzen zwischen den Kommando-/Anweisungstabellen und den tatsächlich vorhandenen Objekten auftreten können. Entdeckt SDF solche Inkonsistenzen bei der Aktivierung der Syntaxdatei, dann wird die Syntaxdatei abgewiesen.
Eine von SDF abgewiesene Syntaxdatei können Sie erneut mit SDF-A öffnen mit MODE=*UPDATE(...). Sie erhalten dann die Warnmeldung SDA0446. Nach einer Prüfung und, wenn nötig, einer Korrektur der Syntaxdatei kann diese abgespeichert werden. Sie wird danach von SDF nicht mehr abgewiesen.

Sie können Folgendes tun, um Probleme zu vermeiden:

- Führen Sie nach Möglichkeit immer ein CLOSE-CMD-OR-STMT durch, bevor Sie sich mit SHOW Objekte anzeigen lassen. Sie verhindern damit den Verlust von Objekten, falls die SHOW-Ausgabe mit K2 unterbrochen wird und Sie nicht sofort mit RESUME-PROGRAM zu SDF-A zurückkehren.
- Öffnen Sie Syntaxdateien nur dann mit MODE=*UPDATE(..), wenn Sie Änderungen vornehmen wollen.

2.7 Syntaxdateien mit altem Format

Die Objekte in den Syntaxdateien sind nicht in allen SDF-A-Versionen gleich strukturiert. Sie sollten deshalb die folgenden Hinweise beachten, um Kompatibilitätsprobleme zu vermeiden:

- SDF und SDF-A ab V2.0 können sowohl das alte als auch das neue Format der Syntaxdateien behandeln. SDF-A konvertiert die Objekte, auf die es zugreift, in das neue Format. SDF dagegen konvertiert die Objekte nur im virtuellen Speicher und ändert nichts in den Syntaxdateien.
- Syntaxdateien, die mit einer SDF-Version < 2.0 laufen sollen, dürfen nicht mit SDF-A-Versionen ≥ 2.0 erzeugt werden.
- Zur Korrektur von Syntaxdateien mit altem Format (bis SDF V1.4 benötigtem Format) steht folgende Funktion zur Verfügung:
  Ist der Auftragsschalter 15 gesetzt, so lädt SDF-A (ab V2.0) intern das Programm SDF-A-V1 mit der Funktionalität von SDF-A V1.0D nach. Die Syntax der SDF-A-Anweisungen entspricht dann der von SDF-A V1.0D, allerdings mit drei Ausnahmen:
  – Der Default-Wert der Operanden SYSTEM-DESCRIPTIONS und GROUP-DESCRIPTIONS der Anweisung OPEN-CMD-SET ist *NO anstatt *CURRENT, sodass standardmäßig keine Referenzsyntaxdateien zugewiesen sind. Falls die Referenzsyntaxdateien explizit angegeben werden, müssen sie ebenfalls altes Format haben.
  – Systemsyntaxdateien können damit nicht erzeugt werden.
  – Die SDF-A-Anweisung SHOW ist nicht verfügbar.
- Werden Syntaxdateien mit altem, von SDF-A V1.0D erzeugtem Format mit SDF-A ab V2.0A bearbeitet, so erhalten die bearbeiteten Objekte und die Globalinformationen das neue Format.
  Diese Syntaxdateien sind dann *nicht mehr* unter einer SDF-Version < 2.0 einsetzbar.
- Ab SDF-A V4.0 haben Sie mit der Anweisung DEFINE-ENVIRONMENT die Möglichkeit, das Syntaxdatei-Format auszuwählen. Z.B. behalten mit SDF-A V3.0 erzeugte und bearbeitete Syntaxdateien ihr ursprüngliches Format, wenn Sie SYNTAX-FILE-FORMAT=*V3 angegeben.
  Eine Syntaxdatei kann nie mit einer kleineren SDF-A-Version geändert werden als die SDF-A-Version, mit der sie erzeugt oder abgespeichert wurde.

3 Kommando- und Anweisungsdefinitionen ändern

Die Systembetreuung kann die von Fujitsu Siemens Computers gelieferten Kommando- und Anweisungsdefinitionen systemweit (Systemsyntaxdateien) oder gezielt für einzelne Benutzer (Gruppen- oder Benutzersyntaxdatei) ändern. Wenn sie eine Änderung systemweit durchgeführt hat, kann sie diese gezielt für einzelne Benutzer (mittels Gruppensyntaxdatei) wieder aufheben.

Ab SDF-A V3.0 verfügt jeder Benutzer über den vollen Funktionsumfang von SDF-A. Damit kann er zwar (mit Rücksicht auf Dateiattribute und Privilegien) Gruppen- und Systemsyntaxdateien bearbeiten, er selbst kann jedoch keine Gruppen- oder Systemsyntaxdateien aktivieren. Nur die Systembetreuung kann steuern, welche Systemsyntaxdateien für das System und welche Gruppensyntaxdatei für eine Benutzerkennung aktiviert wird. Beispielsweise können Sie eine Gruppensyntaxdatei mit modifizierten Kommandodefinitionen erstellen. Die Systembetreuung kann dann die von Ihnen erstellte Gruppensyntaxdatei übernehmen und einer Benutzerkennung zuordnen, sodass sie bei der LOGON-Verarbeitung aktiviert wird.

Die von Fujitsu Siemens Computers gelieferten Definitionen der Kommandos, die durch System-Module implementiert sind, können Sie nur insoweit ändern, wie der Funktionsumfang von der Änderung unberührt bleibt. Beispielsweise können Sie den Default-Wert eines Operanden ändern, nicht aber den Datentyp eines für diesen Operanden definierten Werts. Eine Änderung besteht häufig aus einer Vielzahl einzelner aufeinander abgestimmter Arbeitsschritte. Für mehrmals durchzuführende Änderungen ist es deshalb ratsam, Änderungsprozeduren zu schreiben, in denen alle SDF-A-Anweisungen zum Ändern eines Kommandos oder einer Anweisung enthalten sind. Beim Versionswechsel eines Produktes sollten diese Prozeduren zur Anpassung der Gruppen- und Benutzersyntaxdateien verwendet werden.

Hauptinhalt dieses Kapitels sind Beispiele, die die Vorgehensweise und die Möglichkeiten, die SDF-A bietet, zeigen. Am Schluss des Kapitels sind einige generelle Hinweise für das Definieren und Aufheben von funktionellen Einschränkungen zusammengestellt.

Werden Kommandodefinitionen auf Grund des Versionswechsels eines Produktes von Fujitsu Siemens Computers neu ausgeliefert, müssen Gruppen- und Benutzersyntaxdateien, in denen diese Kommandodefinitionen geändert wurden, neu aufgebaut werden.

3.1 Beispiele

Die folgenden Beispiele zeigen einige typische Anwendungsfälle von SDF-A.
Die Ablaufbeispiele wurden mit SDF-A V4.1E unter BS2000/OSD-BC V5.0 erstellt. In der Testumgebung ist u.a. auch das kostenpflichtige Produkt SDF-P (siehe Handbuch „SDF-P“ [12]) geladen. Die ausgegebenen Meldungen sind von der aktuellen System-Konfiguration abhängig.
In den Ablaufbeispielen werden Benutzereingaben in **halbfetter, dicktengleicher** Schrift dargestellt. Ausgaben, auf die besonders hingewiesen wird, sind in *kursiver, dicktengleicher* Schrift dargestellt.

3.1.1 Beispiel 1: Kommandos sperren

Die Benutzer der Benutzerkennung EXAMPLE sollen keine Programme laden und starten dürfen. Das lässt sich dadurch erreichen, dass für die Benutzerkennung EXAMPLE eine Gruppensyntaxdatei eingerichtet wird, in der die Kommandos LOAD-PROGRAM, START-PROGRAM sowie die alten Kommandos LOAD und EXECUTE gesperrt werden. Die speziellen START-Kommandos bestimmter Programme (z.B. START-SDF-A, START-EDT, START-LMS, ...) sind dann ebenfalls gesperrt, da diese START-Kommandos auf die Kommandos START-PROGRAM oder LOAD-PROGRAM zurückgreifen.

Ab BLSSERV V2.3 wird die Funktionalität von START- und LOAD-PROGRAM zusätzlich mit verbesserter Syntax über die neuen Kommandos START- und LOAD-EXECUTABLE-PROGRAM angeboten. In diesem Fall müssen die beiden neuen Kommando ebenfalls gesperrt werden.

```
/set-logon-parameters tsos,...  ————————————————————————————————  (1)
 ...
/start-sdf-a  ——————————————————————————————————————————————————  (2)
% %  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,*group,*create  ————  (3)
//remove *command((load,exec,load-prog,start-prog,load-exec,start-exec))  (4)
//end
```

(1) Unter der privilegierten Benutzerkennung TSOS wird ein Prozess gestartet.

(2) SDF-A wird geladen und gestartet.

(3) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird eröffnet und dabei neu angelegt. Standardmäßig ist die aktivierte Systemsyntaxdatei als Referenzdatei zugewiesen. Die in der Referenzdatei stehenden Kommandodefinitionen können in der eröffneten Gruppensyntaxdatei verändert werden.

(4) Die Kommandos LOAD, EXECUTE, LOAD-, START-PROGRAM, LOAD- und START-EXECUTABLE-PROGRAM werden gesperrt.

```
/mod-file-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all    (5)
/mod-user example,profile-id=user1 ──────────────────────────────────────  (6)
/mod-sdf-parameters scope=*permanent,
         syntax-file-type=*group(sys.sdf.group.syntax.example,user1) ──   (7)
%  CMD0681 SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' INSERTED IN
 PARAMETER FILE '$.SYSPAR.SDF'
%  CMD0718 GROUP SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' HAS BEEN
 ASSOCIATED WITH 'PROFILE-ID USER1' IN MEMORY TABLES
/exit-job
```

(5) Die Datei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird als mehrbenutzbar erklärt. Es ist nur Lesezugriff auf sie erlaubt.

(6) Die PROFILE-ID USER1 wird der Benutzerkennung EXAMPLE zugeordnet.

(7) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird der PROFILE-ID USER1 zugewiesen. Diese Zuweisung wird permanent in der SDF-Parameterdatei abgespeichert.

```
/set-logon-parameters example ──────────────────────────────────────────  (8)
/show-sdf-options ──────────────────────────────────────────────────────  (9)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : UNDEFINED
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *EXPERT
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
```

```
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
/start-prog $edt ------------------------------------------------------------ (10)
%  CMD0086 OPERATION NAME 'START-PROG' REMOVED BY USER
/start-edt ------------------------------------------------------------------ (11)
%  CMD0086 OPERATION NAME 'START-PROGRAM' REMOVED BY USER
/load-prog $edt
%  CMD0086 OPERATION NAME 'LOAD-PROG' REMOVED BY USER
/exec $edt ------------------------------------------------------------------ (12)
%  SDP0222 OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR 'SDP0116'. IN SYSTEM
MODE: /HELP-MSG SDP0116
/load $edt
%  CMD0187 ABBREVIATION OF OPERATION NAME 'LOAD' AMBIGUOUS WITH REGARD TO
'LOAD-ALIAS-CATALOG,LOAD-LOCAL-SUBSYSTEM-CATALOG'
/exit-job
 .
 .
```

(8) Unter der Benutzerkennung EXAMPLE wird ein Prozess gestartet.

(9) Die aktivierten Syntaxdateien werden angezeigt.
Die zuvor von der privilegierten Benutzerkennung TSOS bearbeitete Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE ist aktiviert.

(10) SDF akzeptiert das Kommando START-PROG nicht.

(11) SDF akzeptiert das Kommando START-EDT ebenfalls nicht, da START-EDT eine Prozedur aufruft, die wiederum das Kommando START-PROGRAM aufruft.

(12) Da das Kommando EXEC entfernt wurde, interpretiert SDF die Benutzereingabe als das SDF-P-Kommando EXEC-CMD und weist es wegen fehlerhafter Syntax ab.

3.1.2 Beispiel 2: Default-Wert ändern

Standardmäßig wird beim Anlegen einer neuen Datei mit dem Kommando CREATE-FILE vereinbart, dass von fremden Benutzerkennungen auf die Datei nicht zugegriffen werden darf (USER-ACCESS = *OWNER-ONLY*). Für Dateien, die unter der Benutzerkennung EXAMPLE eingerichtet werden, soll aber standardmäßig der Zugriff von fremden Benutzerkennungen möglich sein.

Das lässt sich dadurch erreichen, dass für die Benutzerkennung EXAMPLE eine Benutzersyntaxdatei eingerichtet wird, in der das Kommando CREATE-FILE entsprechend verändert wird.

Hinweis

Default-Werte können auch sehr flexibel (d.h. ohne Modifizierung von Syntaxdateien) als task-spezifische Default-Werte definiert werden. Task-spezifische Default-Werte werden jedoch nur bei der Eingabe im interaktiven Dialog ausgewertet. Für das nachfolgende Beispiel könnte der Default-Wert auch task-spezifisch vereinbart werden:

```
/!create-file user-access=*all-users
```

Weitere Einzelheiten zu task-spezifischen Default-Werten siehe Handbuch „Einführung in die Dialogschnittstelle“ [1].

```
/set-logon-parameters example,...  ————————————————————————————  (1)
 .
 .
/create-file demo.1  ——————————————————————————————————————————  (2)
/show-f-attr demo.1,inf=*par(security=*yes)  ——————————————————  (3)
%0000000003 :2OSG:$EXAMPLE.DEMO.1
%  ------------------------------ SECURITY     ----------------------------
%  READ-PASS  = NONE        WRITE-PASS = NONE        EXEC-PASS  = NONE
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
%  AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = NONE
%  DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME = NONE
%  SP-REL-LOCK= NO
%:2OSG: PUBLIC:      1 FILE  RES=         3 FRE=        3 REL=        3 PAGES
 .
 .
/start-sdf-a  ————————————————————————————————————————————————  (4)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sdf.user.syntax,,*crea  ——————————————————————  (5)
```

(1) Unter der Benutzerkennung EXAMPLE wird ein Prozess gestartet.

(2) Mit dem Kommando CREATE-FILE wird ein Benutzerkatalogeintrag für die Datei DEMO.1 eingerichtet.

(3) Die Schutzmerkmale der Datei DEMO.1 werden ausgegeben. Sie ist nicht mehrbenutzbar (USER-ACC = *OWNER-ONLY*).

(4) SDF-A wird geladen und gestartet.

(5) Die Benutzersyntaxdatei SDF.USER.SYNTAX wird eröffnet und dabei neu angelegt. Wenn eine unter der Benutzerkennung EXAMPLE katalogisierte Benutzersyntaxdatei diesen Namen hat, wird sie für Tasks der Benutzerkennung EXAMPLE bei der LOGON-Verarbeitung automatisch aktiviert. Standardmäßig sind die aktivierte Systemsyntaxdatei und die aktivierte Gruppensyntaxdatei als Referenzdateien zugewiesen. Kommandodefinitionen, die in den Referenzdateien stehen, können in der eröffneten Benutzersyntaxdatei verändert werden.

```
//show *oper(prot,orig=*com(create-file)),siz=*max ————————————————— (6)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          USER-ACCESS = *BY-PROTECTION-ATTR
              *BY-PROTECTION-ATTR or *OWNER-ONLY or *ALL-USERS or
               *SPECIAL
              Specifies whether external user IDs may access the file
          BASIC-ACL = *BY-PROTECTION-ATTR
             .
             .
             .
//edit *oper(prot,orig=*com(create-file)) ——————————————————————— (7)
//mod-oper def='PARAMETERS' ————————————————————————————————— (8)
//edit *oper(user-acc) ———————————————————————————————————————— (9)
//mod-oper def='ALL-USERS' ————————————————————————————————— (10)
```

(6) Der Operand PROTECTION des Kommandos CREATE-FILE wird in der ausführlichsten Form ausgegeben.

(7) Es wird auf den Operanden PROTECTION des Kommandos CREATE-FILE positioniert, d.h. der Operand PROTECTION wird aktuelles Objekt der bearbeiteten Benutzersyntaxdatei SDF.USER.SYNTAX.

(8) Der Operand, der aktuelles Objekt ist (PROTECTION), soll PARAMETERS als Default-Wert haben. Diese Änderung ist erforderlich, damit die folgende Änderung (siehe Arbeitsschritt 9 und 10) auch dann wirksam wird, wenn der struktureinleitende Wert PARAMETERS nicht explizit angegeben wird. Der Wert *STD enthält

implizit die Festlegung USER-ACCESS=*BY-PROTECTION-ATTR (entspricht USER-ACCESS= *OWNER-ONLY, wenn mit SECOS keine andere Default-Protection vereinbart ist).

(9) Es wird auf den Operanden USER-ACCESS des gerade bearbeiteten Kommandos CREATE-FILE positioniert, d.h. dieser Operand wird aktuelles Objekt der eröffneten Benutzersyntaxdatei SDF.USER.SYNTAX.

(10) Der Operand, der aktuelles Objekt ist (USER-ACCESS), soll ALL-USERS als Default-Wert haben.

```
//show *oper(prot,orig=*com(create-file)),siz=*max ———————————————— (11)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          USER-ACCESS = *ALL-USERS
              *BY-PROTECTION-ATTR or *OWNER-ONLY or *ALL-USERS or
               *SPECIAL
              Specifies whether external user IDs may access the file
          BASIC-ACL = *BY-PROTECTION-ATTR
             .
             .
             .
//end
/mod-sdf-opt synt-file=*add(*std) ———————————————————————————— (12)
/create-file demo.2 ——————————————————————————————————————— (13)
/show-f-attr demo.2,inf=*par(security=*yes) ——————————————————— (14)
%0000000003 :2OSG:$EXAMPLE.DEMO.2
%  ------------------------------ SECURITY     ----------------------------
%  READ-PASS  = NONE        WRITE-PASS = NONE        EXEC-PASS  = NONE
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
%  AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = NONE
%  DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME = NONE
%  SP-REL-LOCK= NO
%:2OSG: PUBLIC:      1 FILE  RES=         3 FRE=        3 REL=        3 PAGES
 .
 .
/exit-job
```

(11) Der Operand PROTECTION des gerade bearbeiteten Kommandos CREATE-FILE wird in der ausführlichsten Form ausgegeben.

(12) Die Benutzersyntaxdatei $EXAMPLE.SDF.USER.SYNTAX wird aktiviert.

(13) Mit dem Kommando CREATE-FILE wird ein Benutzerkatalogeintrag für die Datei DEMO.2 eingerichtet.

(14) Die Schutzmerkmale der Datei DEMO.2 werden ausgegeben. Sie ist mehrbenutzbar (USER-ACC = *ALL-USERS*).

3.1.3 Beispiel 3: Dateischutz durch vierstellige Kennwörter erzwingen

Die Benutzer der Benutzerkennungen EXAMPLE und EXAMP1 sollen aus Gründen des Datenschutzes gezwungen werden, ihre Dateien mit vier Byte langen Kennwörtern zu schützen. Das lässt sich erreichen, indem entsprechende Einschränkungen in einer Gruppensyntaxdatei definiert werden, die diesen Benutzerkennungen zugeordnet wird.

Die Definitionen der Kommandos CREATE-FILE, CREATE-FILE-GROUP, MODIFY-FILE-ATTRIBUTES und MODIFY-FILE-GROUP-ATTRIBUTES sind in der Gruppensyntaxdatei entsprechend zu verändern. Die Definitionen der alten Kommandos CATALOG und FILE lassen sich nicht so verändern. Diese Kommandos müssen deshalb gesperrt werden. Da sie jedoch mit dem Makro CMD aufgerufen werden können, hätte ein generelles Sperren unvorhersehbare Folgen. Außerdem sollen sie für den Stapelbetrieb weiterhin zugelassen sein. Die Kommandos werden deshalb lediglich für den Dialogbetrieb gesperrt.

```
/set-logon-parameters sdfusr,... ——————————————————————————— (1)
 .
 .
/start-sdf-a ——————————————————————————————————————————————— (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,group,*crea ———— (3)
//edit *command(catalog) ———————————————————————————————————— (4)
//modify-cmd dial-allow=*n,dial-proc-allow=*n ——————————————— (5)
//edit *command(file) ———————————————————————————————————————— (6)
//modify-cmd dial-allow=*n,dial-proc-allow=*n
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) SDF-A wird geladen und gestartet.

(3) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird eröffnet und dabei neu angelegt. Für die folgende Bearbeitung wird als Referenzdatei standardmäßig die aktivierte Systemsyntaxdatei zugewiesen.
Würde man keine Referenzdatei zuweisen, so müsste die folgende Bearbeitung etwas anders durchgeführt werden.

(4) Es wird auf das Kommando CATALOG positioniert, d.h. dieses Kommando wird aktuelles Objekt.

(5) Das Kommando, das aktuelles Objekt ist (CATALOG), wird für den Dialogbetrieb gesperrt. Auch innerhalb von Prozeduren, die im Dialogbetrieb ablaufen, ist es nicht zugelassen.

(6) Das Kommando FILE wird aktuelles Objekt und anschließend für den Dialogbetrieb gesperrt.

```
//show *oper(prot,orig=*com(create-file)),siz=*max ———————————————— (7)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          WRITE-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                write access
          READ-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                read access
          EXEC-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                execution
          DESTROY-BY-DELETE = *BY-PROTECTION-ATTR
               *BY-PROTECTION-ATTR or *NO or *YES
             .
             .
             .
//edit *oper(prot,orig=*com(create-file)) ———————————————————————— (8)
//modify-oper default='PARAMETERS' ———————————————————————————————— (9)
```

(7) Der Operand PROTECTION des Kommandos CREATE-FILE wird in der ausführlichsten Form ausgegeben.

(8) Es wird auf den Operanden PROTECTION des Kommandos CREATE-FILE positioniert, d.h. dieser Operand wird aktuelles Objekt der eröffneten Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE.

(9) Der Operand, der aktuelles Objekt ist, soll den Default-Wert PARAMETERS haben. STD ist nicht mehr Default-Wert des Operanden PROTECTION. Anschließend wird der erste für PROTECTION definierte Operandenwert aktuelles Objekt. Das ist die Eingabealternative STD.

```
//remove *value ——————————————————————————————————————— (10)
//edit *oper(write-pass) ——————————————————————————————— (11)
//modify-oper default=*n,struct-impl=*y ———————————————— (12)
//remove *value ——————————————————————————————————————— (13)
//edit *value(write-pass,*c-string) ———————————————————— (14)
//modify-value *c-string(short-l=4,long-l=4) ——————————— (15)
```

(10) Die Definition des Operandenwerts, der aktuelles Objekt ist (*STD), wird gelöscht.

(11) Es wird auf den Operanden WRITE-PASSWORD des Kommandos CREATE-FILE positioniert, d.h. dieser Operand wird aktuelles Objekt der eröffneten Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE. Der Operand WRITE-PASSWORD steht in einer Struktur. Da sein Name kommandoglobal eindeutig ist, brauchen der Name des übergeordneten Operanden PROTECTION und der Struktureinleiter PARAMETERS nicht angegeben zu werden. Die explizite Angabe von CREATE-FILE ist nicht erforderlich, da SDF-A auf Grund der vorherigen Anweisungen standardmäßig das Kommando CREATE-FILE nimmt.

(12) Der Operand, der aktuelles Objekt ist, soll keinen Default-Wert haben. Der bisherige Default-Wert *BY-PROT-ATTR-OR-NONE (entspricht *NONE, wenn mit SECOS keine andere Default-Protection vereinbart ist) ist nun nicht mehr Default-Wert des Operanden WRITE-PASSWORD. Wird bei der Kommandoeingabe jetzt der Operand WRITE-PASSWORD angegeben, dann wird implizit die Struktur PARAMETERS ausgewählt. Anschließend wird der erste für WRITE-PASSWORD definierte Operandenwert aktuelles Objekt. Das ist die Eingabealternative vom Typ NONE.

(13) Die Definition des Operandenwerts, der aktuelles Objekt ist (*BY-PROT-ATTR-OR-NONE), wird gelöscht.

(14) Es wird auf die Eingabealternative vom Typ C-STRING des Operanden WRITE-PASSWORD im Kommando CREATE-FILE positioniert, d.h. dieser Operandenwert wird in der bearbeiteten Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE aktuelles Objekt.
Der Operand WRITE-PASSWORD steht in einer Struktur. Da sein Name kommandoglobal eindeutig ist, brauchen der Name des übergeordneten Operanden PROTECTION und der Struktureinleiter PARAMETERS nicht angegeben zu werden. Die explizite Angabe von CREATE-FILE ist nicht erforderlich, da SDF-A auf Grund der vorherigen Anweisungen standardmäßig das Kommando CREATE-FILE nimmt.

(15) Für den Operandenwert, der aktuelles Objekt ist, wird definiert, dass er vom Typ C-STRING ist und sowohl seine Mindestlänge als auch seine Maximallänge vier Byte ist. Anschließend wird der nächste für WRITE-PASSWORD definierte Operandenwert aktuelles Objekt. Das ist die Eingabealternative vom Typ X-STRING.

```
//modify-value *x-string(short-l=4,long-l=4) ------------------------------ (16)
//remove *value ------------------------------------------------------------ (17)
//edit *oper(read-pass) ---------------------------------------------------- (18)
//modify-oper default=*n,struct-impl=*y
//remove *value
//edit *value(read-pass,*c-string)
//modify-value *c-string(4,4)
//modify-value *x-string(4,4)
//remove *value
//edit *oper(exec-pass) ---------------------------------------------------- (19)
//modify-oper default=*n,struct-impl=*y
//remove *value
//edit *value(exec-pass,*c-string)
//modify-value *c-string(4,4)
//modify-value *x-string(4,4)
//remove *value
```

(16) Für den Operandenwert, der aktuelles Objekt ist, wird definiert, dass er vom Typ X-STRING ist und sowohl seine Mindestlänge als auch seine Maximallänge vier Byte ist. Anschließend wird der nächste für WRITE-PASSWORD definierte Operandenwert aktuelles Objekt. Das ist die Eingabealternative vom Typ INTEGER.

(17) Die Definition des Operandenwerts, der aktuelles Objekt ist (INTEGER), wird gelöscht.

(18) Der Operand READ-PASSWORD und seine Werte werden genauso geändert wie zuvor der Operand WRITE-PASSWORD (siehe 12.-17.).

(19) Der Operand EXEC-PASSWORD und seine Werte werden genauso geändert wie zuvor der Operand WRITE-PASSWORD (siehe 12.-17.).

```
//show *oper(prot,orig=*com(create-file)),siz=*max ———————————————— (20)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          WRITE-PASSWORD =
               *NONE or c-string_4..4 or x-string_7..8 or *SECRET
              Specifies the password for protection against unauthorized
               write access
          READ-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
              x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
              Specifies the password for protection against unauthorized
               read access
          EXEC-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
              x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
              Specifies the password for protection against unauthorized
               execution
          DESTROY-BY-DELETE = *BY-PROTECTION-ATTR
               *BY-PROTECTION-ATTR or *NO or *YES
             .
             .
             .
 .
 . ———————————————————————————————————————————————————————————————— (21)
 .
//end
/mod-file-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all   (22)
/exit-job
 .
 .
```

(20) Der Operand PROTECTION des Kommandos CREATE-FILE wird in der ausführlichsten Form ausgegeben.

(21) Die Definitionen der Kommandos CREATE-FILE-GROUP, MODIFY-FILE-ATTRIBUTES und MODIFY-FILE-GROUP-ATTRIBUTES werden genauso geändert wie zuvor die des Kommandos CREATE-FILE.

(22) Die Datei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird als mehrbenutzbar erklärt. Es ist nur Lesezugriff auf sie erlaubt.

```
/set-logon-parameters tsos,...  ——————————————————————————— (23)
/copy-file from-file=$sdfusr.sys.sdf.group.syntax.example,to-file=-
/sys.sdf.group.syntax.example,prot=*same  ————————————————————— (24)
/modify-user example,profile-id=user1  ———————————————————————— (25)
/modify-sdf-param scope=*temporary,syntax-file=*group-
/(sys.sdf.group.syntax.example,user1)  ———————————————————————— (26)
/modify-user examp1,profile-id=user1  ————————————————————————— (27)
/exit-job
 .
 .
```

(23) Unter der privilegierten Benutzerkennung TSOS wird ein Prozess gestartet.

(24) Die Gruppensyntaxdatei $SDUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE, die zuvor unter der Benutzerkennung SDFUSR erstellt wurde, wird kopiert. Der Name der Kopie ist $TSOS.SYS.SDF.GROUP.SYNTAX.EXAMPLE. Sie hat die gleichen Schutzmerkmale wie die Originaldatei.

(25) Die PROFILE-ID USER1 wird der Benutzerkennung EXAMPLE zugewiesen.

(26) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird der PROFILE-ID USER1 zugewiesen.

(27) Die PROFILE-ID USER1 wird der Benutzerkennung EXAMP1 zugewiesen.

```
/set-logon-parameters example ————————————————————————————— (28)
/show-sdf-options ————————————————————————————————————————— (29)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : UNDEFINED
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *EXPERT
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
/catalog demo ————————————————————————————————————————————— (30)
%  CMD0087 OPERATION NAME 'CATALOG' IS NOT PERMITTED AT THE MOMENT
/file demo ———————————————————————————————————————————————— (31)
%  CMD0087 OPERATION NAME 'FILE' IS NOT PERMITTED AT THE MOMENT
```

(28) Unter der Benutzerkennung EXAMPLE wird ein Prozess gestartet.

(29) Die aktivierten Syntaxdateien werden angezeigt. Die Gruppensyntaxdatei $.SYS.SDF.GROUP.SYNTAX.EXAMPLE ist aktiviert.

(30) SDF akzeptiert das Kommando CATALOG nicht. Weil es für den Dialogbetrieb gesperrt ist, wird es als unbekannt behandelt.

(31) SDF akzeptiert das Kommando FILE nicht. Weil es für den Dialogbetrieb gesperrt ist, wird es als unbekannt behandelt.

```
/create-file demo,wr-pass=2,ex-pass='3' ------------------------------- (32)
%  CMD0051 INVALID OPERAND 'PROTECTION=PARAMETERS:WRITE-PASSWORD'
%  CMD0064 OPERAND VALUE 'P' DOES NOT MATCH DATA TYPE 'C-STRING_4..4 OR
 X-STRING_7..8 OR SECRET'
%  CMD0051 INVALID OPERAND 'PROTECTION=PARAMETERS:READ-PASSWORD'
%  CMD0099 MANDATORY OPERAND MISSING OR INVALID
%  CMD0051 INVALID OPERAND 'PROTECTION=PARAMETERS:EXEC-PASSWORD'
%  CMD0062 LENGTH OF VALUE ''P'' NOT IN PERMISSIBLE RANGE FOR DATA TYPE
 'C-STRING_4..4'
/create-file demo,read-pass='1234',wr-pass='2345',ex-pass='3456' ------- (33)
/show-file-attr demo,inf=*par(security=*yes) -------------------------- (34)
%0000000003 :2OSG:$EXAMPLE.DEMO
%  ------------------------------- SECURITY     -----------------------------
%  READ-PASS  = YES         WRITE-PASS = YES         EXEC-PASS  = YES
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
%  AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = NONE
%  DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME = NONE
%  SP-REL-LOCK= NO
%:2OSG: PUBLIC:      1 FILE  RES=         3 FRE=        3 REL=        3 PAGES
 .
 .
/exit-job
```

(32) Aus folgenden Gründen akzeptiert SDF das Kommando CREATE-FILE nicht:
- Das angegebene Schreibkennwort ist weder das Schlüsselwort SECRET noch vom Typ C-STRING oder X-STRING ist.
- Es ist kein Lesekennwort angegeben.
- Das angegebene Ausführungskennwort vom Typ C-STRING nur ein Byte lang ist.

(33) SDF akzeptiert das Kommando CREATE-FILE, weil vier Byte lange Kennwörter vom Typ C-STRING angegeben sind.

(34) Für die Datei DEMO wurde ein Katalogeintrag eingerichtet. Sie ist durch Kennwörter geschützt.

3.1.4 Beispiel 4: Nur ein Programm (EDT) zulassen

Die Benutzer der Benutzerkennung EXAMPLE sollen als einziges Programm nur den unter der privilegierten Benutzerkennung TSOS katalogisierten Dateibearbeiter EDT laden und starten dürfen.

Über die Gruppensyntaxdatei $TSOS.SYS.SDF.GROUP.SYNTAX.EXAMPLE lässt sich die gewünschte Einschränkung verwirklichen. In ihr sind die Definitionen der Kommandos START-PROGRAM und LOAD-PROGRAM entsprechend zu verändern.

– Das Kommando START-PROGRAM soll in START-EDITOR umbenannt werden und keine sichtbaren Operanden haben. Das Kommando START-EDITOR darf in der Syntaxdatei-Hierarchie noch nicht definiert sein.

– Das Kommando LOAD-PROGRAM soll seinen Namen behalten. Mit Ausnahme des Operanden FROM-FILE sollen seine Operanden ausgeblendet werden.

– Ab BLSSERV V2.3 wird die Funktionalität von START- und LOAD-PROGRAM zusätzlich mit verbesserter Syntax über die neuen Kommandos START- und LOAD-EXECUTABLE-PROGRAM angeboten. In diesem Fall müssen die beiden neuen Kommandos gesperrt oder ihre Syntaxdefinition analog zu START- und LOAD-PROGRAM angepasst werden.

– Damit die definierte Einschränkung nicht umgangen werden kann, werden die alten Kommandos EXECUTE und LOAD sowie die START-Kommandos anderer Programme gesperrt.

```
/set-logon-parameters tsos,...  ————————————————————————————  (1)
 .
 .
/start-sdf-a  ————————————————————————————————————————————————  (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,*group,*crea  ————————  (3)
//remove *command((load,exec))  ————————————————————————————  (4)
//show *command(start-prog)  ————————————————————————————  (5)
```

(1) Unter der privilegierten Benutzerkennung TSOS wird ein Prozess gestartet.

(2) SDF-A wird geladen und gestartet.

(3) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird eröffnet und dabei neu angelegt. Standardmäßig ist die aktivierte Systemsyntaxdatei als Referenzdatei zugewiesen. Die in der Referenzdatei stehenden Kommandodefinitionen können in der eröffneten Gruppensyntaxdatei verändert werden.

(4) Die Kommandos LOAD und EXECUTE werden gesperrt.

(5) Das Kommando START-PROGRAM wird ausgegeben. Standardmäßig hat die Ausgabe den Umfang der Stufe MINIMUM:

```
START-PROGRAM(SRPG,SR,START-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
     FROM-FILE =
          STRUCTURE: *MODULE
               LIBRARY = *DBL-DEFAULT
                    STRUCTURE: *LINK
                         LINK =
               ELEMENT-OR-SYMBOL(ELEMENT,ELEM) = *ALL
                    STRUCTURE: composed-name
                         VERSION = *STD
                    STRUCTURE: c-string
                         VERSION = *STD
               PROGRAM-MODE = *DBL-DEFAULT
               RUN-MODE = *DBL-DEFAULT
                    STRUCTURE: *ADVANCED
                         ALTERNATE-LIBRARIES = *DBL-DEFAULT
                         NAME-COLLISION = *DBL-DEFAULT
                         UNRESOLVED-EXTRNS = *DBL-DEFAULT
                         ERROR-EXIT = *DBL-DEFAULT
                         MESSAGE-CONTROL = *DBL-DEFAULT
                         LOAD-INFORMATION = *DBL-DEFAULT
                         PROGRAM-MAP = *DBL-DEFAULT
                              STRUCTURE: *SYSLST
                                   SYSLST-NUMBER = *STD
                              STRUCTURE: *BOTH
                                   SYSLST-NUMBER = *STD
                         SHARE-SCOPE = *DBL-DEFAULT
                              STRUCTURE: *MEMORY-POOL
                                   SCOPE = *ALL
                         IGNORE-ATTRIBUTES = *DBL-DEFAULT
                         REP-FILE = *DBL-DEFAULT
                         AUTOLINK = *DBL-DEFAULT
                         PROGRAM-VERSION = *DBL-DEFAULT
          STRUCTURE: *PHASE
               LIBRARY =
               ELEMENT =
               VERSION = *STD
     CPU-LIMIT = *JOB-REST
     TEST-OPTIONS = *DBL-DEFAULT
     MONJV = *NONE
     RESIDENT-PAGES = *PARAMETERS
          STRUCTURE: *PARAMETERS
               MINIMUM = *STD
               MAXIMUM = *STD
     VIRTUAL-PAGES = *STD
```

```
//show *operand(from-f,orig=*com(start-prog)),siz=*med,att-inf=*n ———— (6)
FROM-FILE =
     filename or *MODULE() or *PHASE()
//edit *oper(from-f,orig=*com(start-prog)) ———————————————————————— (7)
//mod-oper def='$edt',pres=*intern ———————————————————————————————— (8)
//edit *oper(cpu-lim) ————————————————————————————————————————————— (9)
//mod-oper pres=*intern
//edit *oper(test-opt)
//mod-oper pres=*intern
//edit *oper(monjv)
//mod-oper pres=*intern
//edit *oper(resid-p)
//mod-oper pres=*intern
//edit *oper(resid-p,par,min) ————————————————————————————————————— (10)
//mod-oper pres=*intern
//edit *oper(resid-p,par,max)
//mod-oper pres=*intern
//edit *oper(virt-p)
//mod-oper pres=*intern
```

(6) Der Operand FROM-FILE des Kommandos START-PROGRAM wird in mittlerem Umfang ausgegeben. Die Ausgabe der Strukturen, die an den Werten MODULE und PHASE hängen, wird unterdrückt.

(7) Es wird auf den Operanden FROM-FILE des Kommandos START-PROGRAM positioniert. Dieser Operand wird aktuelles Objekt in der geöffneten Syntaxdatei.

(8) Der Operand, der aktuelles Objekt ist (FROM-FILE), erhält den Default-Wert „$EDT“. Der Operand wird an der Benutzeroberfäche des Kommandos ausgeblendet. Voraussetzung für das Ausblenden ist, dass der Operand einen Default-Wert erhält.

(9) Die übrigen Operanden des Kommandos START-PROGRAM werden ebenfalls ausgeblendet. Beim Positionieren ist es nicht erforderlich, den Kommandonamen anzugeben, da die in Arbeitsschritt 7 gemachte Angabe noch wirksam ist.

(10) Der ausgeblendete Operand RESIDENT-PAGES hat einen Default-Wert, an dem eine Struktur hängt. Die in dieser Struktur stehenden Operanden müssen ebenfalls ausgeblendet werden.

```
//show *com(start-prog),att-inf=*n,size=*max ————————————————————— (11)
START-PROGRAM(SRPG,SR,START-PROG)
     Loads a program (load or object module) to the memory and starts it
//edit *com(start-prog) ———————————————————————————————————————— (12)
//mod-cmd start-editor,help=(e('Loads the EDT to the memory and starts -
//it.'),d('Laedt den EDT in den Speicher und startet ihn.')) ————— (13)
//show *com(start-editor) ——————————————————————————————————————— (14)
```

(11) Der Kopf des Kommandos START-PROGRAM wird mit dem zugehörigen Hilfetext ausgegeben.

(12) Es wird auf das Kommando START-PROGRAM positioniert.

(13) Der Name des Kommandos, das aktuelles Objekt ist (START-PROGRAM), wird in START-EDITOR geändert. Die Hilfetexte werden entsprechend des eingeschränkten Funktionsumfangs verändert. Falls in Ihrem System bereits ein Kommando START-EDITOR definiert wurde, wird die Anweisung abgewiesen.

(14) Das Kommando START-EDITOR wird ausgegeben (siehe unten). Die Ausgabe hat den Umfang der Stufe MINIMUM. Als Standardnamen hat das Kommando seinen früheren Namen START-PROGRAM sowie einige Kurznamen. Die Operanden der Strukturen FROM-FILE=*MODULE(...) und FROM-FILE=*PHASE(...) werden in SDF-A noch angezeigt, da sie mit PRESENCE=*NORMAL definiert sind. An der Benutzeroberfläche sind diese Operanden aber nicht mehr sichtbar und sie können auch nicht mehr eingegeben werden.

```
START-EDITOR(SRPG,SR,START-PROG,START-PROGRAM)
   FROM SYS.SDF.GROUP.EXAMPLE (GROUP)
              LIBRARY = *STD
                   STRUCTURE: *LINK
                        LINK =
              ELEMENT = *ALL
                   STRUCTURE: filename
                        VERSION = *STD
                   STRUCTURE: c-string
                        VERSION = *STD
              PROGRAM-MODE = *24
              RUN-MODE = *STD
                   STRUCTURE: *ADVANCED
                        ALTERNATE-LIBRARIES = *NO
                        NAME-COLLISION = *STD
                        UNRESOLVED-EXTRNS = *STD
                        ERROR-EXIT = X'FFFFFFFF'
                        MESSAGE-CONTROL = *INFORMATION
                        LOAD-INFORMATION = *DEFINITIONS
                        PROGRAM-MAP = *NO
                             STRUCTURE: *SYSLST
```

```
                                        SYSLST-NUMBER = *STD
                                   STRUCTURE: *BOTH
                                        SYSLST-NUMBER = STD
                              SHARE-SCOPE = *SYSTEM-MEMORY
                                   STRUCTURE: *MEMORY-POOL
                                        SCOPE = *ALL
                              IGNORE-ATTRIBUTES = *NONE
                              REP-FILE = *NONE
                              AUTOLINK = *YES
                         LIBRARY =
                         ELEMENT =
                         VERSION = *STD
//remove *com((start-archive,start-binder,start-hsms,...)) ------------ (15)
//show *com(load-prog) ------------------------------------------------- (16)
```

(15) Damit tatsächlich nur das Programm EDT gestartet werden kann, müssen alle anderen (noch funktionierenden) START-Kommandos gesperrt werden. Die START-Kommandos vieler Programme werden jedoch auch ohne Sperrung mit der Meldung

```
%  CMD0185 OPERAND NAME 'FROM-FILE' COULD NOT BE IDENTIFIED
```

abgewiesen.

(16) Das Kommando LOAD-PROGRAM wird ausgegeben. Die Ausgabe hat den Umfang der Stufe MINIMUM (siehe Seite 70).

```
LOAD-PROGRAM(LDPG,LOAD-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
     FROM-FILE =
          STRUCTURE: *MODULE
               LIBRARY = *DBL-DEFAULT
                    STRUCTURE: *LINK
                         LINK =
               ELEMENT-OR-SYMBOL(ELEMENT,ELEM) = *ALL
                    STRUCTURE: composed-name
                         VERSION = *STD
                    STRUCTURE: c-string
                         VERSION = *STD
               PROGRAM-MODE = *DBL-DEFAULT
               RUN-MODE = *DBL-DEFAULT
                    STRUCTURE: *ADVANCED
                         ALTERNATE-LIBRARIES = *DBL-DEFAULT
                         NAME-COLLISION = *DBL-DEFAULT
                         UNRESOLVED-EXTRNS = *DBL-DEFAULT
                         ERROR-EXIT = *DBL-DEFAULT
                         MESSAGE-CONTROL = *DBL-DEFAULT
                         LOAD-INFORMATION = *DBL-DEFAULT
                         PROGRAM-MAP = *DBL-DEFAULT
                              STRUCTURE: *SYSLST
                                   SYSLST-NUMBER = *STD
                              STRUCTURE: *BOTH
                                   SYSLST-NUMBER = *STD
                         SHARE-SCOPE = *DBL-DEFAULT
                              STRUCTURE: *MEMORY-POOL
                                   SCOPE = *ALL
                         IGNORE-ATTRIBUTES = *DBL-DEFAULT
                         REP-FILE = *DBL-DEFAULT
                         AUTOLINK = *DBL-DEFAULT
                         PROGRAM-VERSION = *DBL-DEFAULT
          STRUCTURE: *PHASE
               LIBRARY =
               ELEMENT =
               VERSION = *STD
     CPU-LIMIT = *JOB-REST
     TEST-OPTIONS = *DBL-DEFAULT
     MONJV = *NONE
     RESIDENT-PAGES = *PARAMETERS
          STRUCTURE: *PARAMETERS
               MINIMUM = *STD
               MAXIMUM = *STD
     VIRTUAL-PAGES = *STD
```

```
//show *oper(from-f,orig=*com(load-prog)),siz=*med,att-inf=*n ———————— (17)
FROM-FILE =
     filename or *MODULE() or *PHASE()
//edit *oper(from-f,orig=*com(load-prog)) ———————————————————————— (18)
//mod-oper def='$edt' ——————————————————————————————————————————— (19)
//mod-value value=('$edt','edt'(outp='$edt')) ———————————————————— (20)
//remove *value ————————————————————————————————————————————————— (21)
//remove *value(from-f,phase) ——————————————————————————————————— (22)
//edit *oper(cpu-lim) ——————————————————————————————————————————— (23)
//mod-oper pres=*intern
//edit *oper(test-opt)
//mod-oper pres=*intern
//edit *oper(monjv)
//mod-oper pres=*intern
//edit *oper(resid-p)
//mod-oper pres=*intern
//edit *oper(resid-p,par,min)
//mod-oper pres=*intern
//edit *oper(resid-p,par,max)
//mod-oper pres=*intern
```

(17) Der Operand FROM-FILE des Kommandos LOAD-PROGRAM wird in mittlerem Umfang ausgegeben. Die Ausgabe der Strukturen, die an den Werten *MODULE und *PHASE hängen, wird unterdrückt.

(18) Es wird auf den Operanden FROM-FILE im Kommando LOAD-PROGRAM positioniert. Dieser Operand wird aktuelles Objekt in der eröffneten Syntaxdatei.

(19) Der Operand, der aktuelles Objekt ist (FROM-FILE), erhält den Default-Wert $EDT. Danach ist die Definition des ersten zu FROM-FILE gehörenden Operandenwerts (FILENAME) aktuelles Objekt.

(20) Für die Operandenwertdefinition, die aktuelles Objekt ist (FILENAME), werden die Werte $EDT und EDT als einzig zulässig definiert. Für den Eingabewert EDT wird definiert, dass SDF nicht ihn, sondern den Wert $EDT an die Implementierung übergibt. Danach ist die Definition des nächsten zu FROM-FILE gehörenden Operandenwerts (MODULE) aktuelles Objekt.

(21) Die Operandenwertdefinition, die aktuelles Objekt ist (MODULE), wird gelöscht. **Achtung:** Danach ist die Definition des Operandenwerts FILENAME aktuelles Objekt.

(22) Die Operandenwertdefinition PHASE des Operanden FROM-FILE im Kommando LOAD-PROGRAM wird gelöscht. Dabei ist es nicht erforderlich, den Kommandonamen anzugeben, da die Angabe in Arbeitsschritt 18 noch wirksam ist.

(23) Die übrigen Operanden des Kommandos LOAD-PROGRAM werden ausgeblendet.

```
//edit *oper(virt-p)
//mod-oper pres=*intern
//show *com(load-prog),att-info=*y,siz=*max ──────────────────────────── (24)
LOAD-PROGRAM(LDPG,LOAD-PROG)
   FROM SYS.SDF.GROUP.EXAMPLE (GROUP)
     loads a program (load or  object module) to the memory
     FROM-FILE = $EDT
          $EDT or EDT
          name of the file containing the load  module or details of the
          object module/load module library
//end
/mod-f-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all ─── (25)
/mod-user example,profile-id=user1 ──────────────────────────────────── (26)
/mod-sdf-parameters scope=*permanent,syntax-file=*group-
/(sys.sdf.group.syntax.example,user1) ──────────────────────────────── (27)
%  CMD0681 SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' INSERTED IN
PARAMETER FILE '$.SYSPAR.SDF'
%  CMD0718 GROUP SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' HAS BEEN
ASSOCIATED WITH 'PROFILE-ID USER1' IN MEMORY TABLES
/start-prog $example.edt ──────────────────────────────────────────────── (28)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//end
/exit-job
```

(24) Das Kommando LOAD-PROGRAM wird ausgegeben. Die Ausgabe hat den Umfang der Stufe MAXIMUM. Es besitzt als einzigen an der Benutzeroberfläche sichtbaren Operanden den Operanden FROM-FILE. Dieser hat den Default-Wert $EDT. Die für ihn zugelassenen Werte sind $EDT und EDT.

(25) Die Datei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird als mehrbenutzbar erklärt. Es ist nur Lesezugriff auf sie erlaubt.

(26) Die PROFILE-ID USER1 wird der Benutzerkennung EXAMPLE zugeordnet.

(27) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird der PROFILE-ID USER1 zugewiesen. Diese Zuweisung wird permanent in der SDF-Parameterdatei abgespeichert.

(28) In der Datei $EXAMPLE.EDT steht das Programm SDF-A.
Die privilegierte Benutzerkennung TSOS darf auf diese Datei zugreifen, weil das Kommando START-PROGRAM für TSOS nicht eingeschränkt ist. Im Folgenden wird gezeigt, dass ein Benutzer unter der Benutzerkennung EXAMPLE selbst dann nicht auf ein Programm namens EDT zugreifen kann, wenn unter seiner eigenen Benutzerkennung dieses Programm existiert. Es ist auf Grund der oben definierten Einschränkung tatsächlich nur das Laden von $EDT möglich.

```
/set-logon-parameters example,... ──────────────────────────────────── (29)
/show-sdf-options ──────────────────────────────────────────────────── (30)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : UNDEFINED
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *EXPERT
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
/exec $sdf-a ──────────────────────────────────────────────────────────── (31)
%  SDP0222 OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR 'SDP0116'. IN SYSTEM
MODE: /HELP-MSG SDP0116
```

(29) Unter der Benutzerkennung EXAMPLE wird ein Prozess gestartet.

(30) Die aktivierten Syntaxdateien werden angezeigt. Die zuvor von der privilegierten Benutzerkennung TSOS bearbeitete Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE ist aktiviert.

(31) Da das Kommando EXEC entfernt wurde, interpretiert SDF die Benutzereingabe als das SDF-P-Kommando EXEC-CMD und weist es wegen fehlerhafter Syntax ab.

```
/load-program $sdf-a ──────────────────────────────────────────────────── (32)
%  CMD0051 INVALID OPERAND 'FROM-FILE'
%  CMD0063 OPERAND VALUE '$SDF-A' NOT A MEMBER OF THE SINGLE VALUE LIST OF
 SCOPE '$EDT OR EDT'
/load-program edt ──────────────────────────────────────────────────────── (33)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1996. ALL RIGHTS
RESERVED
/load-program ──────────────────────────────────────────────────────────── (34)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1996. ALL RIGHTS
RESERVED
/start-program $sdf-a ──────────────────────────────────────────────────── (35)
%  CMD0376 SPECIFICATION OF POSITIONAL OPERANDS UP FROM POSITION '1' NOT
 PERMITTED
/start-editor ──────────────────────────────────────────────────────────── (36)
%  BLS0500 PROGRAM 'EDT', VERSION '16.5A' OF '1994-09-02' LOADED
 .
 .
halt
%  EDT8000 EDT TERMINATED
/start-program ─────────────────────────────────────────────────────────── (37)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
.
 .
halt
%  EDT8000 EDT TERMINATED
/exit-job
```

(32) SDF akzeptiert im Kommando LOAD-PROGRAM den Operandenwert $SDF-A nicht.

(33) SDF übergibt an die Implementierung nicht den Wert EDT, sondern den Wert $EDT. Es wird nicht das in der Datei $EXAMPLE.EDT stehende Programm SDF-A (siehe Arbeitsschritt 28) geladen, sondern das Programm, das in der Datei $TSOS.EDT steht.

(34) SDF übergibt den Default-Wert $EDT des Operanden FROM-FILE an die Implementierung. Der EDT wird geladen.

(35) START-PROGRAM ist der Standardname des umbenannten Kommandos START-EDITOR. SDF erkennt das Kommando START-EDT und weist den Operandenwert $SDF-A als unzulässig zurück.

(36) Die Eingabe START-EDITOR bewirkt, dass der EDT geladen und gestartet wird.

(37) An seinem Standardnamen START-PROGRAM erkennt SDF das Kommando START-EDITOR. Der EDT wird geladen und gestartet.

3.1.5 Beispiel 5: Menge der benutzbaren Programme einschränken

Die Benutzer der Benutzerkennung EXAMPLE sollen nur Programme laden und starten dürfen, die unter der Benutzerkennung SDFUSR katalogisiert sind. Diese Programme stehen ausnahmslos in Dateien und nicht in Bibliotheken.

Über die Gruppensyntaxdatei $SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE lässt sich die gewünschte Einschränkung verwirklichen. Sie soll die Versionsnummer EXAMPLE#5 erhalten. In ihr sind die Definitionen der Kommandos START-PROGRAM und LOAD-PROGRAM entsprechend zu verändern. Da die Benutzer der Benutzerkennung EXAMPLE keine Programme testen, soll beim Verändern der beiden Kommandodefinitionen der Operand TEST.OPTIONS gesperrt werden.

Hinweis

Ab BLSSERV V2.3 wird die Funktionalität von START- und LOAD-PROGRAM zusätzlich mit verbesserter Syntax über die neuen Kommandos START- und LOAD-EXECUTABLE-PROGRAM angeboten. In diesem Fall müssen die beiden neuen Kommandos gesperrt oder ihre Syntaxdefinition analog zu START- und LOAD-PROGRAM angepasst werden.

Die Definitionen der alten Kommandos EXECUTE und LOAD lassen sich nicht in der gewünschten Art verändern. Würde man EXECUTE und LOAD nur für den Dialog- und Stapelbetrieb sperren, nicht aber für den Aufruf über den CMD-Makro, so ließe sich die Sperre leicht umgehen. Die Kommandos sind deshalb generell zu sperren.

Systemweit ist die Benutzerführung auf GUIDANCE=*EXPERT voreingestellt.
Für die Benutzerkennung EXAMPLE soll die Benutzerführung die Voreinstellung GUIDANCE=*NO haben. Unter der Benutzerkennung SDFUSR ist der EDT verfügbar.

```
/set-logon-parameters sdfusr,...  ————————————————————————————  (1)
 .
 .
 .
/start-sdf-a  ————————————————————————————————————————————————  (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,*group,*crea  ————————  (3)
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) SDF-A wird geladen und gestartet.

(3) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird eröffnet und dabei neu angelegt. Für die folgende Bearbeitung wird als Referenzdatei standardmäßig die aktivierte Systemsyntaxdatei zugewiesen.

```
//set-glob vers=example#5,guid=*n ————————————————————————————— (4)
//remove *com((load,exec)) ————————————————————————————————————— (5)
//show *com(start-prog),att-inf=*n,size=*max ——————————————————— (6)
START-PROGRAM(SRPG,SR,START-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
     Loads a program (load or object module) to the memory and starts it
//edit *com(start-prog) ————————————————————————————————————————— (7)
//mod-cmd help=(e('Loads a $SDFUSR-program to the memory and starts it.'),-
//d('Laedt ein $SDFUSR-Programm in den Speicher und startet es.')) ——— (8)
//show *oper(from-f),siz=*med,att-inf=*n ———————————————————————— (9)
FROM-FILE =
     filename or *MODULE() or *PHASE()
//show *oper(from-f),impl=*y,att-inf=*n ————————————————————————— (10)
ADD-OPERAND NAME=FROM-FILE,INTERNAL-NAME=FROMFI,STANDARD-NAME=          -
            FROM-FILE,HELP=(D(TEXT='Name der Datei, die das Lademodul-
 enthaelt oder Angaben zur Bindemodul- bzw. Lademodulbibliothek'),E(    -
            TEXT='name of the file containing the load module or-
 specification of the object module/load module library')),             -
            RESULT-OPERAND-NAME=*POSITION(POSITION=1),                  -
            CONCATENATION-POS=1
//add-oper prefix,def='$sdfusr.',res-oper-n=*pos(1),conc-pos=1,pres=*int (11)
```

(4) Die Gruppensyntaxdatei SYS.SDF.GROUP.SYNTAX.EXAMPLE erhält die Versionsnummer EXAMPLE#5. Die Benutzerführung ist mit GUIDANCE=*NO voreingestellt.

(5) Die Kommandos LOAD und EXECUTE werden generell gesperrt.

(6) Die SDF-A-Anweisung, mit der das Kommando START-PROGRAM definiert ist, wird ausgegeben.

(7) Es wird auf das Kommando START-PROGRAM positioniert, d.h. dieses Kommando wird aktuelles Objekt der eröffneten Gruppensyntaxdatei.

(8) Der Hilfetext des Kommandos, das aktuelles Objekt ist, wird geändert. Danach wird der erste Operand dieses Kommandos (FROM-FILE) aktuelles Objekt.

(9) Der Operand FROM-FILE des gerade bearbeiteten Kommandos START-PROGRAM wird in der mittleren Ausführlichkeitsstufe ausgegeben.

(10) Die SDF-A-Anweisung, mit der der Operand FROM-FILE des Kommandos START-PROGRAM definiert ist, wird ausgegeben.

(11) Der Operand PREFIX wird definiert. SDF-A fügt seine Definition nach dem aktuellen Objekt (FROM-FILE) in die Definition des Kommandos START-PROGRAM ein. An der Benutzeroberfläche des Kommandos START-PROGRAM ist er unsichtbar. Sein Default-Wert ist „$SDFUSR.“. Er hat bei der Übergabe an die Implementierung die gleiche Position wie der Operand FROM-FILE (siehe Arbeitsschritt 10) und steht bei der Verkettung mit ihm an erster Stelle.

```
//add-value *part-filename ——————————————————————————— (12)
//edit *oper(from-f) ————————————————————————————————— (13)
//mod-oper conc-pos=2,help=(e('specifies the name of the file holding the -
//load module.'),d('Name der Datei, die das Lademodul enthaelt.')) ——— (14)
//mod-value *filename(user-id=*n) ———————————————————— (15)
//remove *value ——————————————————————————————————— (16)
//remove *value(from-f,phase) ———————————————————————— (17)
//show *oper(test-opt) ——————————————————————————————— (18)
TEST-OPTIONS = *NONE
//edit *oper(test-opt) ——————————————————————————————— (19)
//mod-oper pres=*intern ——————————————————————————— (20)
```

(12) Für den Operanden PREFIX wird ein Operandenwert vom Typ PARTIAL-FILENAME definiert.

(13) Es wird auf den Operanden FROM-FILE des gerade bearbeiteten Kommandos START-PROGRAM positioniert, d.h. dieser Operand wird aktuelles Objekt.

(14) Der Operand, der aktuelles Objekt ist (FROM-FILE), soll bei einer Verkettung mit dem Operanden PREFIX an zweiter Stelle stehen. Die Hilfetexte für FROM-FILE werden verändert. Danach wird der erste Operandenwert (FILENAME) von FROM-FILE aktuelles Objekt.

(15) Die Definition des Operandenwerts, der aktuelles Objekt ist (FILENAME), wird verändert. Die Angabe der Benutzerkennung als Teil des Dateinamens ist nicht mehr zulässig. Danach wird der Operandenwert MODULE aktuelles Objekt.

(16) Die Definition des Operandenwerts, der aktuelles Objekt ist (MODULE), wird gelöscht. Die an ihm hängende Struktur wird ebenfalls gelöscht.
Achtung:
Danach ist die Definition des Operandenwerts FILENAME aktuelles Objekt.

(17) Die Definition des zum Operanden FROM-FILE gehörenden Operandenwerts PHASE wird gelöscht.

(18) Der Operand TEST-OPTIONS wird ausgegeben.

(19) Es wird auf den Operanden TEST-OPTIONS des gerade bearbeiteten Kommandos START-PROGRAM positioniert, d.h. TEST-OPTIONS wird aktuelles Objekt.

(20) Der Operand, der aktuelles Objekt ist (TEST-OPTIONS), wird an der Benutzeroberfläche des Kommandos START-PROGRAM ausgeblendet.

```
//show *com(load-prog),att-inf=*n,size=*max ──────────────────────────── (21)
LOAD-PROGRAM(LDPG,LOAD-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
    Loads a program (load or object module) to the memory
//edit *com(load-prog)
//mod-cmd help=(e('Loads a $SDFUSR-program to the memory.'),-
//d('Laedt ein $SDFUSR-Programm in den Speicher.'))
//show *oper(from-f),siz=*med,att-inf=*n
FROM-FILE =
     filename or *MODULE() or *PHASE()
//show *oper(from-f),impl=*y,att-inf=*n
ADD-OPERAND NAME=FROM-FILE,INTERNAL-NAME=FROMFI,STANDARD-NAME=         -
            FROM-FILE,HELP=(D(TEXT='Name der Datei, die das Lademodul-
 enthaelt oder Angaben zur Bindemodul- bzw. Lademodulbibliothek'),E(   -
            TEXT='name of the file containing the load  module or-
 specification of the object module/load module library')),            -
            RESULT-OPERAND-NAME=*POSITION(POSITION=1),                 -
            CONCATENATION-POS=1
//copy *oper(prefix,orig=*com(start-prog)) ──────────────────────────── (22)
//edit *oper(from-f)
//mod-oper conc-pos=2,help=(e('specifies the name of the file holding -
//the load module.'),d('Name der Datei, die in das Lademodul enthaelt.'))
//mod-value *filename(user-id=*n)
//remove *value
//remove *value(from-f,phase)
//show *oper(test-opt)
TEST-OPTIONS = *NONE
//edit *oper(test-opt)
//mod-oper pres=*intern
//end
```

(21) Die Definition des Kommandos LOAD-PROGRAM wird ausgegeben und in analoger Weise geändert wie zuvor das Kommando START-PROGRAM.

(22) Der Operand PREFIX und sein Operandenwert werden für das Kommando LOAD-PROGRAM definiert. Dies geschieht nicht mit der ADD-Anweisung, sondern mit der COPY-Anweisung. Die in Arbeitsschritt 11 und 12 für START-PROGRAM erstellten Definitionen werden kopiert.

```
/mod-f-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all ——— (23)
/start-prog demo ————————————————————————————————————————— (24)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file ....
 .
 .
//end
/exit-job
 .
 .
/set-logon-parameters tsos,... ———————————————————————————— (25)
/mod-user example,profile-id=user1 ———————————————————————— (26)
/mod-sdf-param scope=*permanent,syntax-file=*group($sdfusr.sys.sdf.group.-
/syntax.example,user1) ———————————————————————————————————— (27)
% CMD0681 SYNTAX FILE '$SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE' INSERTED IN
PARAMETER FILE '$.SYSPAR.SDF'
% CMD0718 GROUP SYNTAX FILE '$SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE' HAS BEEN
ASSOCIATED WITH 'PROFILE-ID USER1' IN MEMORY TABLES
/exit-job
 .
 .
 .
```

(23) Die Datei SYS.SDF.GROUP.SYNTAX.EXAMPLE wird als mehrbenutzbar erklärt. Es ist nur Lesezugriff auf sie erlaubt.

(24) Das unter der Benutzerkennung SDFUSR katalogisierte Programm DEMO wird geladen und gestartet. Es ist das Programm SDF-A.

(25) Unter der privilegierten Benutzerkennung TSOS wird ein Prozess gestartet.

(26) Die PROFILE-ID USER1 wird der Benutzerkennung EXAMPLE zugewiesen.

(27) Die Gruppensyntaxdatei $SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE wird der PROFILE-ID USER1 zugewiesen. Die Zuweisung wird permanent in der SDF-Parameterdatei abgespeichert.

```
/set-logon-parameters example,...  ————————————————————————————— (28)
 .
 .
%CMD:show-sdf-options  ———————————————————————————————————————— (29)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : EXAMPLE#5
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *NO
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
%CMD:exec sdf-a  ——————————————————————————————————————————————— (30)
%  SDP0222 OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR 'SDP0116'. IN SYSTEM
MODE: /HELP-MSG SDP0116
```

(28) Unter der Benutzerkennung EXAMPLE wird ein Prozess gestartet.

(29) Die aktivierten Syntaxdateien werden angezeigt. Die Gruppensyntaxdatei $SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE (Versionsnummer EXAMPLE#5) ist aktiviert. Die Benutzerführung ist auf GUIDANCE=*NO eingestellt. SDF fordert deshalb zur Eingabe von Kommandos und Anweisungen mit „%CMD:“ bzw. „%STMT:“ auf.

(30) Da das Kommando EXEC entfernt wurde, interpretiert SDF die Benutzereingabe als das SDF-P-Kommando EXEC-CMD und weist es wegen fehlerhafter Syntax ab.

```
%CMD:start-prog sdf-a ──────────────────────────────────────────────── (31)
%  BLS0514 ERROR WHEN OPENING FILE $SDFUSR.SDF-A . DMS ERROR '0D33'. IN
SYSTEM MODE /HELP-MSG DMS0D33
%  NRTT101 ABNORMAL JOBSTEP TERMINATION BLS0514
%CMD:help-msg 0d33
%  DMS0D33 PROGRAM ERROR: REQUESTED FILE NOT CATALOGED
%  ? The requested file has not been cataloged in the system.
%    For the file or job variable (JV) no catalog entry could be found.
%  ! Correct the error and try again.
%CMD:start-prog $sdfusr.demo ────────────────────────────────────────── (32)
%  CMD0051 INVALID OPERAND 'FROM-FILE'
%  CMD0072 ATTRIBUTE SPECIFIED IN FILE NAME '$SDFUSR.DEMO' NOT PERMITTED
%ENTER OPERANDS:
%$sdfusr.demo
demo ──────────────────────────────────────────────────────────────── (33)
%  BLS0517 MODULE 'SDAMAIN' LOADED ──────────────────────────────────── (34)
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%STMT:open-syntax-file user,,*crea ──────────────────────────────────── (35)
```

(31) SDF akzeptiert den Kommandonamen START-PROGRAM. An die Implementierung übergibt SDF aber den Dateinamen $SDFUSR.SDF-A, den es durch Verkettung gebildet hat. Diese Datei existiert nicht.

(32) SDF akzeptiert den Dateinamen $SDFUSR.DEMO nicht, weil er verbotenerweise eine Benutzerkennung enthält.

(33) SDF akzeptiert den Dateinamen DEMO. An die Implementierung übergibt SDF aber den Dateinamen $SDFUSR.DEMO, den es durch Verkettung gebildet hat.

(34) Das Programm SDF-A, das unter der Benutzerkennung SDFUSR in der Datei DEMO steht (siehe Arbeitsschritt 24), wird geladen und gestartet.

(35) Die Benutzersyntaxdatei USER wird geöffnet und dabei neu eingerichtet. Standardmäßig werden die aktivierte Systemsyntaxdatei und die aktivierte Gruppensyntaxdatei als Referenzdateien zugewiesen.

```
%STMT:show *com(start-prog),siz=*max ————————————————————————————— (36)
START-PROGRAM(SRPG,SR,START-PROG)
   FROM SYS.SDF.GROUP.EXAMPLE (GROUP)
     Loads a $SDFUSR-program to the memory and starts it.
     FROM-FILE =
          filename_1..54_without-user-id-generation
          Specifies the name of the file holding the load module.
     CPU-LIMIT = JOB-REST
          JOB-REST or integer_1..32767
          specifies the maximum CPU time in seconds the program may use for
          execution
     MONJV = *NONE
          *NONE or filename_1..54_without-generation
          specifies the name of the job variable which is to monitor the
          program.
     RESIDENT-PAGES = *PARAMETERS
          *PARAMETERS()
          specifies the number of resident memory pages required for program
          execution
          STRUCTURE: *PARAMETERS
               MINIMUM = *STD
                    *STD or integer_0..32767
                    specifies the minimum number of resident memory pages
                    required
               MAXIMUM = *STD
                    *STD or integer_0..32767
                    specifies the maximum number of resident memory pages
                    required
     VIRTUAL-PAGES = *STD
          *STD or integer_0..32767
          specifies the total number of memory pages (both resident and
          pageable) required for program execution
%STMT:end
%CMD:exit-job
```

(36) Die Definition des Kommandos START-PROGRAM wird in der ausführlichsten Form ausgegeben. Die durchgeführten Änderungen sind sichtbar. Der Operand TEST-OPTIONS ist ausgeblendet.

3.1.6 Beispiel 6: Sperre für ein Kommando aufheben

In einer Benutzersyntaxdatei ist das Kommando START-SDF-A gesperrt. Die Sperre soll wieder aufgehoben werden.

```
/set-logon-parameters sdfusr,...  ————————————————————————————————  (1)
 .
/show-sdf-options  ———————————————————————————————————————————————  (2)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.SDFUSR
%              VERSION : UNDEFINED
%  USER      : 2OSH:$SDFUSR.SDF.USER.SYNTAX
%              VERSION : USER001
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *EXPERT
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet. Die unter dieser Benutzerkennung katalogisierte Benutzersyntaxdatei SDF.USER.SYNTAX wird bei der LOGON-Verarbeitung automatisch aktiviert. In dieser Syntaxdatei ist das Kommando START-SDF-A gesperrt.

(2) Die aktivierten Syntaxdateien werden angezeigt. Die Benutzersyntaxdatei SDF.USER.SYNTAX ist aktiviert.

```
/start-sdf-a ──────────────────────────────────────────────────────────── (3)
%  CMD0086 OPERATION NAME 'START-SDF-A' REMOVED BY USER
/mod-sdf-opt synt-file=*rem(*std) ──────────────────────────────────────── (4)
/start-sdf-a ──────────────────────────────────────────────────────────── (5)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sdf.user.syntax ─────────────────────────────────────── (6)
//show *com(start-sdf-a) ──────────────────────────────────────────────── (7)
%  CMD0051 INVALID OPERAND 'OBJECT=*COM:NAME'
%  SDA0083 NAME 'START-SDF-A' UNKNOWN
%  SDA0407 CORRECTION REJECTED OR NOT POSSIBLE. STATEMENT IGNORED
//restore *com(start-sdf-a) ──────────────────────────────────────────── (8)
//end
/mod-sdf-opt synt-file=*add(*std) ──────────────────────────────────────── (9)
/start-sdf-a ─────────────────────────────────────────────────────────── (10)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//end
/exit-job
 .
 .
```

(3) SDF akzeptiert das Kommando START-SDF-A nicht.

(4) Die Benutzersyntaxdatei SDF.USER.SYNTAX wird deaktiviert.

(5) SDF akzeptiert das Kommando START-SDF-A, da die Benutzersyntaxdatei SDF.USER.SYNTAX jetzt deaktiviert ist. SDF-A wird geladen und gestartet.

(6) Die Benutzersyntaxdatei SDF.USER.SYNTAX wird geöffnet. Standardmäßig werden die aktivierte Systemsyntaxdatei und die aktivierte Gruppensyntaxdatei als Referenzdateien zugewiesen.

(7) Da das Kommando START-SDF-A gesperrt ist, weist SDF-A seine Ausgabe zurück.

(8) Die Sperrung des Kommandos START-SDF-A wird aufgehoben. Voraussetzung dafür ist, dass die zugewiesenen Referenzsyntaxdateien die Definition abdecken.

(9) Die zuvor bearbeitete Benutzersyntaxdatei SDF.USER.SYNTAX wird aktiviert.

(10) Das Kommando START-SDF-A ist nicht mehr gesperrt. Die Sperre wurde mit dem Arbeitsschritt 8 erfolgreich aufgehoben.

3.2 Funktionsumfang einschränken

Sie können mit SDF-A den Funktionsumfang des BS2000 systemweit oder kennungsspezifisch einschränken. Eine systemweite Einschränkung ist in einer Systemsyntaxdatei zu definieren, eine kennungsspezifische in einer Gruppensyntaxdatei.

Beim Definieren der Einschränkung in einer Gruppensyntaxdatei ist Folgendes zu beachten: wenn die Definition des Kommandos bzw. der Anweisung nicht bereits in der Gruppensyntaxdatei steht, muss die zugehörige Systemsyntaxdatei als Referenz (OPEN-SYNTAX-FILE ... SYSTEM-DESCRIPTIONS=) zugewiesen sein.

Kommandos, Anweisungen, Operanden und Operandenwerte können Sie generell mit der Anweisung REMOVE oder gezielt für bestimmte Betriebsarten mit der Anweisung MODIFY-xxx ...,DIALOG-ALLOWED=...,DIALOG-PROC-ALLOWED=...,BATCH-ALLOWED=..., BATCH-PROC-ALLOWED=... sperren. Kommandos können Sie außerdem in Abhängigkeit von der Art ihres Aufrufs (CMD-ALLOWED=) sperren.

Funktionsumfang eines Kommandos einschränken

Wenn Sie den Funktionsumfang eines Kommandos wirksam einschränken wollen, müssen Sie sicherstellen, dass die definierte Einschränkung nicht mithilfe anderer Kommandos umgangen werden kann. In den meisten Fällen kann eine Einschränkung mithilfe der alten ISP-Kommandos umgangen werden. Diese sind so definiert, dass ihr Funktionsumfang nicht mit SDF-A modifiziert werden kann. Man kann sie nur sperren. Wenn sie über die MCLP(Macro Command Language Processor)-Schnittstelle aufgerufen werden können, kann ein generelles Sperren unvorhersehbare Folgen haben. Es kann dann angebracht sein, diese Kommandos vorsichtshalber nur für den Dialog- und Stapelbetrieb zu sperren, nicht aber für den Aufruf über MCLP. Eine solche partielle Sperre kann allerdings relativ leicht umgangen werden, z.B. mithilfe des EDT. Was sinnvoll ist, hängt vom Einzelfall ab (siehe Beispiele 4 und 6). Kommandos, die über das SDF-P-Kommando EXECUTE-CMD abgesetzt werden (z.B. um ihre Ausgabe in eine Variable umzulenken), unterliegen nicht den Vorgaben für DIALOG-ALLOWED=...,DIALOG-PROC-ALLOWED=..., BATCH-ALLOWED=... und BATCH-PROC-ALLOWED=..., da sie über die MCLP-Schnittstelle aufgerufen werden.

Operanden sperren

Wenn Sie einen Operanden sperren wollen, können Sie meist nicht abschätzen, welche Folgen es aufseiten der Implementierung hat, wenn Sie seine Definition löschen. Im Zweifelsfall empfiehlt es sich, den Operanden lediglich mit der Anweisung MODIFY-OPERAND...,PRESENCE=*INTERNAL-ONLY an der Benutzeroberfläche auszublenden. Für ihn muss dann ein Default-Wert definiert sein, der durch einen für den Operanden definierten Operandenwert abgedeckt ist.

Auch die Definition eines geheimen Operanden und dessen Verkettung mit einem sichtbaren Operanden ist eine Möglichkeit, mit der sich eine Funktionseinschränkung definieren lässt (siehe Beispiel 5, Seite 75).

Operandenwerte einschränken

Für Operandenwerte können Sie mit der Anweisung MODIFY-VALUE Einschränkungen hinsichtlich der zulässigen Länge oder des zulässigen Zahlenwerts definieren. Außerdem können Sie bei MODIFY-VALUE...,VALUE=(<c-string>(...),...),... eine Liste von zulässigen Eingabewerten angeben. Dann werden als Eingabe nur die in der Liste definierten Werte akzeptiert. Mit der Anweisung MODIFY-VALUE...,VALUE=<c-string>(...,OUTPUT=...,) können Sie festlegen, dass SDF einen definierten Einzelwert vor der Übergabe an die Implementierung in einen anderen Wert umwandelt oder die Übergabe des Wertes unterdrückt. Eine Einschränkung lässt sich in manchen Fällen auch durch Änderung des Datentyps erreichen. Für Operandenwerte, die als voll- oder teilqualifizierte Dateinamen definiert sind, können Sie die Angabe der Katalogkennung, der Benutzerkennung, einer Generationsnummer oder einer Versionsbezeichnung sowie die Eingabe von Platzhalterzeichen als unzulässig erklären. Die Angabe von Platzhalterzeichen können Sie auch bei den Datentypen <alphanum-name>, <composed-name> und <name> erlauben oder verbieten.

Funktionsumfang eines Benutzerprogramms einschränken

Die in einer System- oder Gruppensyntaxdatei definierte Einschränkung für ein Benutzerprogramm kann mithilfe einer Benutzersyntaxdatei wirkungslos gemacht werden. Wenn Sie den Funktionsumfang eines Benutzerprogramms *wirksam* einschränken wollen, müssen Sie demzufolge sicherstellen, dass die betroffenen Benutzer nicht den vollen Funktionsumfang in einer Benutzersyntaxdatei definieren können.

Eingeschränktes Laden und Ausführen von Programmen

Die Systembetreuung kann für Benutzer das Laden und Ausführen von Programmen auf bestimmte Anwendungen einschränken (z.B. EDT). Dazu kann ein Kommando START-EDITOR in den SDF-Anwendungsbereichen UTILITIES oder PROGRAMMING-SUPPORT definiert werden (siehe „Beispiel 4: Nur ein Programm (EDT) zulassen“ auf Seite 65).

Damit die Kommandos START-PROGRAM, EXEC, LOAD-PROGRAM und LOAD nicht mehr direkt ausführbar sind, können diese Kommandos in System- oder Gruppensyntaxdateien mit den Attributen DIALOG-ALLOWED=*NO und BATCH-ALLOWED=*NO versehen werden. Die gleichen Attribute müssen für die Kommandos CALL-PROCEDURE, CALL und DO angegeben werden. Diese Methode verhindert das direkte Ausführen jedes Programms und jeder Prozedur, ermöglicht aber das Ausführen von Prozedurkommandos, wobei die aufgerufenen Prozeduren die für direkte Ausführung gesperrten Kommandos (START-PROGRAM,...) enthalten können.

3.3 Einschränkungen aufheben

Was zu tun ist, hängt davon ab,

– in welchem Typ Syntaxdatei die Einschränkung definiert ist

– ob Sie die Einschränkung generell oder nur für einzelne Benutzer aufheben wollen

– ob der Funktionsumfang eines durch System-Module implementierten Kommandos, eines durch Prozedur implementierten Kommandos oder eines Benutzer-Programms eingeschränkt ist

– ob Sie die Einschränkung vollständig oder nur teilweise aufheben wollen

– ob die Einschränkung durch Sperren, Löschen oder Modifizieren einer Kommando- bzw. Anweisungsdefinition realisiert ist.

Die Vorgehensweise gleicht häufig der, die beim Definieren der Einschränkung angewandt wird:

– Ist die Einschränkung durch Löschen einer Kommando-, Anweisungs-, Operanden- oder Operandenwertdefinition realisiert, so lässt sie sich dadurch aufheben, dass mit COPY die gelöschte Definition wieder eingebracht wird.

– Ist die Einschränkung durch Änderung einer Kommando-, Anweisungs-, Operanden- oder Operandenwertdefinition realisiert, so ist in einem ersten Arbeitsschritt die veränderte Definition mit REMOVE zu löschen. In einem zweiten Arbeitsschritt ist dann die ursprüngliche Definition mit COPY wieder einzubringen.

– Soll die in einer Gruppen- oder Benutzersyntaxdatei definierte Änderung eines Kommandos oder einer Anweisung vollständig aufgehoben werden, so ist die Syntaxdatei ohne Zuweisung einer Referenzdatei zu eröffnen und die geänderte Definition mit REMOVE zu löschen. Voraussetzung für diese Vorgehensweise ist, dass die Originalfassung der Definition nach wie vor in der zugehörigen System- oder Gruppensyntaxdatei steht.

– Mithilfe der Anweisung RESTORE kann die in einer Gruppen- oder Benutzersyntaxdatei definierte Sperre eines Kommandos oder einer Anweisung aufgehoben werden. Wird die Definition eines durch System-Module implementierten Kommandos in einer Benutzersyntaxdatei entsperrt, so ist zu beachten, dass die Kommandodefinition durch die beim Eröffnen der Benutzersyntaxdatei zugewiesenen Referenzsyntaxdateien abgedeckt sein muss.

4 Eigene Kommandos und Anweisungen definieren und implementieren

4.1 Syntax gestalten

In diesem Abschnitt sind Regeln zusammengestellt, die beim Gestalten der Syntax zu beachten sind. Der Abschnitt informiert nicht über einzelne Details, z.B. über die maximale Länge eines Operandennamens. Derartige Detailinformationen finden Sie in der Beschreibung der ADD-Anweisungen, mit denen die Syntax der Kommandos und Anweisungen in eine Syntaxdatei einzubringen ist.

In der folgenden Zusammenstellung werden für Empfehlungen die Wörter „sollen“ und „können“ benutzt, für zwingende Vorschriften das Wort „müssen“. Die Empfehlungen für Kommandos gelten auch für Anweisungen.

– Der Name eines Kommandos, einer Anweisung oder eines Operanden sowie ein Operandenwert vom Typ KEYWORD (Schlüsselwortwert) kann aus mehreren, durch Bindestrich miteinander verbundenen Teilnamen zusammengesetzt sein. Alle Teilnamen sollten der natürlichen Sprache entstammen. Für gleiche Sachverhalte sollte derselbe Teilname verwendet werden.

  Bei der Eingabe können die Teilnamen von rechts nach links beliebig abgekürzt oder ganz weggelassen werden, solange SDF anhand der Abkürzung den gesamten Namen eindeutig identifizieren kann.

– Der Name eines Kommandos sollte mit einem Verb beginnen. Die dem Verb folgenden Teilnamen sollten das Objekt bezeichnen, das mit dem Kommando bearbeitet wird. Der ALIAS-NAME muss von NAME oder STANDARD-NAME verschieden sein. Soll mit absoluter Sicherheit ausgeschlossen werden, dass ein Kommandoname nach einem Versionswechsel mit dem Namen eines neuen, von Fujitsu Siemens Computers gelieferten Kommandos kollidiert, so kann man den Kommandonamen mit dem Teilnamen „X“ beginnen lassen.

– Jede Funktion sollte durch ein eigenes Kommando abgedeckt sein. Dabei ist darauf zu achten, dass die Funktion nicht zu komplex ist. Für gleiche Funktionen sollte es gleiche Kommandos geben.

- Jeder Operand muss einen Namen haben.
  Bei der Eingabe kann ein Operand wahlweise als Stellungsoperand oder als Schlüsselwortoperand angegeben werden.
- Wahlweise Operanden müssen einen Default-Wert haben.
- Operanden, die nur dann relevant sind, wenn ein anderer Operand einen ganz bestimmten Wert hat, sollten in einer Struktur an diesem Wert hängen. Eine Struktur besteht aus einem oder mehreren in runde Klammern eingeschlossenen Operanden. Sie hängt am Wert eines übergeordneten Operanden.
- Logisch zusammengehörende Operanden sollten in einer Struktur zusammengefasst werden. Es kann sinnvoll sein, eigens zum Zweck der Struktureinleitung einen übergeordneten Operanden zu konstruieren.
- Es können nicht mehr als fünf Strukturen ineinander geschachtelt werden.
- Der Name eines in einer Struktur stehenden Operanden muss nur innerhalb der Struktur eindeutig sein. Für die Eingabe ist es allerdings vorteilhaft, wenn er kommandoglobal eindeutig ist. Durch Angabe des Operanden kann dann die Struktur implizit mit ausgewählt werden.
- Ein Operandenwert muss als einer der folgenden Datentypen definiert sein:

<alphanumeric-name>	alphanumerischer Name,
<cat-id>	Katalogkennung,
<composed-name>	zusammengesetzter Name,
<c-string>	C-Zeichenkette,
<date>	Datum,
<device>	Gerät,
<fixed>	Festpunktzahl,
<filename>	Dateiname,
<integer>	Ganzzahl,
KEYWORD	Schlüsselwort,
<name>	Name,
<partial-filename>	teilqualifizierter Dateiname,
<posix-pathname>	POSIX-Pfadname
<posix-filename>	POSIX-Dateiname
<product-version>	Produkt-Version,
<structured-name>	strukturierter Name,
<time>	Uhrzeit,
<vsn>	Datenträger,
<x-string>	X-Zeichenkette,
<x-text>	X-Text.

Bei einigen dieser Datentypen lässt sich der mögliche Operandenwert durch zusätzliche Angaben noch genauer eingrenzen, z.B. durch Angabe einer minimalen und maximalen Länge (siehe ADD-VALUE). Die Datentypen <command-rest>, KEYWORD-NUMBER, <label> und <text> sind der Software-Entwicklung von Fujitsu Siemens Computers vorbehalten.

– Mit Ausnahme der Schlüsselwörter müssen die für einen Operanden definierten Eingabealternativen von verschiedenem Datentyp sein. Diese Datentypen müssen syntaktisch disjunkt sein, es sei denn, dass mit der Anweisung ADD-OPERAND...VALUE-OVERLAPPING=*YES eine Überlappung von Datentypen erlaubt wurde. Ansonsten kann beispielsweise für einen Operanden nicht gleichzeitig ein Wert vom Typ NAME und ein alternativer Wert vom Typ STRUCTURED-NAME definiert werden (siehe „Sich ausschließende Datentypen“ auf Seite 635).

VALUE-OVERLAPPING=*YES sollte nur vom SDF-A Experten benutzt werden, und dann auch nur, wenn sich das Problem nicht anders lösen lässt!

– Wenn zur Unterscheidung von anderen Eingabealternativen erforderlich, muss einem Schlüsselwort ein Stern vorangestellt werden
(siehe ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY)).

– Es kann definiert werden, dass zu einem Operanden eine Liste von Werten eingegeben werden kann. Bei der Eingabe sind diese Werte durch Komma voneinander zu trennen und in runde Klammern einzuschließen.

– Die Koexistenz von Operandenwerten des Datentyps KEYWORD und Operandenwerten eines Datentyps mit Wildcards führt zu speziellen Situationen. Im Folgenden finden Sie eine Liste 6 möglicher Situationen und das Ergebnis der SDF-Syntaxanalyse bei verschiedenen Eingaben:

1. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*YES
   ADD-VALUE TYPE=*KEYWORD(STAR=*OPTIONAL),VALUE='ALL'
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)

2. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*YES
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)
   ADD-VALUE TYPE=*KEYWORD(STAR=*OPTIONAL),VALUE='ALL'

3. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*NO
   ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY),VALUE='ALL'
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)

4. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*NO
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)
   ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY),VALUE='ALL'

5. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*YES
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)
6. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*NO
   ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY),VALUE='V4.1'
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)

SDF V4.1 erkennt die Eingaben bei der Syntaxanalyse als folgenden Datentyp:

Eingabe von:	**Situation Nr.**					
	1	**2**	**3**	**4**	**5**	**6**
A	keyword	filename	filename	filename	filename	filename
*A	keyword	keyword	keyword	keyword	Fehler	Fehler
**A	filename	filename	filename	filename	filename	filename
*A/	filename	filename	filename	filename	filename	filename
*	filename	filename	filename	filename	filename	filename
*B	Fehler	Fehler	Fehler	Fehler	Fehler	Fehler
*V4.1	Fehler	Fehler	Fehler	Fehler	Fehler	keyword
*V4.	filename	filename	filename	filename	filename	filename
**V4.1	filename	filename	filename	filename	filename	filename

4.2 Kommandos definieren und implementieren

4.2.1 Beispiel 1: Kommando zum Assemblieren

Es soll ein Kommando definiert und mittels Prozedur implementiert werden, das ein Assembler-Quellprogramm übersetzt und das erzeugte Bindemodul in der EAM-Bindemoduldatei speichert. Die benötigten Makros können in einer benutzereigenen Makrobibliothek stehen. Das Kommando soll folgendes Format haben:

ASSEMBLE-SOURCE
SOURCE = <filename 1..54> ,**MACRO-LIBRARY** = ***NONE** / <filename 1..54> ,**TEST-SUPPORT** = ***NO** / ***YES**

Das Kommando wird mittels Prozedur implementiert. Die symbolischen Operanden stehen als Stellungsoperanden in der Operandenliste des BEGIN-PROCEDURE-Kommandos. Die folgende Prozedur steht als Element ASSEMB in der Programmbibliothek $SDFUSR.PROC.LIB.

```
/BEGIN-PROCEDURE PARAMETERS=*YES(PROC-PARAM=(&SOURCE,&MACROLIB,&TEST))
/DELETE-SYSTEM-FILE FILE-NAME=*OMF
/ASSIGN-SYSDTA TO=*SYSCMD
/START-ASSEMBH
//     COMPILE SOURCE=&SOURCE,-
//             MACRO-LIB=&MACROLIB,-
//             TEST=&TEST
//     END
/SET-JOB-STEP
/ASSIGN-SYSDTA TO-FILE=*PRIMARY
/END-PROCEDURE
```

Das Kommando ASSEMBLE-SOURCE wird in der Benutzersyntaxdatei SDF.USER.SYNTAX definiert. Anschließend wird es getestet.

```
/set-logon-parameters  sdfusr,...  ———————————————————————————  (1)
/mod-sdf-options syntax-file=*remove(*std)  ——————————————————————  (2)
/start-sdf-a  ————————————————————————————————————————————————  (3)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%//open-syntax-file sdf.user.syntax  ———————————————————————————  (4)
%//set-glob cont=*new  ————————————————————————————————————————  (5)
%//add-cmd assemble-source,help=e('Assembles a program'), -
%//domain=programming-support, -
%//implementor=*proc('*lib-elem(lib=$sdfusr.proc.lib,elem=assemb)')  ——  (6)
%//add-oper source,res-oper-name=*pos(1)  ——————————————————————  (7)
%//add-value *filename  ————————————————————————————————————————  (8)
%//add-oper macro-library,def='*none',res-oper-name=*pos(2)  ———————  (9)
%//add-value *keyw(*mand),value='*none'  ———————————————————————  (10)
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) Die bei der LOGON-Verarbeitung automatisch aktivierte Benutzersyntaxdatei SDF.USER.SYNTAX wird deaktiviert.

(3) SDF-A wird geladen und gestartet.

(4) Die bereits existierende Benutzersyntaxdatei SDF.USER.SYNTAX wird geöffnet.

(5) In der Globalinformation wird festgelegt, dass das Fortsetzungszeichen „-“ für Folgezeilen bei Eingabe von SYSCMD oder SYSSTMT in Spalte 2 bis 72 stehen kann.

(6) Der Kopf des Kommandos ASSEMBLE-SOURCE wird definiert. Es erhält einen englischen Hilfetext und wird dem Anwendungsbereich PROGRAMMING-SUPPORT zugeordnet. Es ist mittels der Prozedur implementiert, die als Element ASSEMB in der Programmbibliothek $SDFUSR.PROC.LIB steht.

(7) Der erste Operand des Kommandos ASSEMBLE-SOURCE wird definiert. Sein Name ist SOURCE. In der Zeichenkette, die an die Prozedur übergeben wird, steht er an erster Position.

(8) Es wird definiert, dass der Wert des Operanden SOURCE vom Datentyp FILENAME sein muss.

(9) Der zweite Operand des Kommandos ASSEMBLE-SOURCE wird definiert. Sein Name ist MACRO-LIBRARY. Sein Default-Wert ist *NONE. In dem an die Prozedur zu übergebenden String steht er an zweiter Position.

(10) Es wird definiert, dass das Schlüsselwort NONE ein zulässiger Wert des Operanden MACRO-LIBRARY ist. Ihm muss bei der Eingabe ein Stern vorangestellt sein.

```
%//add-value *filename ------------------------------------------------------ (11)
%//add-oper test-support,def='no',res-oper-name=*pos(3) ---------------------- (12)
%//add-value *keyw,value='no' ------------------------------------------------ (13)
%//add-value *keyw,value='yes' ----------------------------------------------- (14)
%//close-cmd ----------------------------------------------------------------- (15)
%//show *com(assemb-source),siz=*max ----------------------------------------- (16)
ASSEMBLE-SOURCE
     Assembles a program
     SOURCE =
          filename_1..54
     MACRO-LIBRARY = *NONE
          *NONE or filename_1..54
     TEST-SUPPORT = *NO
          *NO or *YES
%//end ----------------------------------------------------------------------- (17)
```

(11) Es wird definiert, dass der Wert des Operanden MACRO-LIBRARY vom Datentyp FILENAME sein kann.

(12) Der dritte globale Operand des Kommandos ASSEMBLE-SOURCE wird definiert. Sein Name ist TEST-SUPPORT und der Default-Wert ist NO. In dem an die Prozedur zu übergebenden String steht er an dritter Position.

(13) Es wird definiert, dass das Schlüsselwort NO ein zulässiger Wert des Operanden TEST-SUPPORT ist.

(14) Es wird definiert, dass das Schlüsselwort YES ein zulässiger Wert des Operanden TEST-SUPPORT ist.

(15) Die Definition des Kommandos ASSEMBLE-SOURCE wird beendet.

(16) Die in der Benutzersyntaxdatei SDF.USER.SYNTAX erstellte Definition des Kommandos ASSEMBLE-SOURCE wird in der ausführlichsten Form ausgegeben. Für alle Dateinamen hat SDF-A eine minimale Länge von 1 und eine maximale von 54 definiert.

(17) SDF-A wird beendet. Die Benutzersyntaxdatei SDF.USER.SYNTAX wird damit implizit abgespeichert.

```
/mod-sdf-opt synt-file=*add(*std),guid=*n ————————————————————————— (18)
/assemb-source demo.prog1,macro-lib=demo.maclib.test-support=*yes ———— (19)
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2B00' OF '1998-04-24' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1990. ALL RIGHTS
RESERVED
%  ASS6010 V01.2B02 OF BS2000 ASSTRAN  READY
%  ASS6011 ASSEMBLY TIME: 302 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NOTE
%  ASS6006 LISTING GENERATOR TIME: 130 MSEC
%  ASS6012 END OF ASSTRAN
 .
 .
 .
```

(18) Die Benutzersyntaxdatei SDF.USER.SYNTAX, in der das Kommando ASSEMBLE-SOURCE definiert ist, wird aktiviert.

(19) Das Kommando ASSEMBLE-SOURCE wird eingegeben. Die benutzereigene Makrobibliothek DEMO.MACLIB wird angegeben. Nach Ausführung des Kommandos ist das erzeugte Bindemodul in der EAM-Bindemoduldatei abgelegt.

4.2.2 Beispiel 2: Kommando zum Ausgeben von Datei-Inhalten

Es soll ein Kommando definiert und mittels Prozedur implementiert werden, das den Inhalt einer SAM- oder ISAM-Datei auf SYSOUT ausgibt. Standardmäßig stehen in den auszugebenden Dateien Sätze variabler Länge. Es können aber auch Dateien mit Sätzen fester Länge ausgegeben werden. Bei den auszugebenden ISAM-Dateien ist der ISAM-Schlüssel im Standardfall acht Byte lang. Er kann aber auch kürzer sein. Das Kommando soll folgendes Format haben:

```
TYPE-FILE

NAME = <filename 1..54>
,ACCESS-METHOD = *ISAM(...) / *SAM
    *ISAM( . . . )
        | KEY-LENGTH = 8 / <integer 1..8>
,RECORD-SIZE = *VARIABLE / <integer 1..256>
```

Das Kommando wird mittels Prozedur implementiert. Die symbolischen Operanden stehen als Stellungsoperanden in der Operandenliste des BEGIN-PROCEDURE-Kommandos. In der Prozedur wird mit dem Dienstprogramm EDT gearbeitet. Die folgende Prozedur steht in der Datei $SDFUSR.TYPE.FILE:

```
/       BEGIN-PROCEDURE N,PARAM=YES(PROC-PARAM=(&FILE,&ACCESS,&KEY,&RECORD),-
/       ESCAPE-CHARACTER='&')
/       MODIFY-JOB-SWITCHES ON=(4,5)
/       ASSIGN-SYSDTA TO-FILE=*SYSCMD
/       CREATE-JV JV=RECORD-SIZE
/       MODIFY-JV JV=RECORD-SIZE,VALUE='&RECORD'
/       LOAD-PROGRAM FROM-FILE=$EDT
/       SKIP-COMMAND TO-LABEL=&ACCESS   "ACCESS IST ISAM ODER SAM"
/.SAM SKIP-COMMANDS TO-LABEL=SAM1,IF=JV(COND=(RECORD-SIZE='VARIABLE'))
/       SET-FILE-LINK LINK-NAME=EDTSAM,FILE-NAME=&FILE,ACCES-METHOD=SAM,-
/       RECORD-FORMAT=FIXED(REC-SIZE=&RECORD)
/.SAM1 RESUME-PROGRAM
@       READ '&FILE'
/       HOLD-PROGRAM
/       SKIP-COMMANDS TO-LABEL=COMMON,IF=JV(COND=(RECORD-SIZE='VARIABLE'))
/       REMOVE-FILE-LINK LINK-NAME=EDTSAM
/       SKIP-COMMANDS TO-LABEL=COMMON
/.ISAM CREATE-JV JV=ISAM-KEY
/       MODIFY-JV JV=ISAM-KEY,VALUE='&KEY'
/       SKIP-COMMANDS TO-LABEL=ISAM3,IF=JV(COND=(ISAM-KEY='8' AND -
/       RECORD-SIZE='VARIABLE'))
/       SKIP-COMMANDS TO-LABEL=ISAM1,IF=JV(COND=(ISAM-KEY='8'))
/       SKIP-COMMANDS TO-LABEL=ISAM2,IF=JV(COND=(RECORD-SIZE='VARIABLE'))
/       SET-FILE-LINK LINK-NAME=EDTISAM,FILE-NAME=&FILE,ACCESS-METHOD=-
/       ISAM(KEY-LEN=&KEY,KEY-POS=1),RECORD-FORMAT=FIXED(REC-SIZE=&RECORD)
/       SKIP-COMMANDS TO-LABEL=ISAM3
/.ISAM1 SET-FILE-LINK LINK-NAME=EDTISAM,FILE-NAME=&FILE,ACCES-METHOD=-
/       ISAM(KEY-LEN=8,KEY-POS=1),RECORD-FORMAT=FIXED(REC-SIZE=&RECORD)
/       SKIP-COMMANDS TO-LABEL=ISAM3
/.ISAM2 SET-FILE-LINK LINK-NAME=EDTISAM,FILE-NAME=&FILE,ACCES-METHOD=-
/       ISAM(KEY-LEN=&KEY)
/.ISAM3 RESUME-PROGRAM
@       GET '&FILE'
/       HOLD-PROGRAM
/       SKIP-COMMANDS TO-LABEL=ISAM4,IF=JV(COND=(ISAM-KEY='8' AND -
/       RECORD-SIZE='VARIABLE'))
/       REMOVE-FILE-LINK LINK-NAME=EDTISAM
/.ISAM4 DELETE-JV JV=ISAM-KEY
/.COMMON RESUME-PROGRAM
@       PRINT %.-$N
@       HALT
/       SET-JOB-STEP
/       DELETE-JV JV=RECORD-SIZE
/       ASSIGN-SYSDTA TO-FILE=*PRIMARY
/       MODIFY-JOB-SWITCHES OFF=(4,5)
/       END-PROCEDURE
```

Das Kommando TYPE-FILE wird in der Benutzersyntaxdatei SDF.USER.SYNTAX definiert. Anschließend wird es getestet.

```
/set-logon-parameters sdfusr,...  ————————————————————————————  (1)
 .
 .
/mod-sdf-opt synt-file=*remove(*std)  ————————————————————————  (2)
/start-sdf-a  ————————————————————————————————————————————————  (3)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%//open-syntax-file sdf.user.syntax  ———————————————————————————  (4)
%//set-globals cont=*new  ——————————————————————————————————————  (5)
%//add-cmd type-file,help=e('Outputs the contents of an ISAM or SAM -
%//file to SYSOUT.'),domain=user,impl=*proc('$sdfusr.type.file')  ————  (6)
%//add-oper name,help=e('Name of file to be output'),res-oper-nam=*pos(1) (7)
%//add-value *filename  ————————————————————————————————————————  (8)
%//add-oper access-method,def='isam',res-oper-name=*pos(2)  ————————  (9)
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) Die bei der LOGON-Verarbeitung automatisch aktivierte Benutzersyntaxdatei SDF.USER.SYNTAX wird deaktiviert.

(3) SDF-A wird geladen und gestartet.

(4) Die bereits existierende Benutzersyntaxdatei SDF.USER.SYNTAX wird eröffnet.

(5) In der Globalinformation wird festgelegt, dass das Fortsetzungszeichen „-“ für Folgezeilen bei Eingabe von SYSCMD in Spalte 2 bis 72 stehen kann.

(6) Der Kopf des Kommandos TYPE-FILE wird definiert. Es erhält einen englischen Hilfetext und wird dem Anwendungsbereich USER zugeordnet. Es ist mittels der Prozedur implementiert, die in der Datei $SDFUSR.TYPE.FILE steht.

(7) Der erste Operand des Kommandos TYPE-FILE wird definiert. Sein Name ist NAME. Er erhält einen englischen Hilfetext. In dem an die Prozedur zu übergebenden String steht er an erster Position.

(8) Es wird definiert, dass der Wert des Operanden NAME vom Datentyp FILENAME sein muss.

(9) Der zweite Operand des Kommandos TYPE-FILE wird definiert. Sein Name ist ACCESS-METHOD. Sein Default-Wert ist ISAM. In dem an die Prozedur zu übergebenden String steht er an zweiter Position.

```
%//add-value *keyw,struct=*y,value='isam' ------------------------------ (10)
%//add-oper key-length,def='8',res-oper-name=*pos(3) ------------------- (11)
%//add-value *integ(1,8) ----------------------------------------------- (12)
%//close-struct -------------------------------------------------------- (13)
%//add-value *keyw,value='sam' ----------------------------------------- (14)
%//add-oper record-size,def='variable',res-oper-name=*pos(4) ----------- (15)
%//add-value *keyw,value='variable' ------------------------------------ (16)
%//add-value *integ(1,256) --------------------------------------------- (17)
%//close-cmd ----------------------------------------------------------- (18)
```

(10) Es wird definiert, dass das Schlüsselwort ISAM ein zulässiger Wert des Operanden ACCESS-METHOD ist. ISAM leitet eine Struktur ein.

(11) Der erste Operand in der Struktur ISAM wird definiert. Sein Name ist KEY-LENGTH. Sein Default-Wert ist die Ganzzahl 8. In dem an die Prozedur zu übergebenden String steht er an dritter Position.

(12) Es wird definiert, dass der Wert des Operanden KEY-LENGTH vom Datentyp INTEGER sein muss. Als untere Schranke wird 1 und als obere 8 definiert.

(13) Die gerade bearbeitete Struktur ISAM wird geschlossen.

(14) Es wird definiert, dass das Schlüsselwort SAM ein zulässiger Wert des Operanden ACCESS-METHOD ist.

(15) Der dritte Operand des Kommandos TYPE-FILE wird definiert. Sein Name ist RECORD-SIZE. Sein Default-Wert ist VARIABLE. In dem an die Prozedur zu übergebenden String steht er an vierter Position.

(16) Es wird definiert, dass das Schlüsselwort VARIABLE ein zulässiger Wert des Operanden RECORD-SIZE ist.

(17) Es wird definiert, dass der Wert des Operanden RECORD-SIZE vom Datentyp INTEGER sein kann. Als untere Grenze wird 1 und als obere 256 definiert.

(18) Die Definition des Kommandos TYPE-FILE wird beendet.

```
%//show *com(type-f),siz=*max ──────────────────────────────────────── (19)
TYPE-FILE
     Outputs the contents of an ISAM or SAM file to SYSOUT.
     NAME =
          filename_1..54
          Name of file to be output
     ACCESS-METHOD = *ISAM
          *ISAM() or *SAM
          STRUCTURE: *ISAM
               KEY-LENGTH = 8
                    integer_1..8
     RECORD-SIZE = *VARIABLE
          *VARIABLE or integer_1..256
%//end
/type-file demo.1 ──────────────────────────────────────────────────── (20)
%  CMD0186 OPERATION NAME 'TYPE-FILE' UNKNOWN
/mod-sdf-opt synt-file=*add(*std) ──────────────────────────────────── (21)
/type-f demo.1 ─────────────────────────────────────────────────────── (22)
Contents of ISAM file DEMO.1
 .
 .
/type-f demo.2,rec-siz=52
Contents of ISAM file DEMO.2
 .
 .
/type-f demo.3,isam(6)
Contents of ISAM file DEMO.3
 .
 .
```

(19) Die in der Benutzersyntaxdatei SDF.USER.SYNTAX erstellte Definition des Kommandos TYPE-FILE wird in der ausführlichsten Form ausgegeben. Für FILENAME hat SDF-A standardmäßig eine minimale Länge von 1 und eine maximale von 54 Zeichen festgesetzt.

(20) SDF lehnt das Kommando TYPE-FILE ab, da es in keiner aktivierten Syntaxdatei definiert ist.

(21) Die Benutzersyntaxdatei SDF.USER.SYNTAX wird aktiviert.

(22) Verschiedene ISAM-Dateien werden mit dem Kommando TYPE-FILE ausgegeben:
DEMO.1: der ISAM-Schlüssel ist 8 Byte lang, die Satzlänge ist variabel
DEMO.2: der ISAM-Schlüssel ist 8 Byte lang, die Sätze sind 52 Byte lang
DEMO.3: der ISAM-Schlüssel ist 6 Byte lang, die Satzlänge ist variabel

```
/type-f demo.4,sam,rec-siz=60 ──────────────────────────────────────── (23)
Contents of SAM file DEMO.4
 .
 .
/type-f? ──────────────────────────────────────────────────────────────── (24)
```

```
COMMAND  : TYPE-FILE


------------------------------------------------------------------------------
NAME                  = ?
ACCESS-METHOD         = *ISAM(KEY-LENGTH=8)
RECORD-SIZE           = ?VARIABLE
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
                                                                    
------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

(23) Die SAM-Datei DEMO.4 wird mit dem Kommando TYPE-FILE ausgegeben. Sie hat eine feste Satzlänge von 60 Byte.

(24) Für die Eingabe des Kommandos TYPE-FILE wird in den temporär geführten Dialog gewechselt.

Über die Operanden NAME und RECORD-SIZE werden weitere Informationen angefordert.

```
COMMAND  : TYPE-FILE
OPERANDS : NAME=?,RECORD-SIZE=*VARIABLE

-------------------------------------------------------------------------------
NAME                  = demo.5
                        filename_1..54
                        Name of the file to be output
ACCESS-METHOD         = *ISAM(KEY-LENGTH=8)
RECORD-SIZE           = 55
                        *VARIABLE or integer_1..256









-------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

```
Contents of ISAM file DEMO.5
 .
 .
/exit-job
```

Es wird angegeben, dass der Name der auszugebenden Datei demo.5 ist und dass die Sätze eine feste Länge von 55 Byte haben. Die Zugriffsmethode *ISAM(KEY-LENGTH=8) ist der Default-Wert und deshalb schon voreingestellt.

Anschließend wird das Kommando ausgeführt. Nach der Kommandoausführung wird der Prozess beendet.

4.2.3 Hinweise

Wenn ein mittels Prozedur implementiertes Kommando allgemein verfügbar sein soll, muss

– die Prozedurdatei ausführend mehrbenutzbar sein

– die Benutzerkennung, unter der die Prozedurdatei katalogisiert ist, als Teil des Dateinamens in der Kommandodefinition angegeben sein (siehe ADD-CMD).

In der Operandenliste, die SDF an die Prozedur übergibt, können die Operanden als Schlüsselwort- oder als Stellungsoperanden definiert sein. Bei einem Schlüsselwortoperanden kann ein anderer Operandenname übergeben werden, als der, der in der Kommandosyntax steht (siehe ADD-OPERAND...,RESULT-OPERAND-NAME=).

Operandendefinitionen	**Kommandoeingabe**	**an die Prozedur übergebene Parameterliste**

//ADD-OPER OP1,...,RES-OPER-N=*POS(1)
//ADD-OPER OP2,...,RES-OPER-N=*POS(2)
//ADD-OPER OP3,...,RES-OPER-N=*SAME
//ADD-OPER OP4,...,RES-OPER-N=PARAM
	/KDO W,X,Y,Z
	/KDO W,OP4=Z,
	OP3=Y,OP2=X
		('W','X',OP3='Y',
		PARAM='Z')

Mehrere Operanden des Kommandos können zu einem einzigen Operanden verkettet an die Prozedur übergeben werden (siehe ADD-OPERAND...,RESULT-OPERAND-NAME=, CONCATENATION-POS=).

Operandendefinitionen	**Kommando-eingabe**	**an die Prozedur übergebene Parameterliste**

//ADD-OPER OP1,...,RES-OPER-N=*POS(2),
CONC-POS=2
//ADD-OPER OP2,...,RES-OPER-N=*POS(2)
CONC-POS=1
//ADD-OPER OP3,...,RES-OPER-N=*POS(1)
	/KDO X,Y,Z
	
	/KDO X,OP3=Z,
	OP2=Y
		('Z','YX')

//ADD-OPER OP1,...,RES-OPER-N=OP2,
CONC-POS=2
//ADD-OPER OP2,...,RES-OPER-N=*SAME
CONC-POS=1
//ADD-OPER OP3,...,RES-OPER-N=*POS(1)
	/KDO X,Y,Z
	
	/KDO X,OP3=Z,
	OP2=Y
		('Z',OP2='YX')

Ein Operand, für den es nur einen einzigen zulässigen Wert gibt, lässt sich so definieren, dass er zwar an die Prozedur übergeben wird, aber nicht in der Kommandosyntax erscheint (siehe ADD-OPERAND...,PRESENCE=*INTERNAL-ONLY). Steht dieser Operand als einziger Operand in einer Struktur, so ist auch die Struktur in der Kommandosyntax unsichtbar.

Das lässt sich u.a. dazu nutzen, an die Prozedur Sprungziele zu übergeben, zu denen in Abhängigkeit von Eingabealternativen in der Prozedur verzweigt wird (siehe „Beispiel 1: Kommando zum Assemblieren“ auf Seite 93).

Andererseits lässt sich ein Operand, der z.B. nur zur Strukturierung der Kommandosyntax, aber nicht zur Implementierung benötigt wird, bei der Operandenübergabe an die Prozedur unterdrücken (siehe ADD-OPERAND...,PRESENCE=*EXTERNAL-ONLY). Mit ADD-VALUE...,VALUE=<c-string>(...,OUTPUT=...,...),... können Sie festlegen, dass SDF definierte Einzelwerte vor der Übergabe an die Prozedur in andere Werte umwandelt oder deren Übergabe unterdrückt. Beispielsweise können Sie festlegen, dass SDF statt des für die Eingabe definierten Werts ’YES’ den Wert ’ISD’ an die Prozedur übergibt (siehe „Beispiel 1: Kommando zum Assemblieren“ auf Seite 93).

Mit ADD-VALUE...,OUTPUT=*NORMAL(STRING-LITERALS=...) können Sie festlegen, ob SDF einen Operandenwert vor der Übergabe an die Prozedur umcodiert (<c-string> zu <x-string> oder umgekehrt).

Mit ADD-VALUE...,STRUCTURE=*YES(SIZE=...,...),... bestimmen Sie, ob in der MINIMUM- und MEDIUM-Stufe des geführten Dialogs eine Struktur in den Operandenfragebogen integriert wird oder ob SDF für sie einen eigenen Unterfragebogen ausgibt.

Normalerweise können Sie bei der Definition eines Kommandos weitgehend die Default-Werte der ADD-Anweisungen benutzen. Diese Anweisungen bieten Ihnen aber auch viele Möglichkeiten, die Kommandodefinition auf Ihre ganz speziellen Erfordernisse auszurichten. Einzelheiten dazu finden Sie in der Beschreibung der ADD-Anweisungen.

4.3 Anweisungen definieren und implementieren

4.3.1 Beispiel: Programm zum Kopieren von Dateien

Es soll ein Programm erstellt werden, das ISAM- und SAM-Dateien kopiert. Dabei soll es möglich sein, Folgendes zu ändern:

– die Zugriffsmethode

– die Satzlänge

– den ISAM-Schlüssel.

Das Programm soll als Hauptprogramm ablaufen.

Falls die zu bearbeitende Datei durch ein Kennwort geschützt ist, soll dieses dunkelgesteuert eingegeben werden können. Fehlt das Kennwort, so soll es in einem Fehlerdialog angefordert werden.

Das Programm hat neben den SDF-Standardanweisungen die folgende Anweisung:

COPY-FILE

FROM-FILE = <filename 1..54>

,TO-FILE = <filename 1..54 without-gen-vers>(...)

<filename 1..54 without-gen-vers>(...)

ACCESS-METHOD = ***SAME** / ***ISAM**(...) / ***SAM**

***ISAM**(...)

KEY-LENGTH = ***STD** / <integer 1..50>

,RECORD-SIZE = ***SAME** / ***VARIABLE** / <integer 1..2048>

,PASSWORD = ***NONE** / <c-string 1..4> / ***SECRET-PROMPT**

Programm in der Benutzersyntaxdatei definieren

Das Programm KOP wird in der Benutzersyntaxdatei SDF.KOP.SYNTAX definiert:

```
/set-logon-parameters sdfusr, ...  ————————————————————————————  (1)
 .
 .
/start-sdf-a  ————————————————————————————————————————————————  (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%//open-syntax-file sdf.kop.syntax,,*create  ———————————————————  (3)
%//add-program kop  ————————————————————————————————————————————  (4)
%//add-stmt name=copy-file,prog=kop,intern-name=copyfi,stmt-version=1  —  (5)
%//add-oper from-file,res-oper-name=*pos(1)  ———————————————————  (6)
%//add-val *filename  ——————————————————————————————————————————  (7)
%//add-oper to-file,res-oper-name=*pos(2)  ——————————————————————  (8)
%//add-val *filename(gen=*n,vers=*n),structure=*y(siz=*small)  ————  (9)
%//add-oper access-method,default='same',res-oper-name=*pos(3)  ———— (10)
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) SDF-A wird geladen und gestartet.

(3) Die Benutzersyntaxdatei SDF.KOP.SYNTAX wird erzeugt und geöffnet.

(4) Das Programm KOP wird definiert. Als INTERNAL-NAME übernimmt SDF-A standardmäßig die ersten drei Buchstaben KOP.

(5) Der Kopf der zum Programm KOP gehörenden Anweisung COPY-FILE wird definiert. Ihr INTERNAL-NAME ist COPYFI. Sie erhält die Versionsnummer 1.

(6) Der erste Operand der Anweisung COPY-FILE wird definiert. Sein Name ist FROM-FILE. Im Operanden-Array des Übergabebereichs steht er an erster Stelle.

(7) Es wird definiert, dass der Wert des Operanden FROM-FILE vom Datentyp FILENAME sein muss.

(8) Der zweite Operand der Anweisung COPY-FILE wird definiert. Sein Name ist TO-FILE. Im Operanden-Array des Übergabebereichs steht er an zweiter Stelle.

(9) Es wird definiert, dass der Wert des Operanden TO-FILE vom Datentyp FILENAME sein muss. Es darf weder eine Generationsnummer noch eine Versionsbezeichnung angegeben werden. FILENAME leitet eine Struktur ein.

(10) Der erste Operand in der Struktur FILENAME wird definiert. Sein Name ist ACCESS-METHOD. Sein Default-Wert ist SAME. Im Operanden-Array des Übergabebereichs steht er an dritter Stelle.

```
%//add-val *keyw,val='same' ------------------------------------------------ (11)
%//add-val *keyw,val='isam',struct=*y --------------------------------------- (12)
%//add-oper key-length,default='std',res-oper-name=*pos(4) ------------------ (13)
%//add-val *keyw,val='std' -------------------------------------------------- (14)
%//add-val *integer(1,50) --------------------------------------------------- (15)
%//close-structure ---------------------------------------------------------- (16)
%//add-val *keyw,val='sam' -------------------------------------------------- (17)
%//add-oper record-size,default='same',res-oper-name=*pos(5) ---------------- (18)
%//add-val *keyw,val='same' -------------------------------------------------- (19)
%//add-val *keyw,val='variable' ---------------------------------------------- (20)
%//add-val *integer(1,2048) -------------------------------------------------- (21)
%//close-struct ------------------------------------------------------------- (22)
```

(11) Es wird definiert, dass das Schlüsselwort SAME ein zulässiger Wert des Operanden ACCESS-METHOD ist.

(12) Es wird definiert, dass das Schlüsselwort ISAM ein zulässiger Wert des Operanden ACCESS-METHOD ist. ISAM leitet eine Struktur ein.

(13) Der erste Operand in der Struktur ISAM wird definiert. Sein Name ist KEY-LENGTH. Sein Default-Wert ist STD. Im Operanden-Array des Übergabebereichs steht er an vierter Stelle.

(14) Es wird definiert, dass das Schlüsselwort STD ein zulässiger Wert des Operanden KEY-LENGTH ist.

(15) Es wird definiert, dass der Wert des Operanden KEY-LENGTH vom Datentyp INTEGER sein kann. Als untere Grenze wird 1 und als obere 50 definiert.

(16) Die gerade bearbeitete Struktur ISAM wird geschlossen.

(17) Es wird definiert, dass das Schlüsselwort SAM ein zulässiger Wert des Operanden ACCESS-METHOD ist.

(18) Der zweite Operand in der Struktur FILENAME wird definiert. Sein Name ist RECORD-SIZE. Sein Default-Wert ist SAME. Im Operanden-Array des Übergabebereichs steht er an fünfter Stelle.

(19) Es wird definiert, dass das Schlüsselwort SAME ein zulässiger Wert des Operanden RECORD-SIZE ist.

(20) Es wird definiert, dass das Schlüsselwort VARIABLE ein zulässiger Wert des Operanden RECORD-SIZE ist.

(21) Es wird definiert, dass der Wert des Operanden RECORD-SIZE vom Datentyp INTEGER sein kann. Als untere Grenze wird 1 und als obere 2048 definiert.

(22) Die gerade bearbeitete Struktur FILENAME wird geschlossen.

```
%//add-oper password,default='none',secret=*y,res-oper-name=*pos(6) ——— (23)
%//add-val *keyw,val='none' ————————————————————————————————————— (24)
%//add-val *c-string(1,4) ——————————————————————————————————————— (25)
%//add-val *keyw,val='secret-prompt',out=*secret ———————————————————— (26)
%//close-cmd ———————————————————————————————————————————————————— (27)
%//show object=*program(name=kop),size=*max ————————————————————————— (28)
KOP
COPY-FILE
   FROM SDF.KOP.SYNTAX (USER)
     FROM-FILE =
          filename 1..54
     TO-FILE =
          filename 1..54_without-generation-version()
          STRUCTURE: filename
               ACCESS-METHOD = *SAME
                    *SAME or *ISAM() or *SAM
                    STRUCTURE: *ISAM
                         KEY-LENGTH = *STD
                              *STD or integer_1..50
               RECORD-SIZE = *SAME
                    *SAME or *VARIABLE or integer_1..2048
     PASSWORD =
          *NONE or c-string_1..4 or *SECRET-PROMPT -DEFAULT-: NONE

%//end
 .
 .
%//exit-job
```

(23) Der dritte globale Operand der Anweisung COPY-FILE wird definiert. Sein Name ist PASSWORD. Sein Default-Wert ist NONE. Er ist als geheimer Operand definiert. Im Operanden-Array des Übergabebereichs steht er an sechster Stelle.

(24) Es wird definiert, dass das Schlüsselwort NONE ein zulässiger Wert des Operanden PASSWORD ist.

(25) Es wird definiert, dass der Wert des Operanden PASSWORD vom Datentyp C-STRING sein kann. Als minimale Länge ist 1 definiert, als maximale 4.

(26) Es wird definiert, dass das Schlüsselwort SECRET-PROMPT ein zulässiger Wert des Operanden PASSWORD ist. Es wird nicht an die Implementierung übergeben, sondern nach seiner Eingabe fordert SDF die dunkelgesteuerte Eingabe eines Werts für PASSWORD an.

(27) Die Definition der Anweisung COPY-FILE wird abgeschlossen.

(28) Die Definition des Programms KOP wird mit der einzigen zugehörigen Anweisung in der ausführlichsten Form ausgegeben.

Nach Analyse der Anweisung COPY-FILE schreibt SDF folgende Informationen in den Übergabebereich. Die in runde Klammern eingeschlossenen Angaben schreibt SDF-A nur im Fall ACCESS-METHOD=*ISAM.

Byte	Länge in Byte	Bezeichnung	Wert
0 bis 7	8	Standardheader	
8 bis 11	4	Länge des Übergabebereichs	
12 bis 19	8	interner Anweisungsname	C'COPYFI'
20 bis 23	4	reserviert	
24 bis 26	3	Version der Anweisung	C'001'
27 bis 35	9	reserviert	
36 bis 37	2	Anzahl der Positionen im Operanden-Array	6
38 bis 39	2	reserviert	
40	1	Zusatzinformation für FROM-FILE	B'1... ....'
41	1	Typbeschreibung für FROM-FILE	11
42	1	Globale Syntaxattribute für FROM-FILE (immer 0, da Wildcards nicht erlaubt sind)	B'.... ....'
43	1	Syntaxattribute für Datentyp <filename>, wenn:	
		– Cat-ID vorhanden	B'1... ....'
		– User-ID vorhanden	B'.1.. ....'
		– Dateigeneration vorhanden	B'..1. ....'
		– Version vorhanden	B'...1 ....'
		– Datei temporär ist	B'.... 1...'
44 bis 47	4	Absolutadresse des Werts von FROM-FILE	aaaa
48	1	Zusatzinformation für TO-FILE	B'1... ....'
49	1	Typbeschreibung für TO-FILE	11
50	1	Globale Syntaxattribute für TO-FILE (immer 0, da Wildcards nicht erlaubt sind)	B'.... ....'
51	1	Syntaxattribute für Datentyp <filename>, wenn:	
		– Cat-ID vorhanden	B'1... ....'
		– User-ID vorhanden	B'.1.. ....'
		– Datei temporär ist	B'.... 1...'
52 bis 55	4	Absolutadresse des Werts von TO-FILE	aaaa
56	1	Zusatzinformation für ACCESS-METHOD	B'1... ....'
57	1	Typbeschreibung für ACCESS-METHOD	22
58	1	reserviert	
59	1	reserviert	
60 bis 63	4	Absolutadresse des Werts von ACCESS-METHOD	aaaa

Fortsetzung ➟

Byte	Länge in Byte	Bezeichnung	Wert
64	1	Zusatzinformation für KEY-LENGTH	B'0... ....' oder B'1... ....'
65	1	Typbeschreibung für KEY-LENGTH	(2 oder 22)
66	1	reserviert	
67	1	reserviert	
68 bis 71	4	Absolutadresse des Werts von KEY-LENGTH	(aaaa)
72	1	Zusatzinformation für RECORD-SIZE	B'1... ....'
73	1	Typbeschreibung für RECORD-SIZE	2 oder 22
74	1	reserviert	
75	1	reserviert	
76 bis 79	4	Absolutadresse des Werts von RECORD-SIZE	aaaa
80	1	Zusatzinformation für PASSWORD	B'1... ....'
81	1	Typbeschreibung für PASSWORD	2 oder 22
82	1	reserviert	
83	1	Syntaxattribute für Datentyp <c-string>: – wenn Kennwort Hochkommas enthält	B'1... ....'
84 bis 87	4	Absolutadresse des Werts von PASSWORD	aaaa
	2	Längenangabe	II
	2	reserviert	
	≤ 54	Wert von FROM-FILE	xxx
	2	Längenangabe	II
	2	reserviert	
	≤ 54	Wert von TO-FILE	xxx
	2	Längenangabe	II
	2	reserviert	
	≤ 4	Wert von ACCESS-METHOD	xxx
	(2)	(Längenangabe)	(II)
	(2)	(reserviert)	
	(≤ 4)	(Wert von KEY-LENGTH)	(xxx)
	2	Längenangabe	II
	2	reserviert	
	≤ 8	Wert von RECORD-SIZE	xxx
	2	Längenangabe	II
	2	reserviert	
	≤ 4	Wert von PASSWORD	xxx

Die Anweisung COPY-FILE belegt im Übergabebereich maximal 240 Byte. Der Übergabebereich muss deshalb mindestens 240 Byte groß sein.

Programm erstellen

Im Folgenden ist das Programm KOP wiedergegeben:

```
KOP      START
         TITLE 'Example of program using SDF macros'
*-----------------------------------------------------------------------*
* The program reads and corrects the statement COPY-FILE with SDF,
* then executes it.
* The field identifiers used by this program are to be found in the
* SDF macros CMDTA for the transfer area, CMDMEM for the status.
*-----------------------------------------------------------------------*
R0       EQU      0
R1       EQU      1
R2       EQU      2
R3       EQU      3
R4       EQU      4
R5       EQU      5
R6       EQU      6
R7       EQU      7
R8       EQU      8
R9       EQU      9
R10      EQU      10
R11      EQU      11
R12      EQU      12
R13      EQU      13
R14      EQU      14
R15      EQU      15
*-----------------------------------------------------------------------*
FRFILE#  EQU      1                    position of operand FROM-FILE
TOFILE#  EQU      2                                        TO-FILE
ACCESS#  EQU      3                                        ACCESS-METH
KEYLEN#  EQU      4                                        KEY-LENGTH
RECSIZ#  EQU      5                                        RECORD-SIZE
PASSWR#  EQU      6                                        PASSWORD
         CMDANALY ,                    SDF return codes
TAD      CMDTA    MF=D                 SDF transfer area
RSTD     CMDRST   MF=D                 Read statements
CSTD     CMDCST   MF=D                 Correct statement
*-----------------------------------------------------------------------*
*        Register usage
*        R2       SDF output area for CMDSTA, CMDRST, CMDCST.
*        R3       current character in filename analysis
*        R3       current field in SDF output area
*
*        R5       help register
*        R10,R11  base registers
*        R15      return code
```

```
*-------------------------------------------------------------------*
KOP      CSECT    ,
BEGIN    BALR     R10,R0
         USING    *,R10,R11
         B        FOO
SLICE#   DC       F'4096'
FOO      LR       R11,R10
         A        R11,SLICE#
*-------------------------------------------------------------------*
*        Check if user syntax file is activated.
*-------------------------------------------------------------------*
         CMDMEM   D,P=XMD               Layout for CMDSTA output
*
KOP      CSECT    ,
*
         CMDSTA   OUTAREA=STA#OUT       Get SDF options
*
         USING    XMDMEM,R2
         LA       R2,STA#OUT
         LA       R3,XMDUSER
         CLI      0(R3),':'                Check catid
         BNE      NOCATID
CATIDL   LA       R3,1(R3)
         CLI      0(R3),':'
         BNE      CATIDL
         LA       R3,1(R3)
NOCATID  CLI      0(R3),'$'                Check userid
         BNE      NOUSERID
USERIDL  LA       R3,1(R3)
         CLI      0(R3),'.'
         BNE      USERIDL
         LA       R3,1(R3)
NOUSERID CLC      0(USF#L,R3),USF#NAME     Check user syntax file
         BNE      FC#ERROR
         DROP     R2
*-------------------------------------------------------------------*
*        Read SDF statement.
*-------------------------------------------------------------------*
READSTMT DS       0Y
         MVI      OWNFLAGS,0
         USING    RSTD,1            from CMDRST
         USING    TAD,R2            from CMDTA
         LA       R1,RSTPL
         LA       R2,OUTPUT
*
         CMDRST   MF=E,PARAM=RSTPL
*
         CLI      CMDRMR1,CMDRSUCCESSFUL  no error, continue...
```

```
         BE       FO1
         CLI      CMDRMR1,CMDREOF         EOF reached
         BE       FC#EOF
         CLI      CMDRMR1,CMDREND_STMT    //END was read
         BE       FC#END
         B        FC#ERROR              otherwise error...
*--------------------------------------------------------------------*
*        Evaluate structured-description
*--------------------------------------------------------------------*
FO1      DS       OH
         CLC      CMDINTN,COPYFILE      is this //COPY-FILE ?
         BNE      FC#ERROR              it can only be //COPY-FILE !
*--------------------------------------------------------------------*
*        operand FROM-FILE
*--------------------------------------------------------------------*
         LA       R3,CMDMAIN+(FRFILE#-1)*CMDHEAL   FROM-FILE
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(from-file value)
         MVI      FRFILE,' '
         MVC      FRFILE+1(L'FRFILE-1),FRFILE
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         BCTR     R5,RO
         EX       R5,EX#MVC1            mvc with l=R5
         B        FO2                   skip ex#mvc1...
EX#MVC1  MVC      FRFILE(1),CMDAVAL
*--------------------------------------------------------------------*
*        operand PASSWORD
*--------------------------------------------------------------------*
FO2      DS       OH
         LA       R3,CMDMAIN+(PASSWR#-1)*CMDHEAL   PASSWORD
         USING    CMDODES,R3
         CLI      CMDOTYP,CMDCSTR       C-string input ?
         BNE      NO#PASS               no: no password input
*        copy input password into own field
         MVI      PASSWR,' '
         MVC      PASSWR+1(L'PASSWR-1),PASSWR
         USING    CMDHEAD,R3            operand description
         L        R3,CMDOPTR            A(password value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         BCTR     R5,RO
         EX       R5,EX#MVC2            mvc with l=R5
         B        FC#OPEN               skip ex#mvc2...
EX#MVC2  MVC      PASSWR(1),CMDAVAL
NO#PASS  DS       OH
         OI       OWNFLAGS,NOPASS
FC#OPEN  DS       OH
```

```
         CLC      PASSWR,QUESTION       is password = '????'
         BE       PASS#ER               test corstmt part.
*        open of file FROM-FILE
*        ...
*        if error at open of FROM-FILE:
*        B        PASS#ER
*--------------------------------------------------------------------*
*        operand TO-FILE
*--------------------------------------------------------------------*
         LA       R3,CMDMAIN+(TOFILE#-1)*CMDHEAL   TO-FILE
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC       operand input ?
         BZ       FC#ERROR              no: error
         MVI      TOFILE,' '
         MVC      TOFILE+1(L'TOFILE-1),TOFILE
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(to-file value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         BCTR     R5,R0
         EX       R5,EX#MVC3            mvc with l=R5
         B        F03                   skip ex#mvc3...
EX#MVC3  MVC      TOFILE(1),CMDAVAL
*--------------------------------------------------------------------*
*        operand ACCESS-METHOD
*--------------------------------------------------------------------*
F03      DS       0H
         LA       R3,CMDMAIN+(ACCESS#-1)*CMDHEAL   ACCESS-METHOD
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC        operand input ?
         BZ       FC#ERROR              no: error
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(access-method value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         CH       R5,THREE              sam?
         BE       I#SAM
         CLI      CMDAVAL,'I'           isam?
         BE       I#SAM
*        access-method = same
         OI       OWNFLAGS,ACCSAME
         MVI      ACCESS,'X'
         B        F04                   skip sam/isam
*        access-method = sam / isam
I#SAM    DS       0H
         MVC      ACCESS,CMDAVAL        S:sam / I:isam
*--------------------------------------------------------------------*
*        operand KEY-LENGTH
```

```
*-------------------------------------------------------------------*
F04      DS       0H
         LA       R3,CMDMAIN+(KEYLEN#-1)*CMDHEAL   KEY-LENGTH
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC        operand input ?
         BZ       F05                   no: not isam, next operand.
         CLI      CMDOTYP,CMDINT
         BE       KEYINT
*        key-length = std : keylen := 8
         MVI      KEYLEN,8
         B        F05
*        key-length = <integer_1..50>
KEYINT   DS       0H
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(key-length value)
         USING    CMDOVAL,R3            value field
         MVC      KEYLEN,CMDAVAL+3
         B        F05
*-------------------------------------------------------------------*
*        operand RECORD-SIZE
*-------------------------------------------------------------------*
F05      DS       0H
         LA       R3,CMDMAIN+(RECSIZ#-1)*CMDHEAL   RECORD-SIZE
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC        operand input ?
         BZ       FC#ERROR              no: error
         CLI      CMDOTYP,CMDINT
         BE       RECINT
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(record-size value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         CH       R5,FOUR
         BNE      F051
*        record-size = same
         OI       OWNFLAGS,RECSAME
         B        F06
F051     DS       0H
*        record-size = variable
         OI       OWNFLAGS,RECVAR
         B        F06
*        record-size = <integer_1..2048>
RECINT   DS       0H
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(record-size value)
         USING    CMDOVAL,R3            value field
         MVC      RECSIZ,CMDAVAL+2
F06      DS       0H
```

```
*-------------------------------------------------------------------*
*   ...  Copy file ...
*        Output of copied values for test purpose (even password!)
*-------------------------------------------------------------------*
         WROUT    MESS1,FC#END
         WROUT    MESS2,FC#END
         TM       OWNFLAGS,NOPASS
         BZ       WMESS3
         WROUT    MESS11,FC#END
         B        WMESS4
WMESS3   DS       OH
         WROUT    MESS3,FC#END
WMESS4   DS       OH
         TM       OWNFLAGS,ACCSAME
         BZ       WMESS5
         WROUT    MESS4,FC#END          access given
         B        WMESS6
WMESS5   WROUT    MESS5,FC#END          access default
WMESS6   DS       OH
         TM       OWNFLAGS,KEYSTD
         BNZ      WMESS8
         UNPK     BUFF5(5),KEYHW(3)
         TR       BUFF5(4),CONVCHAR-XFO
         MVC      KEYCHAR,BUFF5
         WROUT    MESS6,FC#END
         B        WMESS7
WMESS8   WROUT    MESS8,FC#END
WMESS7   DS       OH
         TM       OWNFLAGS,RECSAME
         BNZ      WMESS9
         TM       OWNFLAGS,RECVAR
         BNZ      WMESS10
         UNPK     BUFF5(5),RECSIZ(3)
         TR       BUFF5(4),CONVCHAR-XFO
         MVC      RECCHAR,BUFF5
         WROUT    MESS7,FC#END
         B        REPEAT
WMESS9   WROUT    MESS9,FC#END
         B        REPEAT
WMESS10  WROUT    MESS10,FC#END
*
*        R E P E A T   READSTMT
*
REPEAT   DS       OH
         B        READSTMT
*-------------------------------------------------------------------*
*        Password handling routine
*-------------------------------------------------------------------*
```

```
PASS#ER  DS       OH
         LA       R3,CMDMAIN+(PASSWR#-1)*CMDHEAL   PASSWORD
         USING    CMDODES,R3
         OI       CMDGSTA,CMDERR        set operand erroneous
         LA       R1,CSTPL
         USING    CSTD,R1
         CMDCST   MF=M,MESSAGE=A(MESSAGE)
CORRSTMT CMDCST   MF=E,PARAM=CSTPL
         CLI      CMDCMR1,CMDCSUCCESSFUL  corstmt successful?
         BNE      READSTMT                no: new read...
         B        FO1                     repeat analysis...
*
*        Game over.
*
         B        FC#END
*-------------------------------------------------------------------------*
FC#ERROR DS       OH                    error handling routine
*        ...
FC#EOF   DS       OH                    EOF   handling routine
*        ...
*-------------------------------------------------------------------------*
FC#END   TERM
*-------------------------------------------------------------------------*
*        Parameter lists
*-------------------------------------------------------------------------*
RSTPL    CMDRST   MF=L,PROGRAM='KOP',OUTPUT=A(OUTPUT)
CSTPL    CMDCST   MF=L,INOUT=A(OUTPUT),MESSAGE=A(0)
*                                     given at execution
*-------------------------------------------------------------------------*
*        Used constants, variables and buffers
*-------------------------------------------------------------------------*
RC       DS       X                     return code byte
USF#NAME DC       'SDF.KOP.SYNTAX'      syntax file name
USF#L    EQU      *-USF#NAME            user syntax file length
COPYFILE DC       'COPYFI  '            //COPY-FILE internal name
QUESTION DC       '????'
THREE    DC       H'3'
FOUR     DC       H'4'
EIGHT    DC       H'8'
CONVCHAR DC       C'0123456789ABCDEF'
XFO      EQU      X'FO'
*        Message for CORSTMT
MESSAGE  DS       OF
         DC       Y(MSGEND-MESSAGE)
         DS       XL2
         DC       C'BITTE PASSWORT FUER EINGABEDATEI ANGEBEN'
MSGEND   EQU      *
*-------------------------------------------------------------------------*
```

```
*        Output fields for statement operands
*-------------------------------------------------------------------------*
MESS1    DS       OF
         DC       Y(END1-MESS1)
         DS       CL2
         DC       X'40'
         DC       C'FROM-FILE = '
FRFILE   DS       CL52' '               own from-file
END1     EQU      *
MESS2    DS       OF
         DC       Y(END2-MESS2)
         DS       CL2
         DC       X'40'
         DC       C'TO-FILE   = '
TOFILE   DS       CL52' '               own to-file
END2     EQU      *
MESS3    DS       OF
         DC       Y(END3-MESS3)
         DS       CL2
         DC       X'40'
         DC       C'PASSWORD  = '
PASSWR   DS       CL4                   own password
END3     EQU      *
MESS4    DS       OF
         DC       Y(END4-MESS4)
         DS       CL2
         DC       X'40'
         DC       C'ACCESS-METHOD IS SAME'
END4     EQU      *
MESS5    DS       OF
         DC       Y(END5-MESS5)
         DS       CL2
         DC       X'40'
         DC       C'ACCESS-METHOD = '
ACCESS   DS       X                     own access-method: Sam | Isam
END5     EQU      *
MESS6    DS       OF
         DC       Y(END6-MESS6)
         DS       CL2
         DC       X'40'
         DC       C'KEY-LENGTH = '
KEYCHAR  DS       CL4' '                key-length converted into hexa
END6     EQU      *
MESS7    DS       OF
         DC       Y(END7-MESS7)
         DS       CL2
         DC       X'40'
         DC       C'REC-SIZE   = '
```

```
RECCHAR  DS       CL4' '                rec-size   converted into hexa
END7     EQU      *
MESS8    DS       0F
         DC       Y(END8-MESS8)
         DS       CL2
         DC       X'40'
         DC       C'KEY-LENGTH IS STD'
END8     EQU      *
MESS9    DS       0F
         DC       Y(END9-MESS9)
         DS       CL2
         DC       X'40'
         DC       C'REC-SIZE IS SAME'
END9     EQU      *
MESS10   DS       0F
         DC       Y(END10-MESS10)
         DS       CL2
         DC       X'40'
         DC       C'REC-SIZE IS VARIABLE'
END10    EQU      *
MESS11   DS       0F
         DC       Y(END11-MESS11)
         DS       CL2
         DC       X'40'
         DC       C'PASSWORD IS NONE'
END11    EQU      *
KEYHW    DC       H'0'
         ORG      KEYHW
FIL1     DS       X
KEYLEN   DS       AL1                   own key-length
BUFF5    DS       CL5                   buffer for unpack
RECSIZ   DS       H                     own record-size
OWNFLAGS DS       X                     own flags
ACCSAME  EQU      X'80'                 access-method=same
KEYSTD   EQU      X'40'                 key-length=std
RECSAME  EQU      X'20'                 record-size=same
RECVAR   EQU      X'10'                           =variable
NOPASS   EQU      X'08'                 no password.
*-------------------------------------------------------------------*
STA#OUT  DS       0F                    CMDSTA output
         ORG      *+XMDMEML             length from CMDMEM
OUTPUT   CMDTA    MF=L,MAXLEN=400       CMDRST output
*-------------------------------------------------------------------*
         END      KOP
```

Programm übersetzen, binden und testen

Das gezeigte Quellprogramm KOP steht in der Datei KOP.SRC. Im Folgenden wird es übersetzt, gebunden und getestet.

```
/set-logon-parameters sdfusr,...  ————————————————————————————————  (1)
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2B00' OF '1998-04-24' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1990. ALL RIGHTS
RESERVED
%  ASS6010 V01.2B02 OF BS2000 ASSTRAN  READY
%//compile source=*lib-elem(lib=kop.lib,elem=kop.src),-                  ⎫
%//mod-lib=kop.lib(el=kop),listing=*parameters-                           ⎪
%//(source-print=*with-object-code(print-statements=*accept),-   ———     ⎬  (2)
%//macro-print=*std,min-mes-w=*note,cross-ref=*std,layout=*std,-          ⎪
%//output=*syslst),macro-lib=($.syslib.sdf.041,$loader.v140.syslib)      ⎭
%  ASS6011 ASSEMBLY TIME: 4223 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 3166 MSEC
%//end
%  ASS6012 END OF ASSTRAN
/start-binder
%  BND0500 BINDER VERSION 'V02.1A30' STARTED
%//start-llm-creation internal-name=kop  ————————————————————————————  (3)
%//include-module *lib(lib=kop.lib,elem=kop,type=r)  ————————————————  (4)
%//resolve-by-autolink $.syslib.sdf.045
%//resolve-by-autolink $loader.sysoml.bs2000-ga.14.0a
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) Das Programm muss mit ASSEMBH übersetzt werden, da es die neuen Makros CMDTA, CMDCST und CMDRST enthält. Das oben auszugsweise gezeigte Quellprogramm steht als Typ-S-Element mit dem Namen KOP.SRC in der Bibliothek KOP.LIB. Die SDF-Makros stehen in der Makrobibliothek $.SYSLIB.SDF.045. Das erzeugte Bindemodul soll unter dem Elementnamen KOP in der Bibliothek KOP.LIB abgelegt werden.

(3) Der Binder BINDER soll ein Bindelademodul (LLM) mit dem internen Namen KOP erzeugen.

(4) Der Binder soll das in Arbeitsschritt 2 erzeugte Bindemodul KOP einlesen und in das LLM einfügen.

```
%//save-llm lib=kop.lib,elem=kop ————————————————————————————— (5)
%  BND1501 LLM FORMAT: '1'
%//end
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
/mod-sdf-options synt-file=*add(sdf.kop.syntax) ———————————————— (6)
/start-exe *lib-elem(lib=kop.lib,elem=kop) ————————————————————— (7)
%  BLS0524 LLM 'KOP', VERSION ' ' OF '2001-10-10 14:00:22' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2001. ALL RIGHTS
RESERVED
%//mod-sdf-opt guid=*n ———————————————————————————————————————— (8)
%STMT:cop-f testdat.sv,pass=*secr ——————————————————————————— (9)
%ENTER SECRET OPERAND (PASSWORD):......
%  CMD0051 INVALID OPERAND 'TO-FILE'
%  CMD0099 MANDATORY OPERAND INVALID OR MISSING
%ENTER OPERANDS:
to-f=testdat.cop.1(isam,50)
%STMT:cop-f testdat.if,testdat.cop.2(acc-m=?) ————————————————— (10)
% CMD0090 EXPLANATION OF THE OPERAND ' TO-FILE=TESTDAT.COP.2:ACCESS-METHOD ':
%SAME or ISAM() or SAM -DEFAULT-: SAME
%ENTER OPERANDS:
%TESTDAT.SV,TESTDAT.COP.2(ACC-M=?)
acc-m=sam
%  CMD0051 INVALID OPERAND 'PASSWORD'
% BITTE PASSWORT FUER EINGABEDATEI ANGEBEN
%ENTER SECRET OPERAND (PASSWORD):......
```

(5) Das ablauffähige Bindelademodul soll unter dem Namen KOP (als Typ-L-Element) in der Bibliothek KOP.LIB abgelegt werden.

(6) Die Benutzersyntaxdatei SDF.KOP.SYNTAX, in der die Anweisungen für das Programm KOP definiert sind, wird aktiviert.

(7) Das Programm KOP wird gestartet. Das hier verwendete Kommando START-EXECUTABLE-PROGRAM steht ab BLSSERV V2.3 zur Verfügung (ggf. ist das Kommando START-PROGRAM mit RUN-MODE=*ADVANCED zu verwenden).

(8) Die Benutzerführung wird auf den Modus NO eingestellt.

(9) Fall 1: Die Anweisung COPY-FILE wird ohne den Pflichtoperanden TO-FILE eingegeben. Die Kennworteingabe soll dunkelgesteuert erfolgen. SDF fordert das Kennwort und den fehlenden Operanden an.

(10) Fall 2: Die Anweisung COPY-FILE wird ohne das erforderliche Kennwort eingegeben. Über den Operanden ACCESS-METHOD werden weitere Informationen angefordert. SDF liefert die Informationen und fordert zur Eingabe des Kennworts auf.

```
%STMT:cop-f testdat.iv,testdat.cop.3(?) ──────────────────────────────── (11)
```

```
PROGRAM  : KOP                    STATEMENT  : COPY-FILE
OPERANDS : FROM-FILE=TESTDAT.IV,TO-FILE=TESTDAT.COP.3

------------------------------------------------------------------------------
FROM-FILE            = TESTDAT.IV
TO-FILE              = TESTDAT.COP.3(ACCESS-METHOD=SAME,RECORD-SIZE=60)
PASSWORD             =
------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

```
PROGRAM  : KOP                    STATEMENT  : COPY-FILE
OPERANDS : FROM-FILE=TESTDAT.IV,TO-FILE=TESTDAT.COP.3(RECORD-SIZE=60)

------------------------------------------------------------------------------
FROM-FILE            = TESTDAT.IV
TO-FILE              = TESTDAT.COP.3(ACCESS-METHOD=SAME,RECORD-SIZE=60)
PASSWORD             = ....
------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL

ERROR:  BITTE PASSWORT FUER EINGABEDATEI ANGEBEN
```

(11) Fall 3: Die Anweisung COPY-FILE wird ohne das erforderliche Kennwort eingegeben. Durch das Fragezeichen wird über die Struktur, die der Dateiname der Ausgabedatei einleitet, weitere Information angefordert. Dies bewirkt den Übergang in den temporär geführten Dialog. SDF liefert im Operandenfragebogen die gewünschten Informationen und fordert danach das benötigte Kennwort an.

```
%STMT:?  ------------------------------------------------------------------ (12)

 PROGRAM  : KOP
 ------------------------------------------------------------------------------
 AVAILABLE STATEMENTS:

   1  COPY-FILE                              8  RESET-INPUT-DEFAULTS
   2  END                           (!)      9  RESTORE-SDF-INPUT
   3  EXECUTE-SYSTEM-CMD                    10  SHOW-INPUT-DEFAULTS
   4  HELP-MSG-INFORMATION                  11  SHOW-INPUT-HISTORY
   5  HOLD-PROGRAM                  (!)     12  SHOW-SDF-OPTIONS
   6  MODIFY-SDF-OPTIONS                    13  SHOW-STMT
   7  REMARK                                14  WRITE-TEXT









 ------------------------------------------------------------------------------
 NEXT =
 KEYS : F1=?  F3=*EXIT  F5=*REFRESH  F6=*EXIT-ALL  F9=REST-SDF-IN  F12=*CANCEL


** NORMALES PROGRAMMENDE **
%CMD:exit-job
```

(12) Mit dem Fragezeichen wird in den temporär geführten Dialog gewechselt. SDF gibt ein Menü aus, in dem die verfügbaren Anweisungen aufgelistet sind. Dazu gehören auch die Standardanweisungen. Die Anweisung STEP ist im Dialogbetrieb nicht verfügbar. Durch Eingabe der Nummer 2 wird die Anweisung END ausgewählt und das Programm beendet.

4.3.2 Hinweise zur Definition von Anweisungen

Standardanweisungen

Die Standardanweisungen (siehe Seite 129) sind nicht SDF-A-spezifisch. Sie werden automatisch systemweit für Benutzer-Programme angeboten, die ihre Anweisungen über SDF einlesen. In Ihrem Programm sind diese Anweisungen nicht zu implementieren. Die Information über eine eingelesene END- oder STEP-Anweisung erhält Ihr Programm über Register 15 (alte Schnittstelle) bzw. den Returncode im Standardheader (neue Schnittstelle).

Operandenposition

In dem Übergabebereich, in den SDF die analysierte Anweisung schreibt, sind die Operanden nur anhand ihrer Position im Operanden-Array identifizierbar (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff). Diese Position bestimmen Sie beim Definieren der Operanden (siehe ADD-OPERAND...,RESULT-OPERAND-NAME =*POS(...)).

Struktur

Beim Definieren des Struktureinleiters (siehe ADD-VALUE...,STRUCTURE= *YES(...,FORM=...)) und der in der Struktur stehenden Operanden (siehe ADD-OPERAND..., RESULT-OPERAND-LEVEL=) bestimmen Sie, ob die Stuktur im Übergabebereich erhalten bleibt oder linearisiert wird. Wenn Sie die Default-Werte der Anweisungen ADD-VALUE und ADD-OPERAND benutzen, wird die Struktur linearisiert. Mit ADD-VALUE..., STRUCTURE=*YES(SIZE=...,...),... bestimmen Sie, ob in der MINIMUM- und MEDIUM-Stufe des geführten Dialogs SDF eine Struktur in den übergeordneten Operandenfragebogen integriert oder dafür einen eigenen Unterfragebogen ausgibt. Standardmäßig wird die Struktur so definiert, dass SDF sie in den übergeordneten Fragebogen integriert.

Operand übergeben

Ein Operand, der z.B. nur zur Strukturierung der Anweisungssyntax, aber nicht zur Implementierung benötigt wird, lässt sich bei der Operandenübergabe an das Programm unterdrücken (siehe ADD-OPERAND...,PRESENCE=*EXTERNAL-ONLY).

Andererseits lässt sich ein Operand, für den es nur einen einzigen zulässigen Wert gibt, so definieren, dass er zwar an das Programm übergeben wird, aber nicht in der Anweisungssyntax erscheint (siehe ADD-OPERAND...,PRESENCE=*INTERNAL-ONLY). Steht dieser Operand als einziger Operand in einer Struktur, so ist auch die Struktur in der Anweisungssyntax unsichtbar.

Operandenwert definieren

Beim Definieren eines Operandenwertes bestimmen Sie, wie SDF den Wert in den Übergabebereich schreibt. Wenn Sie die Default-Werte der Anweisung ADD-VALUE benutzen, so schreibt SDF einen eingegebenen Wert unverändert in den Übergabebereich.

Mit ADD-VALUE...,VALUE=<c-string>(...,OUTPUT=...,...),... können Sie festlegen, dass SDF definierte Einzelwerte vor der Übergabe an das Programm in andere Werte umwandelt oder deren Übergabe unterdrückt. Beispielsweise können Sie festlegen, dass SDF statt des für die Eingabe definierten Werts 'YES' den Wert 'ISD' an das Programm übergibt (siehe Beispiel 1, Seite 93).

Den Default-Wert eines Operanden, der mit ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES definiert ist, kann Ihr Programm durch einen anderen Wert (Innen-Default) so ersetzen, dass im geführten Dialog der Innendefaultwert im Operandenfragebogen sichtbar ist (siehe Operand DEFAULT bei den Makros CMDCST, CMDRST und CMDTST).

Einen eingegebenen Operandenwert kann Ihr Programm durch einen anderen Wert ersetzen, wenn der zu ersetzende Wert mit ADD-VALUE...,VALUE=<c-string> (...,OVERWRITE-POSSIBLE=*YES),... definiert ist. Dabei ist Folgendes zu beachten:

– Der Innen-Default-Wert muss durch einen mit ADD-VALUE definierten Operandenwert abgedeckt sein.

– Je Operand kann nur maximal ein Innen-Default-Wert bestimmt werden. Allerdings kann mehreren Eingabealternativen eines Operanden derselbe Innen-Default-Wert zugeordnet werden.

– Ist einer Eingabealternative ein Innen-Default-Wert zugeordnet und hängt an einer anderen Eingabealternative eine Struktur, in der ebenfalls eine Innen-Defaultierung erfolgen soll, so erfordert das, dass die Struktur im Übergabebereich (und damit auch in der Umsetzbeschreibung für Innen-Default-Wert) linearisiert ist.

– In alternativen Strukturen ist Innen-Defaultierung nur möglich, wenn die Strukturen im Übergabebereich linearisiert sind (eine der alternativen Strukturen braucht nicht linearisiert zu sein).

– Stehen die zu defaultierenden Operanden in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten: Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit zu defaultierenden Operanden hängt. Auf der anderen Seite gibt der Benutzer ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit zu defaultierenden Operanden hängt. SDF versucht zunächst, die vom Benutzer eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen.
  Ist über den struktureinleitenden Wert keine eindeutige Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Benutzer den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

Normalerweise können Sie bei der Definition einer Anweisung weitgehend die Default-Werte der ADD-Anweisungen benutzen. Diese Anweisungen bieten Ihnen aber auch viele Möglichkeiten, die Anweisungsdefinition auf Ihre ganz speziellen Erfordernisse auszurichten. Einzelheiten dazu finden Sie in der Beschreibung der ADD-Anweisungen.

4.3.3 Hinweise zur Makroverwendung

SDF sorgt dafür, dass nur syntaktisch fehlerfreie Anweisungen eingelesen werden. Das schließt nicht aus, dass ein eingelesener Operandenwert in der konkreten Situation unzulässig ist. Wenn Ihr Programm die Zulässigkeit der eingelesenen Operandenwerte überprüft, können Sie für den Fehlerfall mit dem Makro CMDCST einen Korrekturdialog programmieren (siehe Beispiel, Seite 118).

Soll Ihr Programm als Unterprogramm ablaufen, dem das aufrufende Programm die Anweisungen übergibt, so können Sie mit dem Makro CMDTST die Anweisungsanalyse ebenfalls von SDF ausführen lassen.

Mit dem Operanden STMT (siehe Makros CMDRST und CMDTST) können Sie die Menge der zulässigen Anweisungen einschränken. Beispielsweise sind unmittelbar nach dem Start von SDF-A neben den Standardanweisungen nur die Anweisungen OPEN-SYNTAX-FILE, DEFINE-ENVIRONMENT und COMPARE-SYNTAX-FILE zulässig. Mit dem Operanden PREFER (siehe Makro CMDRST) können Sie veranlassen, dass im geführten Dialog statt des Anweisungsmenüs der Fragebogen für die von Ihrem Programm erwartete Anweisung ausgegeben wird. Das hindert den Benutzer allerdings nicht daran, eine andere als die erwartete Anweisung einzugeben. Beispielsweise erwartet SDF-A unmittelbar nach einer EDIT-Anweisung eine entsprechende MODIFY-Anweisung. Statt der erwarteten MODIFY-Anweisung können Sie aber auch eine andere SDF-A-Anweisung eingeben.

Mit dem Operanden SPIN (siehe Makro CMDRST) können Sie festlegen, dass im Stapelbetrieb oder in Prozeduren alle Anweisungen bis zur nächsten STEP- oder END-Anweisung überlesen werden.

Der Makro CMDSTA bewirkt, dass Ihr Programm Informationen über die aktivierten Syntaxdateien und die für die Kommando-/Anweisungseingabe geltenden Festlegungen erhält. Bei der Auswertung dieser Informationen ist zu beachten, dass die Benutzerkennung als Teil des Namens einer Syntaxdatei nur dann angegeben wird, wenn die Syntaxdatei unter einer fremden Kennung katalogisiert ist (siehe Beispiel).

5 SDF-A-Anweisungen

Die Anweisungen an SDF-A sind in von Fujitsu Siemens Computers gelieferten Syntaxdateien definiert. Ihre Eingabe erfolgt über den Kommandoprozessor SDF.

Die Standardanweisungen sind nicht SDF-A-spezifisch. Zu den Standardanweisungen gehören:

- END
- EXECUTE-SYSTEM-CMD
- HELP-MSG-INFORMATION
- HOLD-PROGRAM
- MODIFY-SDF-OPTIONS
- REMARK
- RESET-INPUT-DEFAULTS
- RESTORE-SDF-INPUT
- SHOW-INPUT-DEFAULTS
- SHOW-INPUT-HISTORY
- SHOW-SDF-OPTIONS
- SHOW-STMT
- STEP
- WRITE-TEXT

Sie werden systemweit für Benutzer-Programme angeboten, die ihre Anweisungen über SDF einlesen. Sobald Sie ein eigenes Benutzer-Programm definiert haben, ordnet SDF diese Anweisungen Ihrem Programm zu. Die Anweisungen EXECUTE-SYSTEM-CMD und HOLD-PROGRAM sind ab SDF V4.0 und BS2000/OSD-BC V2.0 verfügbar. Die Anweisungen HELP-MSG-INFORMATION, RESET-INPUT-DEFAULTS und SHOW-INPUT-DEFAULTS können Sie ab SDF V4.1 verwenden.

Jedem Benutzer steht der volle Funktionsumfang von SDF-A zur Verfügung. Bestimmte Operanden, die für die Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert sind, werden jedoch im geführten Dialog ausgeblendet und nicht beschrieben.

Die Anweisungen an SDF-A können mit Kommentaren versehen werden, die in Anführungszeichen (") eingeschlossen sein müssen.

5.1 Starten von SDF-A

SDF-A wird mit folgendem Kommando gestartet:

START-SDF-A	**Kurzname: SDF-A**
VERSION = ***STD** / <product-version> **,MON**JV = ***NONE** / <filename 1..54 without-gen-vers> **,CPU-LIM**IT = ***JOB-RE**ST / <integer 1..32767 *seconds*>	

VERSION =
Sind mehrere Versionen von SDF-A mit IMON installiert, so kann der Benutzer die Version auswählen, mit der er arbeiten möchte. Bei Angabe der Versionsbezeichnung mit Hochkommas kann der Buchstabe C vorangestellt werden (C-STRING-Syntax).
Falls das Produkt nicht mit IMON installiert wurde oder die angegebene Version nicht existiert, gilt VERSION=*STD.

VERSION = *STD
Aufruf der SDF-A-Version mit der höchsten Versionsnummer.

VERSION = <product-version>
Angabe der SDF-A-Version im Format mm.n[a[so]] (vgl. „product-version“ auf Seite 15).

MONJV =
Angabe einer Monitor-Jobvariablen zur Überwachung des SDF-A-Laufs.

MONJV = *NONE
Es wird keine Monitor-Jobvariable verwendet.

MONJV = <filename 1..54 without-gen-vers>
Name der zu verwendenden Jobvariablen.

CPU-LIMIT =
Maximale CPU-Zeit in Sekunden, die das Programm beim Ablauf verbrauchen darf.

CPU-LIMIT = *JOB-REST
Es soll die verbleibende CPU-Zeit für die Aufgabe verwendet werden.

CPU-LIMIT = <integer 1..32767 *seconds*>
Es soll nur die angegebene Zeit verwendet werden.

Unterbrechen und Fortsetzen von SDF-A

Wird SDF-A während der Bearbeitung einer Anweisung mit K2 unterbrochen, so kann es auf zwei Arten fortgesetzt werden.

– Wird SDF-A mit RESUME-PROGRAM fortgesetzt, so wird die unterbrochene Anweisung weiter bearbeitet.

– Wird SDF-A mit INFORM-PROGRAM (bis BS2000/OSD-BC V2.0: /SEND-MSG TO=*PROGRAM) wieder fortgesetzt, so wird die unterbrochene Anweisung abgebrochen und SDF-A steht für eine neue Anweisung bereit.

5.2 Funktionelle Übersicht

SDF-Standardanweisungen

Die ausführliche Beschreibung der SDF-Standardanweisungen finden Sie im Benutzerhandbuch „Einführung in die Dialogschnittstelle SDF“ [1].

END	SDF-A-Lauf beenden
EXECUTE-SYSTEM-CMD	Systemkommando während des SDF-A-Laufs ausführen
HELP-MSG-INFORMATION	Text einer Systemmeldung auf SYSOUT ausgeben
HOLD-PROGRAM	SDF-A-Lauf anhalten
MODIFY-SDF-OPTIONS	Benutzersyntaxdatei aktivieren oder deaktivieren und SDF-Einstellungen ändern
REMARK	Kommentare in die Anweisungsfolge schreiben und die Ausführung eines Programms dokumentieren
RESET-INPUT-DEFAULTS	taskspezifische Default-Werte zurücksetzen
RESTORE-SDF-INPUT	Vorher eingegebene Anweisungen oder Kommandos am Bildschirm wiederanzeigen
SHOW-INPUT-DEFAULTS	taskspezifische Default-Werte anzeigen
SHOW-INPUT-HISTORY	Inhalt des Eingabepuffers anzeigen
SHOW-SDF-OPTIONS	taskspezifische Information über aktivierte Syntaxdateien und die Festlegungen für die Anweisungseingabe und -verarbeitung ausgeben
SHOW-STMT	Syntax einer Anweisung anzeigen
STEP	Wiederaufsetzpunkt in einer Folge von SDF-A-Anweisungen definieren
WRITE-TEXT	einen bestimmten Text nach SYSOUT oder SYSLST ausgeben

Syntaxdateien bearbeiten, erstellen und vergleichen

COMPARE-SYNTAX-FILE	Objekte in zwei Syntaxdateien miteinander vergleichen
COPY	Inhalte einer Syntaxdatei in die geöffnete Syntaxdatei kopieren
EDIT	auf ein Objekt der geöffneten Syntaxdatei positionieren
OPEN-SYNTAX-FILE	Syntaxdatei zur Bearbeitung mit SDF-A öffnen
REMOVE	Objekte in der geöffneten Syntaxdatei löschen
RESTORE	Objekte der Syntaxdatei rekonstruieren
SET-GLOBALS	allgemeine Festlegungen über die Kommandoeingabe und -verarbeitung ändern
DEFINE-ENVIRONMENT	Syntaxdatei-Format und Version festlegen

Objekte in der geöffneten Syntaxdatei definieren

ADD-CMD	Kommando definieren
ADD-DOMAIN	Anwendungsbereich definieren
ADD-OPERAND	Operand definieren
ADD-PROGRAM	Programm definieren
ADD-STMT	Anweisung definieren
ADD-VALUE	Operandenwert definieren

Objekte in der geöffneten Syntaxdatei verändern

MODIFY-CMD	Definition eines Kommandos ändern
MODIFY-CMD-ATTRIBUTES	Kommandoattribute ändern
MODIFY-DOMAIN	Definition eines Anwendungsbereichs ändern
MODIFY-OPERAND	Definition eines Operanden ändern
MODIFY-PROGRAM	Definition eines Programms ändern
MODIFY-STMT	Definition einer Anweisung ändern
MODIFY-STMT-ATTRIBUTES	Anweisungsattribute ändern
MODIFY-VALUE	Definition eines Operandenwerts ändern

Bearbeitung von Kommandos und Strukturen abschließen

CLOSE-CMD-OR-STMT	Kommando- oder Anweisungsdefinition schließen
CLOSE-STRUCTURE	Strukturen schließen

Anzeigefunktionen

SHOW	Inhalte einer geöffneten Syntaxdatei auf SYSOUT oder SYSLST ausgeben
SHOW-CORRECTION-INFORMATION	Korrekturinformationen aus der geöffneten Syntaxdatei ausgeben (nur für Diagnosezwecke)
SHOW-STATUS	Name der geöffneten Syntaxdatei ausgeben

5.3 Beschreibung der Anweisungen

ADD-CMD
Kommando definieren

Mit der Anweisung ADD-CMD definieren Sie in der bearbeiteten Syntaxdatei die globalen Eigenschaften für ein Kommando, wie

– Kommando-Name

– Hilfetext

– Anwendungsbereich (DOMAIN), zu dem das Kommando gehören soll.

Dieses Kommando ist anschließend „aktuelles“ Objekt. Mit Ausnahme des RESULT-INTERNAL-NAME müssen alle für das Kommando vergebenen Namen eindeutig von anderen Namen des gesamten Kommandovorrats zu unterscheiden sein.

BS2000-Kommandos (durch System-Module implementiert) können Sie nur in einer Gruppen- oder Systemsyntaxdatei definieren.

Die Definitionen der zu dem Kommando gehörenden Operanden und Operandenwerte werden nicht mit der Anweisung ADD-CMD, sondern mit den Anweisungen ADD-OPERAND bzw. ADD-VALUE oder COPY in die Syntaxdatei eingebracht.

(Teil 1 von 3)

ADD-CMD

NAME = <structured-name 1..30>
,**INTERN**AL-**NAME** = ***STD** / <alphanum-name 1..8>
,**RES**ULT-**INTERN**AL-**NAME** = ***SAME** / <alphanum-name 1..8>
,**STAND**ARD-**NAME** = ***NAME** / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..30>
,**ALIAS-NAME** = ***NO** / list-poss(2000): <structured-name 1..30>
,**MIN**IMAL-**ABBREV**IATION = ***NO** / <structured-name 1..30>
,**HELP** = ***NO** / list-poss(2000): <name 1..1>(...)
<name 1..1>(...)
TEXT = <c-string 1..500 with-low>
,**DOM**AIN = ***NO** / list-poss(2000): <structured-name 1..30>

Fortsetzung ➠

(Teil 2 von 3)

```
,IMPLEMENTOR = *PROCEDURE(...) / *TPR(...) / *APPLICATION(...) / *BY-TPR(...)
    *PROCEDURE(...)
        NAME = <c-string 1..280> / *BY-IMON(...)
            *BY-IMON(...)
                LOGICAL-ID = <filename 1..30 without-cat-user-gen-vers>
                ,INSTALLATION-UNIT = <text 1..30 without-sep>
                ,VERSION = *STD / <product-version>
                ,DEFAULT-PATH-NAME = <filename 1..54>
                ,ELEMENT = *NONE / <composed-name 1..64>
        ,CALL-TYPE = *CALL-PROCEDURE / *INCLUDE-PROCEDURE / *ENTER-PROCEDURE
        ,CALL-OPTIONS = *NONE / <c-string 1..1800 with-low>
        ,UNLOAD-PROGRAM = *YES / *NO
    *TPR(...)
        ENTRY = <name 1..8>
        ,INTERFACE = *ASS / *SPL / *ISL(...)
            *ISL(...)
                VERSION = 1 / <integer 1..2>
        ,CMD-INTERFACE = *STRING(...) / *TRANSFER-AREA(...) / *NEW(...)
            *STRING(...)
                OUT-CMD-NAME = *SAME / <structured-name 1..30>
            *TRANSFER-AREA(...)
                MAX-STRUC-OPERAND = *STD / <integer 1..3000>
                ,CMD-VERSION = *NONE / <integer 1..999>
            *NEW(...)
                MAX-STRUC-OPERAND = *STD / <integer 1..3000>
        ,LOGGING = *BY-SDF / *BY-IMPLEMENTOR
        ,INPUT-FORM = *NONE / *INVARIANT / *STANDARD
        ,SCI = *NO / *YES
    *APPLICATION(...)
        LOGGING = *BY-SDF / *BY-IMPLEMENTOR
    *BY-TPR(...)
        TPR-CMD = <structured-name 1..30>
,REMOVE-POSSIBLE = *YES / *NO
```

Fortsetzung ➠

(Teil 3 von 3)

,**DIAL**OG-**ALLOW**ED = **_*YES_**(...) / ***NO**(...)

***YES**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

***NO**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

,**DIAL**OG-**PROC-ALLOW**ED = **_*YES_**(...) / ***NO**(...)

***YES**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

***NO**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

,**GUID**ED-**ALLOW**ED = **_*YES_**(...) / ***NO**(...)

***YES**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

***NO**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

,**BATCH-ALLOW**ED = ***YES**(...) / ***NO**(...)

***YES**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

***NO**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

,**BATCH-PROC-ALLOW**ED = ***YES**(...) / ***NO**(...)

***YES**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

***NO**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

,**CMD-ALLOW**ED = **_*YES_**(...) / ***NO**(...)

***YES**(...)
| **UNLOAD** = **_*NO_** / ***YES**
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

***NO**(...)
| **PRIV**ILEGE = **_*SAME_** / list-poss(64): <structured-name 1..30>

,**NEXT-IN**PUT = **_*CMD_** / ***STMT** / ***DATA** / ***ANY**

,**PRIV**ILEGE = **_*ALL_** / ***EXC**EPT(...) / list-poss(64): <structured-name 1..30>

***EXC**EPT(...)
| **EXC**EPT-**PRIV**ILEGE = list-poss(64): <structured-name 1..30>

NAME = <structured-name 1..30>
(externer) Kommandoname, der bei der Kommandoeingabe anzugeben ist. Der Benutzer kann ihn im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Kommandoeingabe abkürzen.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
interner Kommandoname. Dieser kann nicht verändert werden. SDF identifiziert ein Kommando, das in mehreren Syntaxdateien mit unterschiedlichem externen Namen definiert ist, mithilfe des internen Kommandonamens als dasselbe Kommando. Wenn das Kommando durch System-Module implementiert ist und die Übergabe formatiert erfolgt (IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...)), so kennt der ausführende Modul das Kommando unter dem internen Namen. Standardmäßig nimmt SDF-A als internen Kommandonamen die ersten acht Zeichen (ohne Bindestrich) des externen Namens, den Sie beim Operanden NAME angegeben haben.

RESULT-INTERNAL-NAME = *SAME / <alphanum-name 1..8>
Dieser Operand ist lediglich für einige der durch System-Module implementierten Kommandos von Bedeutung. Die Implementierung durch System-Module ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten. Der Operand RESULT-INTERNAL-NAME wird deshalb hier nicht beschrieben.

STANDARD-NAME = *NAME / *NO / list-poss: *NAME / <structured-name 1..30>
zusätzlicher externer Kommandoname, der bei der Kommandoeingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME kann im Gegensatz zum ALIAS-NAME nicht gelöscht werden, solange das Kommando mit diesem Namen in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE) existiert.

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
zusätzlicher externer Kommandoname, der bei der Kommandoeingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
zusätzlicher externer Kommandoname, der die kürzest mögliche Abkürzung für das Kommando festlegt. Eine kürzere Abkürzung wird dann nicht akzeptiert, selbst wenn sie in Bezug auf andere Kommandos eindeutig wäre.

Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst, *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar das richtige Kommando auswählt, es aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem Kommandonamen (NAME) entstehen.

3. Die ALIAS-NAMEs und STANDARD-NAMEs des Kommandos dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Kommandonamens sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (keine Verlängerung) vorgenommen werden.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für das Kommando gibt.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = <c-string 1..500 with-low>
Hilfetext.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

DOMAIN = *NO / list-poss(2000): <structured-name 1..30>
Bestimmt, ob und ggf. welchen Anwendungsbereichen das Kommando zugeordnet wird.

IMPLEMENTOR =
Art der Kommandoimplementierung

IMPLEMENTOR = *PROCEDURE(...)
Das Kommando ist durch eine Prozedur implementiert. Die Eingabe des Kommandos bewirkt den Aufruf der Prozedur.

NAME =
Name der aufzurufenden Prozedur.

NAME = <c-string 1..280>
Name der Datei, in der die Prozedur steht. Wenn SDF-P geladen ist, kann auch der Name einer Listenvariablen, die die Prozedur enthält, angegeben werden. Die Angabe erfolgt dann in der Form ’*VARIABLE(VARIABLE-NAME=varname)’.
Bibliothekselemente können mit ’*LIBRARY-ELEMENT(LIBRARY=library, ELEMENT=element)’ angegeben werden.
Bei Angabe von ’library(element)’ wird der Wert von CALL-OPTIONS ignoriert. Diese Schreibweise sollte deshalb nicht mehr verwendet werden.

Der Benutzer ist selbst dafür verantwortlich, dass die Zeichenkette zur Angabe des Prozedurbehälters korrekt aufgebaut ist. Ein Fehler an dieser Stelle wird erst beim Aufruf des neu definierten Kommandos sichtbar.

NAME = *BY-IMON(...)
Der Name der Prozedur bzw. der Bibliothek, die diese Prozedur als Bibliothekselement enthält, wird durch den Aufruf von IMON-GPN, dem Installation-Pathmanager bereitgestellt (siehe Benutzerhandbuch „IMON“ [13]).

LOGICAL-ID = <filename 1..30 without-cat-user-gen-vers>
Logischer Name der kommandoimplementierenden Prozedur bzw. der Bibliothek, die diese Prozedur als Bibliothekselement enthält (z.B. SYSSPR).

INSTALLATION-UNIT =<text 1..30 without-sep>
Name der Installation-Unit, z.B. SDF-A

VERSION = *STD / <product-version>
Version der Installation-Unit (vgl. „product-version“ auf Seite 15 bzw. Seite 186).
Bei Angabe von *STD wird die Version verwendet, die mit dem Kommando SELECT-PRODUCT-VERSION ausgewählt wurde. Falls dieses Kommando für die betreffende Installation-Unit noch nicht ausgeführt wurde, wird die höchste Version verwendet.

DEFAULT-PATH-NAME = <filename 1..54>
Vollständiger Dateiname einer Prozedur, die aufgerufen wird, falls IMON-GPN nicht verfügbar ist oder wenn LOGICAL-ID, INSTALLATION-UNIT oder VERSION im System nicht bekannt sind. Ist die Prozedur in einem Bibliothekselement gespeichert, bezeichnet der hier angegebene Dateiname die Bibliothek, aus der die im Operanden ELEMENT angegebene Prozedur aufgerufen wird.
Bei anderen Fehlern wird das Kommando mit einer Fehlermeldung abgewiesen, d.h. die hier angegebene Prozedur wird nicht aufgerufen.

ELEMENT = *NONE / <composed-name 1..64>
Gibt an, ob die Prozedur in einem Bibliothekselement gespeichert ist.

ELEMENT = <composed-name 1..64>
Name des Bibliothekselements, das die Prozedur enthält. Der Elementname wird an IMON-GPN übergeben.

CALL-TYPE = *CALL-PROCEDURE / *INCLUDE-PROCEDURE / *ENTER-PROCEDURE
Legt fest, ob die Prozedur mit dem Kommando CALL-PROCEDURE, INCLUDE-PROCEDURE oder ENTER-PROCEDURE aufgerufen wird. Mit CALL- und ENTER-PROCEDURE können sowohl S- als auch Nicht-S-Prozeduren aufgerufen werden. Mit INCLUDE-PROCEDURE können nur S-Prozeduren aufgerufen werden (siehe Handbücher „SDF-P“ [12] und „Kommandos, Band 1-5“ [4]).

Beispiel:
Damit die Prozedur mit `CALL-PROCEDURE NAME=*LIB-ELEM(LIBRARY=xxx,ELEMENT=yyy)` aufgerufen wird, muss das Kommando so definiert sein:

```
//ADD-CMD ...,IMPLEMENTOR=*PROC(NAME='*LIB-ELEM(LIBRARY=xxx,-
//   ELEMENT=yyy)',CALL-TYPE=*CALL-PROCEDURE...
```

CALL-OPTIONS = *NONE / <c-string 1..1800 with-low>
Gibt eine Zeichenkette an, die weitere Operanden (z.B. LOGGING) für den Prozeduraufruf mit CALL-/INCLUDE- oder ENTER-PROCEDURE in der folgenden Form enthält:
`CALL-OPTIONS='operandx=wertx,operandy=werty, ...'`
Der Operand PROCEDURE-PARAMETERS der Kommandos CALL-/INCLUDE-/ ENTER-PROCEDURE darf nicht in dieser Zeichenkette enthalten sein.

UNLOAD-PROGRAM = *YES / *NO
Gibt an, ob ein Programm entladen werden soll, wenn das im Operanden NAME definierte Kommando in dem Programm über den CMD-Makro ausgeführt wird.
Die aufgerufene Prozedur darf selbst kein Kommando enthalten, das mit CMD-ALLOWED= *YES(UNLOAD=*YES) definiert ist, insbesondere darf in der Prozedur kein TU-Programm aufgerufen werden.

IMPLEMENTOR = *TPR(...)
Das Kommando ist durch System-Module implementiert. Diese Möglichkeit der Kommandoimplementierung ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten. Die Struktur *TPR(...) wird deshalb hier nicht beschrieben.

IMPLEMENTOR = *APPLICATION(...)
Das Kommando wird durch eine $CONSOLE-Anwendung implementiert. Diese Möglichkeit ist der Systembetreuung vorbehalten. Die Voraussetzungen dafür sind im Handbuch „Einführung in die Systembetreuung“ [6] beschrieben. Die erforderlichen Kommandos CONNECT-CMD-SERVER und DISCONNECT-CMD-SERVER finden Sie im Handbuch „Kommandos, Band 1-5“ [4].

LOGGING = *BY-SDF / *BY-IMPLEMENTOR
Der Operand ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten und wird deshalb hier nicht beschrieben.

IMPLEMENTOR = *BY-TPR(...)
Ein bereits existierendes Kommando dient als Vorlage für das neue Kommando.

TPR-CMD = <structured-name 1..30>
Name eines mit IMPLEMENTOR=*TPR(...) definierten Kommandos, das in der Syntaxdatei-Hierarchie bekannt ist.

Beispiel:
Sie öffnen eine Benutzersyntaxdatei, positionieren bei Bedarf mit EDIT und definieren mit den folgenden Anweisungen ein neues Kommando X-SET-PROCEDURE-DIALOG:

```
//ADD-CMD X-SET-PROCEDURE-DIALOG,-
//IMPLEMENTOR=*BY-TPR(TPR-CMD=MODIFY-SDF-OPTIONS),PRIVIL=STD-PROCESSING
//COPY *COMMAND(MODIFY-SDF-OPTIONS),-
//     FROM-FILE=$TSOS.SYSSDF.SDF.045(*SYSTEM),ATTACHED-INFO=*ONLY
//EDIT *OPERAND(PROCEDURE-DIALOG,ORIGIN=*COMMAND(X-SET-PROCEDURE-DIALOG))
//MODIFY-OPERAND DEFAULT='*YES'
//END
```

Nach dem Aktivieren der so bearbeiteten Benutzersyntaxdatei können Sie das neue Kommando verwenden. Die beiden folgenden Kommandos wirken nun identisch:

```
/X-SET-PROC-DIALOG
/MODIFY-SDF-OPTIONS PROCEDURE-DIALOG=*YES
```

REMOVE-POSSIBLE = *YES / *NO
Bestimmt, ob das Kommando gelöscht werden darf (siehe Anweisung REMOVE, Seite 307).

DIALOG-ALLOWED =
Bestimmt, ob das Kommando im Dialogbetrieb zugelassen ist.

DIALOG-ALLOWED = *YES(...)
Das Kommando ist im Dialogbetrieb für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 147).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

DIALOG-ALLOWED = *NO(...)
Das Kommando ist im Dialogbetrieb für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 148).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

DIALOG-PROC-ALLOWED =
Bestimmt, ob das Kommando im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

DIALOG-PROC-ALLOWED = *YES(...)
Das Kommando ist im Dialogbetrieb innerhalb einer Prozedur für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 147).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

DIALOG-PROC-ALLOWED = *NO(...)
Das Kommando ist im Dialogbetrieb innerhalb einer Prozedur für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 148).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

GUIDED-ALLOWED =
Bestimmt, ob das Kommando im geführten Dialog zugelassen ist.

GUIDED-ALLOWED = *YES(...)
Das Kommando ist im geführten Dialog für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 147).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

GUIDED-ALLOWED = *NO(...)
Das Kommando ist im geführten Dialog für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 148).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

BATCH-ALLOWED =
Bestimmt, ob das Kommando im Stapelbetrieb zugelassen ist.

BATCH-ALLOWED = *YES(...)
Das Kommando ist im Stapelbetrieb für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 147).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

BATCH-ALLOWED = *NO(...)
Das Kommando ist im Stapelbetrieb für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 148).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

BATCH-PROC-ALLOWED =
Bestimmt, ob das Kommando im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

BATCH-PROC-ALLOWED = *YES(...)
Das Kommando ist im Stapelbetrieb innerhalb einer Prozedur für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 147).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

BATCH-PROC-ALLOWED = *NO(...)
Das Kommando ist im Stapelbetrieb innerhalb einer Prozedur für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 148).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

CMD-ALLOWED =
Bestimmt, ob das Kommando mit dem CMD-Makro aufgerufen werden kann.

CMD-ALLOWED = *YES(...)
Das Kommando kann mit dem CMD-Makro aufgerufen werden. Der Aufruf mit dem CMD-Makro ist für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen. Einschränkungen hinsichtlich der zugelassenen Betriebsart (DIALOG-ALLOWED, DIALOG-PROC-ALLOWED, BATCH-ALLOWED, BATCH-PROC-ALLOWED) gelten nicht, wenn das Kommando mit dem CMD-Makro aufgerufen wird.

UNLOAD = *NO / *YES
Bestimmt, ob das aufrufende Programm entladen wird.
Bei Kommandos, die durch eine Kommandoprozedur implementiert sind, wird der Operand nicht ausgewertet. Ob das aufrufende Programm entladen wird, ist in diesem Fall im Operanden UNLOAD-PROGRAM (siehe Operand IMPLEMENTOR= PROCEDURE(...), Seite 139) vereinbart.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist der Aufruf des Kommandos zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Der Aufruf des Kommandos ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 147).

PRIVILEGE = *ALL
Der Aufruf des Kommandos ist für alle zurzeit definierten Privilegien und für alle Privilegien zulässig, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Aufruf des Kommandos ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

CMD-ALLOWED = *NO(...)
Der Aufruf des Kommandos mit dem CMD-Makro ist für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE =
Für Benutzeraufträge mit den hier angegebenen Privilegien ist der Aufruf des Kommandos gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Der Aufruf des Kommandos ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 148).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Aufruf des Kommandos ist nur für genau die Privilegien gesperrt, die Sie in dieser Liste angeben.

NEXT-INPUT =
Bestimmt, was für eine Eingabe nach dem Kommando erwartet wird. Diese Angabe benötigt SDF zur Steuerung des geführten Dialogs.

NEXT-INPUT = *CMD
Ein Kommando wird für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Kommando.

NEXT-INPUT = *STMT
Eine Anweisung wird für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Anweisung. Beispiel: Ein mittels Prozedur implementiertes Kommando startet ein Programm, das als Erstes eine Anweisung einliest.

NEXT-INPUT = *DATA
Daten werden für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Daten. Beispiel: Ein mittels Prozedur implementiertes Kommando startet ein Programm, das als Erstes Daten einliest.

NEXT-INPUT = *ANY
Die Art der nachfolgenden Eingabe ist nicht vorhersehbar.

PRIVILEGE =
Gibt an, welche Privilegien dem Kommando zugeordnet werden.

PRIVILEGE = *ALL
Das Kommando erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Das Kommando erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Kommando zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

ADD-DOMAIN
Anwendungsbereich definieren

Mit der Anweisung ADD-DOMAIN definieren Sie in der bearbeiteten Syntaxdatei einen Anwendungsbereich. Die für den Anwendungsbereich vergebenen Namen müssen eindeutig von den Namen aller übrigen Anwendungsbereiche zu unterscheiden sein.

```
ADD-DOMAIN

 NAME = <structured-name 1..30>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,HELP = *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
        TEXT = <c-string 1..500 with-low>
,PRIVILEGE = *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
        EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = <structured-name 1..30>
Name des Anwendungsbereichs, der im geführten Dialog benutzt wird.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
interner Name des Anwendungsbereichs. Dieser darf nicht verändert werden. SDF erkennt mithilfe des internen Namens einen Anwendungsbereich, der in mehreren Syntaxdateien mit unterschiedlichem externen Namen definiert ist, als denselben Anwendungsbereich. Standardmäßig nimmt SDF-A als internen Namen die ersten acht Zeichen des externen Namens (ohne Bindestrich), den Sie beim Operanden NAME angegeben haben.

HELP = *NO / list-poss(2000): <name 1..1>(...)
Bestimmt, ob und ggf. welche Hilfetexte es für den Anwendungsbereich gibt.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = <c-string 1..500 with-low>
Hilfetext.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

PRIVILEGE =
Gibt an, welche Privilegien dem Anwendungsbereich zugeordnet werden.

PRIVILEGE = *ALL
Der Anwendungsbereich erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

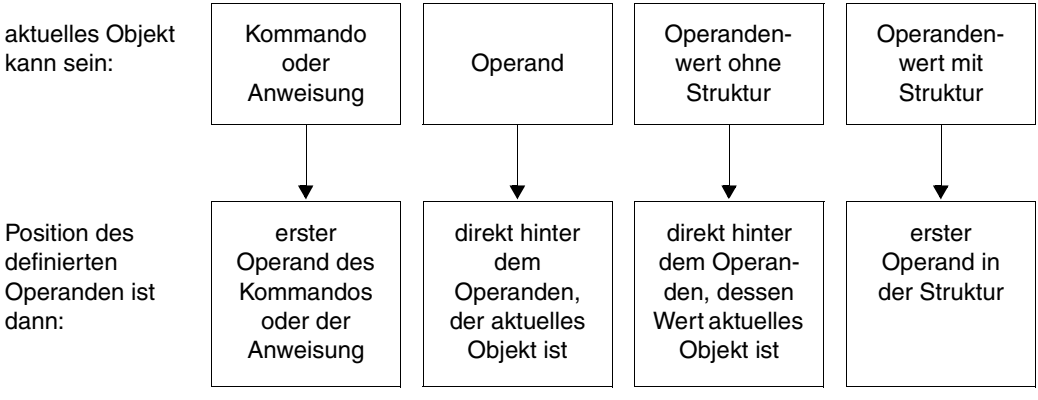
PRIVILEGE = *EXCEPT(...)
Der Anwendungsbereich erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Anwendungsbereich zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Anwendungsbereich erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

ADD-OPERAND
Operand definieren

Mit der Anweisung ADD-OPERAND definieren Sie in der geöffneten Syntaxdatei einen Operanden. Das Kommando bzw. die Anweisung, wofür Sie den Operanden definieren, muss in der Syntaxdatei bereits definiert sein. Welche Position der definierte Operand innerhalb dieses Kommandos bzw. dieser Anweisung erhält, hängt davon ab, welches Objekt zur Zeit der Eingabe von ADD-OPERAND das „aktuelle“ ist. Der definierte Operand ist anschließend aktuelles Objekt.



Alle für den Operanden vergebenen Namen müssen in Bezug auf die übrigen Operanden derselben Ebene (bzw. derselben Struktur) eindeutig sein.

Einen Operanden für ein durch System-Module implementiertes Kommando können Sie nur definieren, wenn die Kommandodefinition in einer Gruppen- oder Systemsyntaxdatei steht.

Die Definitionen der zu dem Operanden gehörenden Operandenwerte werden nicht mit der Anweisung ADD-OPERAND, sondern mit der Anweisung ADD-VALUE bzw. COPY in die Syntaxdatei eingebracht.

(Teil 1 von 2)

ADD-OPERAND

NAME = <structured-name 1..20>

,**INTERN**AL-**NAME** = ***STD** / <alphanum-name 1..8>

,**STAND**ARD-**NAME** = ***NAME** / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..20>

,**ALIAS-NAME** = ***NO** / list-poss(2000): <structured-name 1..20>

,**MIN**IMAL-**ABBREV**IATION = ***NO** / <structured-name 1..30>

,**HELP** = ***NO** / list-poss(2000): <name 1..1>(...)

<name 1..1>(...)

TEXT = <c-string 1..500 with-low>

,**DEF**AULT = ***NONE** / ***JV**(...) / ***VAR**IABLE(...) / <c-string 1..1800 with-low>(...)

***JV**(...)

JV-NAME = <filename 1..54 without-gen-vers>

,**ALTER**NATE-**DEF**AULT = ***NONE** / <c-string 1..1800 with-low>(...)

<c-string 1..1800 with-low>(...)

ANALYSE-**DEF**AULT = ***YES** / ***NO**

***VAR**IABLE(...)

VARIABLE-**NAME** = <composed-name 1..255>

,**ALTER**NATE-**DEF**AULT = ***NONE** / <c-string 1..1800 with-low>(...)

<c-string 1..1800 with-low>(...)

ANALYSE-**DEF**AULT = ***YES** / ***NO**

<c-string 1..1800 with-low>(...)

ANALYSE-**DEF**AULT = ***YES** / ***NO**

,**SECRET-PROMPT** = ***NO** / ***YES**

,**STRUCT**URE-**IMPLICIT** = ***NO** / ***YES**

,**REM**OVE-**POSSIBLE** = ***YES** / ***NO**

,**DIAL**OG-**ALLOW**ED = ***YES** / ***NO**

,**DIAL**OG-**PROC-ALLOW**ED = ***YES** / ***NO**

,**GUID**ED-**ALLOW**ED = ***YES** / ***NO**

,**BATCH-ALLOW**ED = ***YES** / ***NO**

,**BATCH-PROC-ALLOW**ED = ***YES** / ***NO**

Fortsetzung

(Teil 2 von 2)

```
,LIST-POSSIBLE = *NO / *YES(...)
    *YES(...)
        LIMIT = *STD / <integer 1..3000>
        ,FORM = *NORMAL / *OR
,LINES-IN-FORM = 1 / <integer 1..15>
,PRESENCE = *NORMAL / *EXTERNAL-ONLY / *INTERNAL-ONLY
,RESULT-OPERAND-LEVEL = 1 / <integer 1..5>
,RESULT-OPERAND-NAME = *SAME / <structured-name 1..20> / *POSITION(...) / *LABEL /
                       *COMMAND-NAME
    *POSITION(...)
        POSITION = <integer 1..3000>
,CONCATENATION-POS = *NO / <integer 1..20>
,VALUE-OVERLAPPING = *NO / *YES
,OVERWRITE-POSSIBLE = *NO / *YES
,PRIVILEGE = *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
        EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = <structured-name 1..20>
(externer) Operandenname, der bei der Kommando- bzw. Anweisungseingabe anzugeben ist (siehe aber Operand PRESENCE=*INTERNAL-ONLY). Der Benutzer kann ihn im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Eingabe abkürzen.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
interner Operandenname. SDF identifiziert einen Operanden, der in mehreren Syntaxdateien mit unterschiedlichem externen Namen definiert ist, mithilfe des internen Operandennamens als denselben Operanden. Standardmäßig nimmt SDF-A als internen Operandennamen die ersten acht Zeichen (ohne Bindestrich) des externen Namens, den Sie beim Operanden NAME angegeben haben.

STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..20>
zusätzlicher externer Operandenname, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME darf im Gegensatz zum ALIAS-NAME nicht gelöscht werden, solange der Operand mit diesem Namen in einer der zugewiesenen Syntaxdateien (siehe OPEN-SYNTAX-FILE) existiert.

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..20>
zusätzlicher externer Operandenname, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
zusätzlicher externer Operandenname, der die kürzest mögliche Abkürzung für den Operanden festlegt. Eine kürzere Abkürzung wird dann nicht akzeptiert, selbst wenn sie in Bezug auf andere Operanden eindeutig wäre.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar den richtigen Operanden auswählt, ihn aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem Operandennamen (NAME) entstehen.
3. Die ALIAS-NAMEs und STANDARD-NAMEs des Operanden dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Operandennamens sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (keine Verlängerung) vorgenommen werden.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für den Operanden gibt.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = <c-string 1..500 with-low>
Hilfetext.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

DEFAULT=
Bestimmt, ob es für den Operanden einen Default-Wert gibt.

DEFAULT= *NONE
Es gibt keinen Default-Wert. Der Operand ist ein Pflichtoperand.

DEFAULT = *JV(...)
Der Operand ist wahlfrei. Sein Default-Wert ist in der Job-Variablen gespeichert, deren Name angegeben wird. Wird eine Job-Variable als Default-Wert verwendet, wird der Default-Wert von SDF immer zum Zeitpunkt der Ausführung analysiert. Ist der Zugriff auf die Job-Variable nicht möglich, so wird der mit ALTERNATE-DEFAULT definierte alternative Default-Wert verwendet. Gibt es keinen alternativen Default-Wert, ist der Operand als Pflichtoperand zu betrachten (entspricht DEFAULT=*NONE). DEFAULT=*VARIABLE(...) darf deshalb nicht zusammen mit PRESENCE=*INTERNAL-ONLY angegeben werden.

JV-NAME = <filename 1..54 without-gen-vers>
Name der Job-Variablen.

ALTERNATE-DEFAULT =
Alternativer Default-Wert, der verwendet wird, wenn beim Zugriff auf die Jobvariable Fehler auftreten.

ALTERNATE-DEFAULT = *NONE
Es gibt keinen alternativen Default-Wert.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternativer Default-Wert. Für die Angabe dieses alternativen Default-Werts gelten die gleichen Regeln wie für die Definition eines Operanden. Er kann beispielsweise auch eine in Klammern eingeschlossene Liste sein. Der alternative Default-Wert muss durch eine zu dem Operanden gehörende Operandenwertdefinition (siehe ADD-VALUE) abgedeckt sein. Deckt ein mit STAR=*MANDATORY definiertes Schlüsselwort diesen Default-Wert ab, so muss der Default-Wert mit einem Stern eingegeben werden.

ANALYSE-DEFAULT = *YES / *NO
Bestimmt, ob SDF-A den angegebenen Wert bereits bei Abschluss der Kommando- bzw. Anweisungsdefinition syntaktisch analysiert. Das beschleunigt die Kommando- bzw. Anweisungsanalyse zur Laufzeit, setzt aber voraus, dass der Wert weder eine Struktur einleitet noch aus einer Liste besteht.

DEFAULT = *VARIABLE(...)
Der Operand ist wahlfrei. Sein Default-Wert ist in der S-Variablen (siehe Benutzerhandbuch „SDF-P“ [12]) gespeichert, deren Name angegeben wird.
Wird eine S-Variable als Default-Wert verwendet, dann wird der Default-Wert von SDF immer zum Zeitpunkt der Ausführung analysiert. Ist der Zugriff auf die S-Variable nicht möglich, so wird der mit ALTERNATE-DEFAULT definierte alternative Default-Wert verwendet. Gibt es keinen alternativen Default-Wert, ist der Operand als Pflichtoperand zu betrachten (entspricht DEFAULT=*NONE). DEFAULT=*VARIABLE(...) darf deshalb nicht zusammen mit PRESENCE=*INTERNAL-ONLY angegeben werden.

VARIABLE-NAME = <composed-name 1..255>
Name der S-Variablen.

ALTERNATE-DEFAULT =
Alternativer Default-Wert, der verwendet wird, wenn beim Zugriff auf die S-Variable Fehler auftreten.

ALTERNATE-DEFAULT = *NONE
Es gibt keinen alternativen Default-Wert.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternativer Default-Wert. Für die Angabe dieses alternativen Default-Werts gelten die gleichen Regeln wie für die Eingabe eines Operanden. Er kann beispielsweise auch eine in Klammern eingeschlossene Liste sein. Der alternative Default-Wert muss durch eine zu dem Operanden gehörende Operandenwertdefinition (siehe ADD-VALUE) abgedeckt sein. Deckt ein mit STAR=*MANDATORY definiertes Schlüsselwort diesen Default-Wert ab, so muss der Default-Wert mit einem Stern eingegeben werden.

ANALYSE-DEFAULT = *YES / *NO
Bestimmt, ob SDF-A den angegebenen Wert bereits bei Abschluss der Kommando- bzw. Anweisungsdefinition syntaktisch analysiert. Das beschleunigt die Kommando- bzw. Anweisungsanalyse zur Laufzeit, setzt aber voraus, dass der Wert weder eine Struktur einleitet noch aus einer Liste besteht.

DEFAULT= <c-string 1..1800 with-low>(...)
Der Operand ist wahlfrei und hat den angegebenen Default-Wert. Für die Angabe dieses Default-Werts gelten die gleichen Regeln wie für die Eingabe eines Operanden. Er kann beispielsweise auch eine in Klammern eingeschlossene Liste sein. Der Default-Wert muss durch eine zu dem Operanden gehörende Operandenwertdefinition (siehe ADD-VALUE) abgedeckt sein. Deckt ein mit STAR=*MANDATORY definiertes Schlüsselwort den Default-Wert ab, so muss auch der Default-Wert mit einem Stern angegeben werden.

ANALYSE-DEFAULT = *YES / *NO
Bestimmt, ob SDF-A den angegebenen Default-Wert bereits bei Abschluss der Kommando- bzw. Anweisungsdefinition syntaktisch analysiert. Das beschleunigt die Kommando- bzw. Anweisungsanalyse zur Laufzeit, setzt aber voraus, dass der Default-Wert weder eine Struktur einleitet noch aus einer Liste besteht.

SECRET-PROMPT = *NO / *YES
Bestimmt, ob der Operand als geheimer Operand behandelt wird. Das Eingabefeld für den Wert eines geheimen Operanden wird dunkelgesteuert und seine Protokollierung wird unterdrückt (siehe auch ADD-VALUE...,OUTPUT=*SECRET-PROMPT und ADD-VALUE..., SECRET-PROMPT=*SAME/*NO).

STRUCTURE-IMPLICIT =
ist nur relevant für Operanden, die in einer Struktur stehen, und bestimmt, ob bei der Kommando- bzw. Anweisungseingabe durch globale Angabe des Operandennamens implizit die Struktur mit ausgewählt wird, die den Operanden enthält.

STRUCTURE-IMPLICIT = *NO
Die Struktur wird bei Angabe des Operanden nur dann implizit mit ausgewählt, wenn der Operandenname kommando- oder anweisungsglobal bzw. innerhalb einer bereits ausgewählten Struktur eindeutig ist.

STRUCTURE-IMPLICIT = *YES
Die Struktur wird bei Angabe des Operanden auch dann implizit mit ausgewählt, wenn andere Operanden mit gleichem Namen existieren. Diese müssen dann allerdings mit STRUCTURE-IMPLICIT=*NO definiert sein.
Diese Angabe ist nur für solche Operanden erlaubt, deren Struktur mit dem Wert KEYWORD oder KEYWORD-NUMBER eingeleitet wird.

Beispiel

```
/SHOW-FILE-ATTR ACCESS-METHOD=*ISAM
```

ist die verkürzte Schreibweise von

```
/SHOW-FILE-ATTR SEL=*BY-ATTR(ACCESS-METHOD=*ISAM)
```

Die gleichzeitige Angabe des Operanden innerhalb und außerhalb der Struktur kann zu Fehlern führen (z.B. SDF-Meldung `CMD0039: MORE THAN ONE VALUE HAS BEEN SPECIFIED FOR AN OPERAND. ONLY THE LAST ONE IS USED`).

REMOVE-POSSIBLE = *YES / *NO
Bestimmt, ob der Operand gelöscht werden darf (siehe Anweisung REMOVE, Seite 307).

DIALOG-ALLOWED = *YES / *NO
Bestimmt, ob der Operand im Dialogbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Dialogbetrieb zugelassen ist.

DIALOG-PROC-ALLOWED = *YES / *NO
Bestimmt, ob der Operand im Dialogbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

GUIDED-ALLOWED = *YES / *NO
Bestimmt, ob der Operand im geführten Dialog zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im geführten Dialog zugelassen ist.
Zur Durchsetzung sicherheitsrelevanter Aspekte ist GUIDED-ALLOWED=*NO nicht geeignet, da so definierte Operanden im Prozedurfehlerdialog sowie bei /SHOW-CMD bzw. //SHOW-STMT mit FORM=*UNGUIDED angezeigt werden.

BATCH-ALLOWED = *YES / *NO
Bestimmt, ob der Operand im Stapelbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Stapelbetrieb zugelassen ist.

BATCH-PROC-ALLOWED = *YES / *NO
Bestimmt, ob der Operand im Stapelbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

LIST-POSSIBLE =
Bestimmt, ob an der Operandenposition eine Liste zugelassen ist. Für welche Operandenwerte diese Liste zulässig ist, wird mit der Anweisung ADD-VALUE festgelegt.

LIST-POSSIBLE = *NO
Es ist keine Liste zugelassen.

LIST-POSSIBLE = *YES(...)
Es ist eine Liste zugelassen.

LIMIT = *STD / <integer 1..3000>
Bestimmt die maximale Anzahl der Listenelemente. Standardmäßig setzt SDF-A den Wert 2000 ein (siehe auch Seite 383).

FORM = *NORMAL / *OR
Bestimmt, ob die Listenelemente einzeln adressiert (*NORMAL) oder mit logischem OR zu einem einzigen Wert verknüpft an die Implementierung übergeben werden (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371). Letzteres ist nur sinnvoll bei Listenelementen vom Datentyp KEYWORD, für die sedezimale Übergabewerte definiert sind (siehe ADD-VALUE..,VALUE=<c-string>(..,OUTPUT=<x-string>...). Diese Angabe ist nur relevant, wenn der definierte Operand zu einer Anweisung oder zu einem mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierten Kommando (siehe ADD-CMD) gehört. Die hier gemachte Angabe muss konsistent mit dem in der Implementierung definierten Übergabebereich sein.

LINES-IN-FORM = 1 / <integer 1..15>
Bestimmt die Anzahl der Eingabezeilen im Fragebogen des geführten Dialogs.

PRESENCE =
Bestimmt, ob der Operand unterdrückt wird.

PRESENCE = *NORMAL
Der Operand wird nicht unterdrückt.

PRESENCE = *EXTERNAL-ONLY
Die Übergabe des Operanden an die Implementierung wird unterdrückt (z.B. ein nicht mehr benötigter Operand, der aus Kompatibilitätsgründen an der Benutzeroberfläche erhalten bleiben muss, oder ein Operand, der lediglich dazu dient, weitere Operanden in einer Struktur zu gruppieren).

PRESENCE = *INTERNAL-ONLY
Der Operand wird an der Benutzeroberfläche unterdrückt. In Verbindung mit der dann obligatorischen Definition eines Default-Werts (siehe Operand DEFAULT=) lässt sich so einem implementierten Parameter ein fester Wert zuordnen, ohne dass deshalb im Kommando- bzw. Anweisungsformat ein Operand für den Benutzer sichtbar ist. Hängt an dem Operanden eine Struktur, so werden alle in ihr enthaltenen Unteroperanden in die höhere Ebene integriert. PRESENCE=*INTERNAL-ONLY darf nicht zusammen mit DEFAULT=*JV(...) oder DEFAULT=*VARIABLE(...) angegeben werden.

RESULT-OPERAND-LEVEL = 1 / <integer 1..5>
Bestimmt die Strukturebene, auf der der Operand an die Implementierung übergeben wird. Bei einem Operanden, der in keiner Struktur steht, muss dieser Wert 1 sein. Für einen Operanden, der in einer Struktur steht, gilt: Der RESULT-OPERAND-LEVEL ist gleich oder niedriger als die Strukturebene, auf der der Operand im Eingabeformat des Kommandos bzw. der Anweisung steht. Er ist kleiner, gleich oder um eins größer als der RESULT-OPERAND-LEVEL des Operanden, zu dem der struktureinleitende Operandenwert gehört. Bei Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierten Kommandos siehe auch ADD-VALUE... STRUCTURE=*YES(...,FORM=...).

RESULT-OPERAND-NAME =
Bestimmt, wie die Implementierung den Operanden in dem Übergabebereich oder String identifiziert, den SDF an sie übergibt.
Anmerkung: Einen Übergabebereich (siehe Seite 371ff) benutzt SDF bei Anweisungen und bei Kommandos, die mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definiert sind (siehe ADD-CMD). Einen String übergibt SDF bei Kommandos, die mit IMPLEMENTOR=*PROCEDURE oder IMPLEMENTOR=*TPR(..., CMD-INTERFACE=*STRING,...) definiert sind (siehe ADD-CMD).

RESULT-OPERAND-NAME = *SAME
ist nur zulässig, wenn die Operandenübergabe mittels String erfolgt. Der Operand hat in dem zu übergebenden String den gleichen Namen, den Sie ihm mit NAME= gegeben haben.

RESULT-OPERAND-NAME = <structured-name 1..20>
ist nur zulässig, wenn die Operandenübergabe mittels String erfolgt. Der Operand hat in dem zu übergebenden String den angegebenen Namen.

RESULT-OPERAND-NAME = *POSITION(...)
Der Operand hat in dem Übergabebereich (siehe Seite 371ff) oder in dem zu übergebenden String eine bestimmte Position. Ein Name ist ihm nicht zugeordnet.

POSITION = <integer 1..3000>
Position. Bei Operanden, die zu einer Struktur gehören, ist die Positionsangabe strukturrelativ. Das bedeutet, dass der erste Operand in der Struktur die Position 1 hat, wenn für den aktuellen Operanden ein höheres RESULT-OPERAND-LEVEL definiert wurde, als der ihm übergeordnete Operand hat.

RESULT-OPERAND-NAME = *LABEL
*ist nur zulässig für Operanden in Kommandos, die mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE= *STRING,...) definiert sind (siehe ADD-CMD).*
Der Operandenwert ist für die Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert und wird deshalb hier nicht beschrieben.

RESULT-OPERAND-NAME = *COMMAND-NAME
*ist nur zulässig für Operanden in Kommandos, die mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE= *STRING,...) definiert sind (siehe ADD-CMD).*
Der Operandenwert ist für die Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert und wird deshalb hier nicht beschrieben.

CONCATENATION-POS = *NO / <integer 1..20>
Bestimmt, ob und ggf. wie der Operand mit anderen Eingabeoperanden auf einen einzigen Operanden in dem an die Implementierung zu übergebenden String abgebildet wird. Die Eingabeoperanden werden miteinander verkettet. Welche Position sie bei der Verkettung erhalten, ist hier anzugeben. Voraussetzung ist, dass die Übergabe an die Implementierung in Stringform erfolgt (Kommandos, die definiert sind mit IMPLEMENTOR=*PROCEDURE oder IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...), siehe ADD-CMD). Alle miteinander zu verkettenden Eingabeoperanden müssen denselben RESULT-OPERAND-NAME haben. Bei gleicher Positionsangabe für mehrere Eingabe-Operanden verwendet SDF bei der Analyse den zuerst auftretenden Operanden.

VALUE-OVERLAPPING =
Bestimmt, ob eine Überlappung von Datentypen in der Definition der Operandenwerte zugelassen werden soll.

VALUE-OVERLAPPING = *NO
es ist keine Überlappung zugelassen (siehe „Sich ausschließende Datentypen“ auf Seite 635).

VALUE-OVERLAPPING = *YES
Eine Überlappung der Datentypen ist zulässig.
Bei der Kommando- bzw. Anweisungseingabe überprüft SDF den eingegebenen Operandenwert anhand der Datentypdefinitionen in der Reihenfolge, wie diese für den Operanden angegeben sind. Passt keiner der Datentypen zum Eingabewert, so liefert SDF eine Fehlermeldung. Ist der Datentyp KEYWORD(-NUMBER) so überprüft SDF, ob der eingegebene Wert eindeutig hinsichtlich weiterer Definitionen vom Typ KEYWORD(-NUMBER) ist. Zusätzlich überprüft SDF die definierten Eigenschaften (z.B. Länge, Wertebereich) des eingegebenen Wertes. Treffen diese Eigenschaften nicht zu, wird mit dem nächsten definierten Datentyp die Überprüfung fortgesetzt.

OVERWRITE-POSSIBLE = *NO / *YES

*nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE= *NEW/ *TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*

Bestimmt, ob der Default-Wert des Operanden durch einen von der Implementierung dynamisch erzeugten Wert überschrieben wird (siehe Operand DEFAULT in den Makros CMDCST, CMDRST und CMDTST). Der vom Programm erzeugte Wert muss gültiger Operandenwert sein. Im geführten Dialog zeigt SDF den von der Implementierung erzeugten Wert im Fragebogen.

PRIVILEGE =

Gibt an, welche Privilegien dem Operanden zugeordnet werden.

PRIVILEGE = *SAME

Der Operand erhält die gleichen Privilegien, die auch für das zugehörige Kommando oder die Anweisung festgelegt sind. Gehört der Operand zu einer Struktur, erhält der Operand die gleichen Privilegien wie der Operandenwert, der diese Struktur einleitet.

PRIVILEGE = *EXCEPT(...)

Der Operand erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>

Gibt an, welche Privilegien nicht dem Operanden zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>

Der Operand erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

ADD-PROGRAM
Programm definieren

Mit der Anweisung ADD-PROGRAM definieren Sie in der geöffneten Syntaxdatei ein Programm. Die für das Programm vergebenen Namen müssen eindeutig in Bezug auf alle übrigen in der Syntaxdatei definierten Namen sein.

```
ADD-PROGRAM

 NAME = <structured-name 1..30>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,PRIVILEGE = *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      | EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
,COMMENT-LINE = *NONE / *STD / <c-string 1..50 with-low>
```

NAME = <structured-name 1..30>
(externer) Programmname, der im geführten Dialog angezeigt wird. Dieser Name ist frei wählbar (muss nicht mit dem Modul- oder Phasennamen übereinstimmen).

INTERNAL-NAME = *STD / <alphanum-name 1..8>
interner Programmname. Er kann nicht verändert werden. Das Programm gibt ihn gegenüber SDF an, wenn es die Eingabe von Anweisungen anfordert (siehe Makroaufrufe CMDRST und CMDTST). Standardmäßig nimmt SDF-A die ersten acht Zeichen (ohne Bindestriche) des externen Programmnamens, den Sie beim Operanden NAME angegeben haben.

PRIVILEGE =
Gibt an, welche Privilegien dem Programm zugeordnet werden.

PRIVILEGE = *ALL
Das Programm erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Das Programm erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Programm zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Programm erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

COMMENT-LINE =
Gibt an, welche Programm-Kommentarzeile im geführten Dialog angezeigt werden soll. Die Programm-Kommentarzeile erscheint ganz oben in den Fragebögen des geführten Dialogs.

COMMENT-LINE = *NONE
Keine Programm-Kommentarzeile wird angezeigt.

COMMENT-LINE = *STD
In der Programm-Kommentarzeile werden die Programmversion und das Erzeugungsdatum des Programmes angezeigt. Bindemodule (Objektmodule, Typ-R-Elemente) haben keine interne Version. Deshalb wird an Stelle des Erzeugungsdatums das Ausführungsdatum angezeigt.

COMMENT-LINE = <c-string 1..50 with-low>
Zeichenkette, die als Programm-Kommentarzeile ausgegeben wird.

ADD-STMT
Anweisung definieren

Mit der Anweisung ADD-STMT definieren Sie in der geöffneten Syntaxdatei eine Anweisung für ein Programm. Diese Anweisung ist anschließend „aktuelles“ Objekt (siehe Seite 151). Das Programm muss in der Syntaxdatei bereits definiert sein. Alle für die Anweisung vergebenen Namen müssen eindeutig in Bezug auf die Menge der zu dem Programm gehörenden Anweisungen sein.
Die Definitionen der zu der Anweisung gehörenden Operanden und Operandenwerte werden nicht mit der Anweisung ADD-STMT, sondern mit den Anweisungen ADD-OPERAND bzw. ADD-VALUE oder COPY in die Syntaxdatei eingebracht.

ADD-STMT

NAME = <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

,**INTERN**AL-**NAME** = ***STD** / <alphanum-name 1..8>

,**STAND**ARD-**NAME** = ***NAME** / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..30>

,**ALIAS-NAME** = ***NO** / list-poss(2000): <structured-name 1..30>

,**MIN**IMAL-**ABBREV**IATION = ***NO** / <structured-name 1..30>

,**HELP** = ***NO** / list-poss(2000): <name 1..1>(...)

<name 1..1>(...)
TEXT = <c-string 1..500 with-low>

,**MAX-STRUC-OPER**AND = ***STD** / <integer 1..3000>

,**IMPL**EMENTATION = ***NORM**AL / ***RESTORE-SDF-INPUT** / ***SHOW-INPUT-HISTORY** / *REMARK / ***WRITE-TEXT** / *STEP / *END / ***MODIFY-SDF** / ***SHOW-SDF** / ***EXECUTE** / ***HOLD** / ***SHOW-INPUT-DEFAULTS** / ***RESET-INPUT-DEFAULTS** / ***HELP-MSG** / ***SHOW-STMT**

,**REM**OVE-**POSSIBLE** = ***YES** / ***NO**

,**DIAL**OG-**ALLOW**ED = ***YES** / ***NO**

,**DIAL**OG-**PROC-ALLOW**ED = ***YES** / ***NO**

,**GUID**ED-**ALLOW**ED = ***YES** / ***NO**

,**BATCH-ALLOW**ED = ***YES** / ***NO**

,**BATCH-PROC-ALLOW**ED = ***YES** / ***NO**

,**NEXT-IN**PUT = ***STMT** / ***DATA** / ***ANY**

,**PRIV**ILEGE = ***ALL** / ***EXC**EPT(...) / list-poss(64): <structured-name 1..30>

***EXC**EPT(...)
EXCEPT-**PRIV**ILEGE = list-poss(64): <structured-name 1..30>

,**STMT-VERSION** = ***NONE** / <integer 1..999>

NAME = <structured-name 1..30>
(externer) Anweisungsname, der bei der Anweisungseingabe anzugeben ist. Der Benutzer kann ihn im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Anweisungseingabe abkürzen.

PROGRAM = <structured-name 1..30>
externer Name des Programms (siehe ADD-PROGRAM), zu dem die Anweisung gehört.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
interner Anweisungsname. Das Programm, zu dem die Anweisung gehört, kennt diese unter dem internen Namen (vgl. Makroaufrufe CMDRST und CMDTST). Dieser kann nicht verändert werden. SDF identifiziert eine Anweisung, die in mehreren Syntaxdateien mit unterschiedlichem externen Namen definiert ist, mithilfe des internen Anweisungsnamens als dieselbe Anweisung. Standardmäßig nimmt SDF-A als internen Anweisungsnamen die ersten acht Zeichen (ohne Bindestrich) des externen Namens, den Sie beim Operanden NAME angegeben haben.

STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..30>
zusätzlicher externer Anweisungsname, der bei der Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME darf im Gegensatz zum ALIAS-NAME nicht gelöscht werden, solange die Anweisung mit diesem Namen in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE) existiert. STANDARD-NAME ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten und wird deshalb hier nicht beschrieben.

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
zusätzlicher externer Anweisungsname, der bei der Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
zusätzlicher externer Anweisungsname, der die kürzest mögliche Abkürzung für die Anweisung festlegt. Eine kürzere Abkürzung wird dann nicht akzeptiert, selbst wenn sie in Bezug auf andere Anweisungen eindeutig wäre.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar die richtige Anweisung auswählt, sie aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem Anweisungsnamen (NAME) entstehen.
3. Die ALIAS-NAMEs und STANDARD-NAMEs der Anweisung dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Anweisungsnamens sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (keine Verlängerung) vorgenommen werden.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für den Operanden gibt.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = <c-string 1..500 with-low>
Hilfetext.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

MAX-STRUC-OPERAND = *STD / <integer 1..3000>
Anzahl der in der strukturierten Übergabe auf oberster Ebene zu reservierenden Operandenpositionen. Standardmäßig wird das Operanden-Array so groß wie erforderlich angelegt. Für geplante künftige Erweiterungen lässt es sich aber auch größer anlegen.

IMPLEMENTATION = *NORMAL / *RESTORE-SDF-INPUT / *SHOW-INPUT-HISTORY / *REMARK / *WRITE-TEXT / *STEP / *END / *MODIFY-SDF / *SHOW-SDF / *EXECUTE / *HOLD /*SHOW-INPUT-DEFAULTS / *RESET-INPUT-DEFAULTS / *HELP-MSG / *SHOW-STMT
Diese Funktion ist nur für die Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen und wird deshalb hier nicht beschrieben.

REMOVE-POSSIBLE = *YES / *NO
Bestimmt, ob die Anweisung gelöscht werden darf (vgl. REMOVE, Seite 307).

DIALOG-ALLOWED = *YES / *NO
Bestimmt, ob die Anweisung im Dialogbetrieb zugelassen ist.

DIALOG-PROC-ALLOWED = *YES / *NO
Bestimmt, ob die Anweisung im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

GUIDED-ALLOWED = *YES / *NO
Bestimmt, ob die Anweisung im geführten Dialog zugelassen ist.

BATCH-ALLOWED = *YES / *NO
Bestimmt, ob die Anweisung im Stapelbetrieb zugelassen ist.

BATCH-PROC-ALLOWED = *YES / *NO
Bestimmt, ob die Anweisung im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

NEXT-INPUT =
Bestimmt, was für eine Eingabe nach der Anweisung erwartet wird. Diese Angabe benötigt SDF zur Steuerung des geführten Dialogs.

NEXT-INPUT = *STMT
Eine Anweisung wird für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Anweisung.

NEXT-INPUT = *DATA
Daten werden für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Daten.

NEXT-INPUT = *ANY
Die Art der nachfolgenden Eingabe ist nicht vorhersehbar.

PRIVILEGE =
Gibt an, welche Privilegien der Anweisung zugeordnet werden.

PRIVILEGE = *ALL
Die Anweisung erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Die Anweisung erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht der Anweisung zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Die Anweisung erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

STMT-VERSION = *NONE / <integer 1..999>
Version der Anweisung. Der Wert wird im normierten Übergabebereich übergeben. *NONE bedeutet, dass ein NULL-Wert im normierten Übergabebereich eingetragen wird. STMT-VERSION wird ignoriert, falls das Programm, zu dem die Anweisung gehört, nicht mit den neuen Makros CMDRST und CMDTST arbeitet bzw. noch das alte Format des normierten Übergabebereiches verwendet. Mehr zum Format des normierten Übergabebereiches und zu den SDF-Makros finden Sie im Kapitel „SDF-Programmschnittstelle“ ab Seite 371ff und Seite 385ff.

ADD-VALUE
Operandenwert definieren

Mit der Anweisung ADD-VALUE definieren Sie in der gerade geöffneten Syntaxdatei einen Operandenwert. Der Operand, für den Sie den Operandenwert definieren, muss in der Syntaxdatei bereits definiert sein.

An welcher Stelle der Operandenwert eingefügt wird, hängt davon ab, ob der Operand selbst oder einer seiner bereits definierten Werte „aktuelles“ Objekt ist (siehe Seite 151):

– Ist der Operand selbst „aktuelles“ Objekt, so wird der definierte Operandenwert als erster Wert dieses Operanden in die Kommando- bzw. Anweisungsdefinition eingefügt.

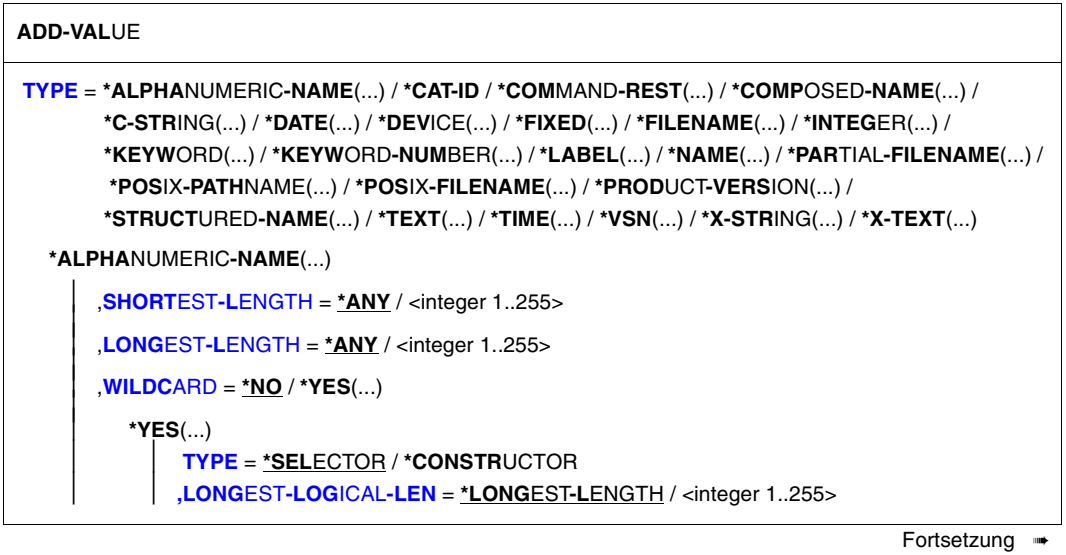
– Ist ein bereits definierter Wert des Operanden „aktuelles“ Objekt, so wird der neu definierte Operandenwert unmittelbar hinter diesem bereits vorhandenen Wert eingefügt.

In beiden Fällen wird der neu definierte Operandenwert anschließend „aktuelles“ Objekt.

Alle für den Operandenwert vergebenen Namen müssen in Bezug auf seine Umgebung eindeutig sein. Wenn Sie die Namen nicht explizit definieren, sondern die Namensvergabe SDF-A überlassen, kann es beim Datentyp KEYWORD vorkommen, dass der von SDF-A standardmäßig gebildete interne Name unzulässig ist.

Einen Operandenwert für ein durch System-Module implementiertes Kommando können Sie nur definieren, wenn die Kommandodefinition in einer Gruppen- oder Systemsyntaxdatei steht.

(Teil 1 von 6)



Fortsetzung ➠

(Teil 2 von 6)

```
*COMMAND-REST(...)
    LOWER-CASE = *NO / *YES
*COMPOSED-NAME(...)
    SHORTEST-LENGTH = *ANY / <integer 1..1800>
    ,LONGEST-LENGTH = *ANY / <integer 1..1800>
    ,UNDERSCORE = *NO / *YES
    ,WILDCARD = *NO / *YES(...)
        *YES(...)
            TYPE = *SELECTOR / *CONSTRUCTOR
            ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..1800>
*C-STRING(...)
    SHORTEST-LENGTH = *ANY / <integer 1..1800>
    ,LONGEST-LENGTH = *ANY / <integer 1..1800>
    ,LOWER-CASE = *NO / *YES
*DATE(...)
    COMPLETION = *NO / *YES
*DEVICE(...)
    CLASS-TYPE = *DISK(...) / list-poss(2000): *DISK(...) / *TAPE(...)
        *DISK(...)
            EXCEPT = *NO / list-poss(50): <text 1..8 without-sep>
            ,SCOPE = *ALL / *STD-DISK
        *TAPE(...)
            EXCEPT = *NO / list-poss(50): <text 1..8 without-sep>
    ,ALIAS-ALLOWED = *YES / *NO
    ,VOLUME-TYPE-ONLY = *NO / *YES
    ,RESULT-VALUE = *BY-NAME / *BY-CODE
*FIXED(...)
    LOWEST = *ANY / <fixed -2147483648..2147483647>
    ,HIGHEST = *ANY / <fixed -2147483648..2147483647>
```

Fortsetzung ➠

(Teil 3 von 6)

***FILENAME**(...)
- **SHORT**EST-**L**ENGTH = ***ANY** / <integer 1..80>
- ,**LONG**EST-**L**ENGTH = ***ANY** / <integer 1..80>
- ,**CAT**ALOG-**ID** = ***YES** / ***NO**
- ,**USER-ID** = ***YES** / ***NO**
- ,**GEN**ERATION = ***YES** / ***NO**
- ,**VERS**ION = ***YES** / ***NO**
- ,**WILDC**ARD = ***NO** / ***YES**(...)
  - ***YES**(...)
    - **TYPE** = ***SEL**ECTOR / ***CONSTR**UCTOR
    - **,LONG**EST-**LOG**ICAL-**LEN** = ***LONG**EST-**L**ENGTH / <integer 1..80>
- ,**PATH-COMPL**ETION = ***NO** / ***YES**
- ,**TEMP**ORARY-**FILE** = ***YES** / ***NO**

***INTEG**ER(...)
- **LOW**EST = ***ANY** / <integer -2147483648..2147483647>
- ,**HIGH**EST = ***ANY** / <integer -2147483648..2147483647>
- ,**OUT-FORM** = ***STD** / ***BIN**ARY / ***P**ACKED / ***UNP**ACKED / ***CHAR**

***KEYW**ORD(...)
- **STAR** = ***OPT**IONAL / ***MAND**ATORY

***KEYW**ORD-**NUM**BER(...)
- **STAR** = ***OPT**IONAL / ***MAND**ATORY

***LABEL**(...)
- **SHORT**EST-**L**ENGTH = ***ANY** / <integer 1..8>
- ,**LONG**EST-**L**ENGTH = ***ANY** / <integer 1..8>

***NAME**(...)
- **SHORT**EST-**L**ENGTH = ***ANY** / <integer 1..255>
- ,**LONG**EST-**L**ENGTH = ***ANY** / <integer 1..255>
- ,**UNDER**SCORE = ***NO** / ***YES**
- ,**LOW**ER-**C**ASE = ***NO** / ***YES**

Fortsetzung ➠

(Teil 4 von 6)

,**WILDC**ARD = ***NO** / ***YES**(...)

***YES**(...)

TYPE = ***SEL**ECTOR / ***CONSTR**UCTOR

,LONGEST-**LOG**ICAL-**LEN** = ***LONG**EST-**L**ENGTH / <integer 1..255>

***PAR**TIAL-**FILENAME**(...)

SHORTEST-**L**ENGTH = ***ANY** / <integer 2..79>

,**LONG**EST-**L**ENGTH = ***ANY** / <integer 2..79>

,**CAT**ALOG-**ID** = ***YES** / ***NO**

,**USER-ID** = ***YES** / ***NO**

,**WILDC**ARD = ***NO** / ***YES**(...)

***YES**(...)

TYPE = ***SEL**ECTOR / ***CONSTR**UCTOR

,LONGEST-**LOG**ICAL-**LEN** = ***LONG**EST-**L**ENGTH / <integer 2..79>

***POS**IX-**PATH**NAME(...)

SHORTEST-**L**ENGTH = ***ANY** / <integer 1..1023>

,**LONG**EST-**L**ENGTH = ***ANY** / <integer 1..1023>

,**WILDCARD** = ***YES** / ***NO**

,**QUOTES** = ***OPT**IONAL / ***MAND**ATORY

***POS**IX-**FILENAME**(...)

SHORTEST-**L**ENGTH = ***ANY** / <integer 1..255>

,**LONG**EST-**L**ENGTH = ***ANY** / <integer 1..255>

,**WILDCARD** = ***YES** / ***NO**

,**QUOTES** = ***OPT**IONAL / ***MAND**ATORY

***PROD**UCT-**VERS**ION(...)

USER-INTERFACE = ***YES**(...) / ***NO** / ***ANY**(...)

***YES**(...)

CORRECTION-**STA**TE = ***YES** / ***NO** / ***ANY**

***ANY**(...)

CORRECTION-**STA**TE = ***ANY** / ***NO** / ***YES**

Fortsetzung

(Teil 5 von 6)

*STRUCTURED-NAME(...)
| SHORTEST-LENGTH = *ANY / <integer 1..255>
| ,LONGEST-LENGTH = *ANY / <integer 1..255>
| ,WILDCARD = *NO / *YES(...)
| | *YES(...)
| | | TYPE = *SELECTOR / *CONSTRUCTOR
| | | ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..255>
*TEXT(...)
| SHORTEST-LENGTH = *ANY / <integer 1..1800>
| ,LONGEST-LENGTH = *ANY / <integer 1..1800>
| ,LOWER-CASE = *NO / *YES
| ,SEPARATORS = *YES / *NO
*TIME(...)
| OUT-FORM = *STD / *BINARY / *CHAR
*VSN(...)
| SHORTEST-LENGTH = *ANY / <integer 1..6>
| ,LONGEST-LENGTH = *ANY / <integer 1..6>
*X-STRING(...)
| SHORTEST-LENGTH = *ANY / <integer 1..1800>
| ,LONGEST-LENGTH = *ANY / <integer 1..1800>
*X-TEXT(...)
| SHORTEST-LENGTH = *ANY / <integer 1..3600>
| ,LONGEST-LENGTH = *ANY / <integer 1..3600>
| ,ODD-POSSIBLE = *YES / *NO
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,REMOVE-POSSIBLE = *YES / *NO
,SECRET-PROMPT = *SAME / *NO
,DIALOG-ALLOWED = *YES / *NO
,DIALOG-PROC-ALLOWED = *YES / *NO
,GUIDED-ALLOWED = *YES / *NO
,BATCH-ALLOWED = *YES / *NO
,BATCH-PROC-ALLOWED = *YES / *NO

Fortsetzung ➠

(Teil 6 von 6)

```
,STRUCTURE = *NO / *YES(...)
    *YES(...)
        SIZE = *SMALL / *LARGE
        ,FORM = *FLAT / *NORMAL
        ,MAX-STRUC-OPERAND = *STD / <integer 1..3000>
,LIST-ALLOWED = *NO / *YES
,VALUE = *NO / list-poss(2000): <c-string 1..1800 with-low>(...)
    <c-string 1..1800 with-low>(...)
        STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..30> /
                        <c-string 1..30>
        ,ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
        ,GUIDED-ABBREVIATION = *NAME / <structured-name 1..30> / <c-string 1..30>
        ,MINIMAL-ABBREVIATION = *NO / <structured-name 1..30> / <c-string 1..30>
        ,NULL-ABBREVIATION = *NO / *YES
        ,OUTPUT = *SAME / *EMPTY-STRING / *DROP-OPERAND / <c-string 1..1800> / <x-string 1..3600>
        ,OUT-TYPE = *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST /
                    *COMPOSED-NAME / *C-STRING / *DATE / *DEVICE / *FIXED / *FILENAME /
                    *INTEGER / *KEYWORD / *KEYWORD-NUMBER / *LABEL / *NAME /
                    *PARTIAL-FILENAME / *PRODUCT-VERSION / *POSIX-PATHNAME /
                    *POSIX-FILENAME / *STRUCTURED-NAME / *TEXT / *TIME / *VSN /
                    *X-STRING / *X-TEXT
        ,OVERWRITE-POSSIBLE = *NO / *YES
,OUTPUT = *NORMAL(...) / *SECRET-PROMPT
    *NORMAL(...)
        AREA-LENGTH = *VARIABLE / <integer 1..3000>
        ,LEFT-JUSTIFIED = *STD / *YES / *NO
        ,FILLER = *STD / <c-string 1..1> / <x-string 1..2>
        ,STRING-LITERALS = *NO / *HEX / *CHAR
        ,HASH = *NO / *YES(...)
            *YES(...)
                OUTPUT-LENGTH = <integer 2..32>
,PRIVILEGE = *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
        EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

TYPE =
Bestimmt, von welchem Datentyp der Operandenwert ist. Die verschiedenen, zu einem Operanden definierten Werte dürfen sich hinsichtlich des Datentyps nicht ausschließen (siehe Seite 635). (Ist dies in Ausnahmefällen nicht möglich, so muss in der Anweisung ADD-OPERAND oder MODIFY-OPERAND für VALUE-OVERLAPPING der Wert *YES angegeben werden.) Ansonsten ist es beispielsweise nicht möglich, für einen Operanden einen Wert vom Typ NAME und einen alternativen Wert vom Typ STRUCTURED-NAME zu definieren. Lediglich der Datentyp KEYWORD darf in mehr als einer alternativen Operandenwertdefinition angegeben sein. Bei der Kommando- oder Anweisungseingabe prüft SDF, ob ein eingegebener Operandenwert von dem für ihn definierten Typ ist. Bei der folgenden Beschreibung der Datentypen wird wiederholt der Begriff „alphanumerisches Zeichen“ benutzt. Dieses kann ein Buchstabe (A,B,C,..,Z), eine Ziffer (0,1,2,...,9) oder ein Sonderzeichen (@,#,$) sein.

TYPE = *ALPHANUMERIC-NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp ALPHANUMERIC-NAME ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD =
Bestimmt, ob Platzhalterzeichen („Wildcards“,siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE =
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN =
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..255>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *CAT-ID
Bestimmt, dass der Operandenwert vom Datentyp CAT-ID ist. Dieser ist als Folge von maximal 4 Zeichen (A-Z, 0-9; Sonderzeichen $, @, # sind nicht erlaubt). Die Begrenzungszeichen ':' werden nicht mitgezählt. Eine 4 Zeichen lange CAT-ID darf nicht mit der Zeichenfolge 'PUB' beginnen.

TYPE = *COMMAND-REST(...)
Bestimmt, dass der Operandenwert vom Datentyp COMMAND-REST ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *COMPOSED-NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp COMPOSED-NAME ist. Dieser Datentyp entspricht weitgehend dem Datentyp FILENAME, hat jedoch folgende Abweichungen:

1. nicht erlaubt sind folgende Zeichen: ':' , '(' , ')'
2. erlaubt sind Buchstaben, Zahlen und die folgenden Sonderzeichen: '@' , '$' , '#' , '.' , '-'. Ob '_' erlaubt ist, hängt vom Operanden UNDERSCORE ab.
3. keine Einschränkung für die Zeichen '@' , '$' und '#'
4. die Zeichen '-' und '.' dürfen nur als Separator zwischen anderen Zeichen angegeben werden. Nicht erlaubt ist z.B. '..' , '--' , '.-' oder '-.'. Außerdem dürfen '.' und '-' nicht am Anfang oder am Ende eines composed-name stehen.
5. keine Einschränkung der Länge auf maximal 54 Zeichen
6. es muss nicht zwingend ein Buchstabe vorhanden sein

SHORTEST-LENGTH = *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

UNDERSCORE = *NO / *YES
Bestimmt, ob der Unterstrich '_' akzeptiert wird.

WILDCARD =
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE =
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN =
gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..1800>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *C-STRING(...)
Bestimmt, dass der Operandenwert vom Datentyp C-STRING ist. Dieser ist definiert als eine Folge von EBCDIC-Zeichen, die in Hochkommas eingeschlossen ist. Ihr kann der Buchstabe C vorangestellt sein. Ein Hochkomma als Wert innerhalb der begrenzenden Hochkommas ist bei der Eingabe doppelt anzugeben.

SHORTEST-LENGTH = *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

LOWER-CASE = *NO / *YES
Bestimmt, ob in Hochkommas eingeschlossene Kleinbuchstaben erhalten bleiben.

TYPE = *DATE(...)
Bestimmt, dass der Operandenwert vom Datentyp DATE ist. Dieser ist definiert als eine Folge von einer vierstelligen und zwei zweistelligen Zahlen, die durch Bindestrich miteinander verbunden sind (<Jahr>-<Monat>-<Tag>). Das Jahr kann auch mit zwei Ziffern an Stelle von vier angegeben werden.

COMPLETION = *NO / *YES
Bestimmt, ob eine zweistellige Jahresangabe komplettiert wird. Bei Angabe von *YES ergänzt SDF zweistellige Jahresangaben (Datum der Form jj-mm-tt) zu:
– 20jj-mm-tt falls jj < 60
– 19jj-mm-tt falls jj ≥ 60.

TYPE = *DEVICE(...)
Bestimmt, dass der Operandenwert vom Datentyp DEVICE ist. Für Operanden, deren Operandenwert mit dem Datentyp DEVICE definiert ist, wird dem Benutzer im geführten Dialog eine Liste der im System verfügbaren Platten- oder Bandgeräte angeboten (siehe auch Handbuch „Systeminstallation“ [9]).

CLASS-TYPE = *DISK(...) / list-poss(2000): *DISK(...) / *TAPE(...)
Gibt die Geräteklasse an.

CLASS-TYPE = *DISK(...)
Das Gerät gehört zur Klasse der Plattengeräte.

EXCEPT = *NO / list-poss(50): <text 1..8 without-sep>
Bestimmt, welche Plattengeräte nicht in der Liste der verfügbaren Geräte erscheinen sollen.

SCOPE =
Legt fest, ob alle Plattengeräte in der Liste der verfügbaren Geräte erscheinen oder nur die über DVS festgelegten Standard-Plattengeräte (siehe Handbuch „Einführung in das DVS“ [7]).
In BS2000/OSD-BC < V3.0 gilt immer SCOPE=*ALL.

SCOPE = *ALL
Alle Plattengeräte erscheinen in der Liste.

SCOPE = *STD-DISK
Nur die Plattengeräte, die im DVS als Standard-Plattengeräte festgelegt sind, erscheinen in der Liste.

CLASS-TYPE = *TAPE(...)
Das Gerät gehört zur Klasse der Bandgeräte.

EXCEPT = *NO / list-poss(50): <text 1..8 without-sep>
Bestimmt, welche Geräte nicht in der Liste der verfügbaren Geräte erscheinen sollen.

ALIAS-ALLOWED = *YES / *NO
Bestimmt, ob die Angabe des Alias-Namens des Gerätes erlaubt ist.

VOLUME-TYPE-ONLY = *NO / *YES
Bestimmt, ob der Volumetyp akzeptiert wird.

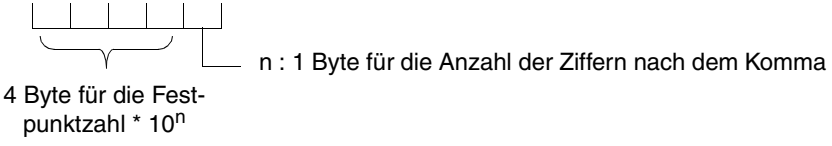
RESULT-VALUE =
Bestimmt, in welcher Form SDF die Information an die Implementierung übergibt.

RESULT-VALUE = *BY-NAME
SDF übergibt den externen Gerätenamen. Der externe Gerätename ist 8 Zeichen lang.

RESULT-VALUE = *BY-CODE
SDF übergibt den internen Geratecode. Der interne Gerätecode ist 2 Byte lang.

TYPE = *FIXED(...)
Bestimmt, dass der Operandenwert vom Datentyp FIXED ist. Dieser ist wie folgt definiert:
[Zeichen][Ziffern].[Ziffern]
[Zeichen] entspricht + oder –
[Ziffern] entspricht 0..9

FIXED muss aus mindestens einer Ziffer, darf aber höchstens aus 10 Zeichen (Ziffern und ein '.') bestehen. Im normierten Übergabebereich wird der Wert im folgenden Format abgelegt:



LOWEST = *ANY / <fixed -2147483648..2147483647>
Bestimmt, ob es für die Festpunktzahl eine untere Grenze gibt und ggf. welche.

HIGHEST = *ANY / <fixed -2147483648..2147483647>
Bestimmt, ob es für die Festpunktzahl eine obere Grenze gibt und ggf. welche.

TYPE = *FILENAME(...)
Bestimmt, dass der Operandenwert vom Datentyp FILENAME ist. Für die einzugebende Zeichenfolge gilt die Definition, die im Handbuch „Einführung in das DVS" [7] für den vollqualifizierten Dateinamen angegeben ist.

SHORTEST-LENGTH = *ANY / <integer 1..80>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..80>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

CATALOG-ID = *YES / *NO
Bestimmt, ob die Katalogkennung als Teil des Dateinamens angegeben werden darf.

USER-ID = *YES / *NO
Bestimmt, ob die Benutzerkennung als Teil des Dateinamens angegeben werden darf.

GENERATION = *YES / *NO
Bestimmt, ob eine Generationsnummer als Teil des Dateinamens angegeben werden darf. Wird dem Dateinamen eine Struktur angehängt, so muss *NO angegeben werden.

VERSION = *YES / *NO
Bestimmt, ob eine Versionsbezeichnung als Teil des Dateinamens angegeben werden darf. Wird dem Dateinamen eine Struktur angehängt, so muss *NO angegeben werden.

WILDCARD =
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.
Der Datentyp <filename x..y with-wild> beinhaltet auch teilqualifizierte Dateinamen, d.h. es ist nicht notwendig, für den Operanden zusätzlich einen Wert vom Datentyp <partial-filename> zu definieren.

TYPE =
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN =
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..80>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

PATH-COMPLETION = *NO / *YES
Legt fest, ob der Dateiname bei der Übergabe an die Implementierung zum vollständigen Pfadnamen ergänzt wird. Das schließt auch die Ersetzung von Aliasnamen durch die ACS-Funktion ein.
PATH-COMPLETION=*YES darf nur angegeben werden, wenn CATALOG-ID und USER-ID erlaubt sind und wenn Wildcards im Dateinamen nicht erlaubt sind.

TEMPORARY-FILE = *YES / *NO
Gibt an, ob temporäre Dateinamen erlaubt sind.

TYPE = *INTEGER(...)
Bestimmt, dass der Operandenwert vom Datentyp INTEGER ist. Dieser ist definiert als eine Folge von Ziffern, der ein Vorzeichen vorangestellt sein kann.

LOWEST = *ANY / <integer -2147483648 .. 2147483647>
Bestimmt, ob es für die Ganzzahl eine untere Grenze gibt und ggf. welche.

HIGHEST = *ANY / <integer -2147483648 .. 2147483647>
Bestimmt, ob es für die Ganzzahl eine obere Grenze gibt und ggf. welche.

OUT-FORM = *STD / *BINARY / *PACKED / *UNPACKED / *CHAR
Bestimmt, in welchem Format SDF die Ganzzahl an die Implementierung übergibt. Erfolgt die Übergabe in einem Übergabebereich (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371), so setzt SDF-A *STD in *BINARY um. Bei Übergabe einer Zeichenkette (Kommandos, die mit IMPLEMENTOR=*PROCEDURE(...) oder IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...) definiert sind, siehe ADD-CMD) setzt SDF-A OUT-FORM=*STD in *CHAR um. Wenn die Zahl im Binärformat übergeben wird, erzeugt SDF mindestens 4 Byte (siehe Operand OUTPUT). Wird die Zahl im *PACKED-Format übergeben, erzeugt SDF mindestens 8 Byte und in den anderen Formaten (*CHAR, *UNPACKED) mindestens 1 Byte.

TYPE = *KEYWORD(...)
Bestimmt, dass der Operandenwert vom Datentyp KEYWORD ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen. Diese Zeichenfolge kann in mehrere Teilzeichenfolgen gegliedert sein, die durch Bindestrich miteinander verbunden sind. Teilzeichenfolgen dürfen den Punkt enthalten. Der Punkt darf jedoch nicht am Beginn oder Ende einer Teilzeichenfolge stehen. Die gesamte Zeichenfolge, d.h. ggf. die erste Teilzeichenfolge muss mit einem Buchstaben oder Sonderzeichen beginnen. Der Wertebereich für einen Operandenwert vom Typ KEYWORD ist auf einen oder auf eine endliche Anzahl von genau festgelegten Einzelwerten beschränkt (siehe Operand VALUE in dieser Anweisung).
Ab SDF-A/SDF Version 2.0 wird auch der Wert '*...' akzeptiert. Dieser Wert kann verwendet werden, wenn für einen Operanden eine Liste von Schlüsselwörtern definiert werden muss (z.B. die Definition für alle externen Geräte). Damit können neue Schlüsselwörter (z.B. neue Gerätenamen) eingefügt werden, ohne dass die Syntaxdatei verändert werden muss, denn mit '*...' akzeptiert SDF zusätzliche Werte und analysiert sie als Schlüsselwörter. Der Datentyp KEYWORD darf maximal 30 Zeichen lang sein.

STAR =
Bestimmt, ob bei der Eingabe der Zeichenfolge ein Stern vorangestellt werden muss.

STAR = *OPTIONAL
Ein Stern darf, muss aber nicht vorangestellt werden.
Falls ein Stern vorangestellt ist, wird er bei der Übergabe an die Implementierung unterdrückt.

STAR = *MANDATORY
Ein Stern muss vorangestellt werden (erforderlich zur Unterscheidung von alternativen Datentypen).

TYPE = *KEYWORD-NUMBER(...)
Bestimmt, dass der Operandenwert vom Datentyp KEYWORD-NUMBER ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *LABEL(...)
Bestimmt, dass der Operandenwert vom Datentyp LABEL ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp NAME ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen, die mit einem Buchstaben oder Sonderzeichen beginnt.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

UNDERSCORE = *NO / *YES
Bestimmt, ob '_' (Unterstrich) akzeptiert wird.

LOWER-CASE = *NO / *YES
Legt fest, ob Kleinbuchstaben erhalten bleiben.

WILDCARD =
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE =
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN =
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..255>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *PARTIAL-FILENAME(...)
Bestimmt, dass der Operandenwert vom Datentyp PARTIAL-FILENAME ist. Für die einzugebende Zeichenfolge gilt die Definition, die im Handbuch „Einführung in das DVS“ [7] für den teilqualifizierten Dateinamen angegeben ist.

SHORTEST-LENGTH = *ANY / <integer 2..79>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 2..79>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

CATALOG-ID = *YES / *NO
Bestimmt, ob die Katalogkennung als Teil des Dateinamens angegeben werden darf.

USER-ID = *YES / *NO
Bestimmt, ob die Benutzerkennung als Teil des Dateinamens angegeben werden darf.

WILDCARD =
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE =
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN =
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 2..79>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *POSIX-PATHNAME(...)
Bestimmt, dass der Operandenwert vom Datentyp POSIX-PATHNAME ist. Für die einzugebende Zeichenfolge gelten folgende Konventionen:

– erlaubt sind Buchstaben, Ziffern und die Zeichen '_' , '-' , '.' und '/' ,wobei '/' immer als Trennzeichen zwischen den Verzeichnissen und Dateinamen dient. Steuerzeichen sind nicht erlaubt.
– Ein POSIX-PATHNAME besteht aus POSIX-FILENAMEs, die durch '/' getrennt sind. Ein POSIX-PATHNAME darf insgesamt nicht länger als 1023 Zeichen sein.
– Das Zeichen '\' (Backslash) entwertet Metazeichen in allen POSIX-spezifischen Namen. Es wird nicht mitgezählt. Das Metazeichen * wird ebenfalls nicht mitgezählt.

Metazeichen sind Zeichen, die für Wildcard-Ausdrücke verwendet werden. Folgende Metazeichen können auftreten:

*	kein, ein oder mehrere beliebige Zeichen
?	genau ein beliebiges Zeichen
[S]	genau ein Zeichen aus der mit S bestimmten Zeichenmenge

[!S] genau ein Zeichen, das aber nicht Element der mit S bestimmten Zeichenmenge sein darf

wobei S eine Menge von Zeichen (z.B. abcd) oder
ein Bereich von Zeichen (z.B. a-d) oder
eine Kombination aus Mengen- und Bereichsangaben sein kann.

POSIX-PATHNAMEs, die andere als die oben genannten Zeichen oder '?' oder '!' am Anfang enthalten, müssen bei der Eingabe in Hochkommas eingeschlossen sein (wie C-STRINGs). Im normierten Übergabebereich bzw. im Übergabe-String werden die Hochkommas von SDF wieder entfernt, da sie nicht zum Dateinamen gehören. Hochkommas, die zum Dateinamen gehören, müssen verdoppelt werden.
In Prozeduren sollte zur Vermeidung von Kompatibilitätsproblemen immer die C-String-Syntax verwendet werden.

SHORTEST-LENGTH = *ANY / <integer 1..1023>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..1023>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD = *YES / *NO
Bestimmt, ob Metazeichen (siehe oben) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

QUOTES = *OPTIONAL / *MANDATORY
Legt fest, ob der Pfadname bei der Eingabe in Hochkommas eingeschlossen sein kann (*OPTIONAL) oder muss (*MANDATORY).

TYPE = *POSIX-FILENAME(...)
Bestimmt, dass der Operandenwert vom Datentyp POSIX-FILENAME ist. Für die einzugebende Zeichenfolge gelten die Konventionen wie für POSIX-PATHNAMEs (siehe Seite 184f), mit folgenden Einschränkungen:
– '/' (Slash) darf nicht enthalten sein.
– Die Länge ist auf maximal 255 Zeichen begrenzt.

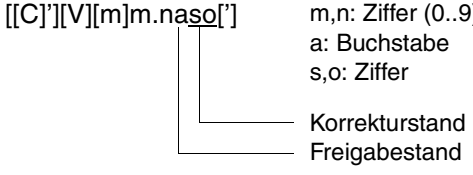
SHORTEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD = *YES / *NO
Bestimmt, ob Metazeichen (siehe POSIX-PATHNAME) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

QUOTES = *OPTIONAL / *MANDATORY
Legt fest, ob der Dateiname bei der Eingabe in Hochkommas eingeschlossen sein kann (*OPTIONAL) oder muss (*MANDATORY).

TYPE = *PRODUCT-VERSION(...)
Bestimmt, dass der Operandenwert vom Datentyp PRODUCT-VERSION ist. Die Produktversion hat folgendes Format:



Die optionalen Zeichen C, V und Hochkomma werden im SDF-Übergabebereich unterdrückt.

USER-INTERFACE =
Legt fest, ob der Freigabestand der Benutzerschnittstelle angegeben werden muss oder darf.

USER-INTERFACE = *YES(...)
Der Freigabestand der Benutzerschnittstelle muss angegeben werden.

CORRECTION-STATE =
Legt fest, ob auch der Korrekturstand angegeben werden muss oder darf.

CORRECTION-STATE = *YES
Der Korrekturstand muss angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.naso['].

CORRECTION-STATE = *NO
Der Korrekturstand darf nicht angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.na['].

CORRECTION-STATE = *ANY
Der Korrekturstand kann angegeben werden.

USER-INTERFACE = *NO
Der Freigabestand der Benutzerschnittstelle und der Korrekturstand dürfen nicht angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.n['].

USER-INTERFACE = *ANY(...)
Der Freigabestand der Benutzerschnittstelle kann angegeben werden.

CORRECTION-STATE =
Legt fest, ob der auch Korrekturstand angegeben werden muss oder darf.

CORRECTION-STATE = *ANY
Der Korrekturstand kann angegeben werden.

CORRECTION-STATE = *NO
Der Korrekturstand darf nicht angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.na['].

CORRECTION-STATE = *YES
Der Korrekturstand muss angegeben werden, wenn der Freigabestand angegeben wird. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.naso['].

TYPE = *STRUCTURED-NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp STRUCTURED-NAME ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen. Diese Zeichenfolge kann in mehrere Teilzeichenfolgen gegliedert sein, die durch Bindestrich miteinander verbunden sind. Die gesamte Zeichenfolge, d.h. ggf. die erste Teilzeichenfolge muss mit einem Buchstaben oder Sonderzeichen beginnen.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD =
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE =
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN =
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..255>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *TEXT(...)
Bestimmt, dass der Operandenwert vom Datentyp TEXT ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *TIME(...)
Bestimmt, dass der Operandenwert vom Datentyp TIME ist. Dieser ist definiert als eine Folge von ein, zwei oder drei vorzeichenlosen Zahlen, die durch Doppelpunkt miteinander verbunden sind (<Stunden>[:<Minuten> [:<Sekunden>]]). Die Angabe für Stunden, Minuten und Sekunden darf maximal zweistellig sein. Einstelligen Zahlen dürfen Nullen vorangestellt sein. Der Wertebereich für Minuten und Sekunden liegt zwischen 0 und 59.

OUT-FORM = *STD / *BINARY / *CHAR
Bestimmt, in welchem Format SDF die Zahlendarstellung der Zeitangabe an die Implementierung übergibt.

OUT-FORM = *STD
Erfolgt die Übergabe in einem Übergabebereich (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371), wird die Zeitangabe im Binär-Format übergeben. Bei Übergabe in einer Zeichenkette (bei Kommandos, die mit IMPLEMENTOR=*PROCEDURE(...) oder IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...) definiert sind, siehe ADD-CMD) wird die Zeitangabe im Character-Format übergeben.

OUT-FORM = *BINARY
Die Zeitangabe wird im Binär-Format übergeben.

OUT-FORM = *CHAR
Die Zeitangabe wird im Character-Format übergeben.

TYPE = *VSN(...)
Bestimmt, dass der Operandenwert vom Datentyp VSN ist. SDF unterscheidet zwischen zwei Typen:

a) VSN mit einem Punkt:
Diese VSN muss aus 6 Zeichen bestehen. Neben einem einzelnen Punkt werden nur die Buchstaben A-Z und die Ziffern 0-9 akzeptiert.
Eine solche VSN hat das Format `pvsid.sequence-number`, wobei `pvsid` aus 2 bis 4 Zeichen und `sequence-number` aus 1 bis 3 Zeichen besteht.
Diesem untergeordneten Typ von VSN darf PUB nicht vorangestellt werden:
z.B. ist PUBA.0 oder PUB.02 ungültig. Der Punkt kann 3., 4. oder 5. Zeichen der VSN sein.

b) VSN ohne Punkt:
Diese VSN besteht aus einer Zeichenfolge von maximal 6 Zeichen. Nur die Buchstaben A-Z, Ziffern 0-9 und die Sonderzeichen $, @, # sind erlaubt. Eine solche VSN kann mit PUB beginnen. In diesem Fall dürfen die folgenden Zeichen keine Sonderzeichen sein (z.B. wird PUB@1 oder PUB$## zurückgewiesen). VSN, die mit der Zeichenfolge „PUB“ beginnen, müssen außerdem 6 Zeichen lang sein.
SDF macht keine weiteren Unterschiede zwischen Public oder Private VSN oder PUBRES.

SHORTEST-LENGTH = *ANY / <integer 1..6>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..6>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

TYPE = *X-STRING(...)
Bestimmt, dass der Operandenwert vom Datentyp X-STRING ist. Dieser ist definiert als eine Folge von Sedezimalzeichen (Ziffern 0-9, Großbuchstaben A-F), die in Hochkommas eingeschlossen ist. Ihr vorangestellt ist der Buchstabe X.

SHORTEST-LENGTH = *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

TYPE = *X-TEXT(...)
Bestimmt, dass der Operandenwert vom Datentyp X-TEXT ist. Dieser Datentyp entspricht weitgehend dem Datentyp X-STRING, hat jedoch keine Hochkommas und kein vorangestelltes 'X'.

SHORTEST-LENGTH = *ANY / <integer 1..3600>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Halbbyte).

LONGEST-LENGTH = *ANY / <integer 1..3600>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Halbbyte).

ODD-POSSIBLE = *YES / *NO
Bestimmt, ob eine ungerade Zahl von Zeichen akzeptiert wird.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Interner Operandenwertname. SDF identifiziert einen Operandenwert mithilfe dieses Namens. Standardmäßig nimmt SDF-A die ersten acht Zeichen (ohne Bindestrich) des Datentyps, den Sie beim Operanden TYPE angegeben haben. Bei Operandenwerten vom Datentyp KEYWORD nimmt SDF-A standardmäßig die ersten acht Zeichen (ohne Bindestrich) des ersten Werts, den Sie beim Operanden VALUE angegeben haben.

SECRET-PROMPT = *SAME / *NO
Gibt an, ob der Operandenwert als geheim behandelt werden soll.
SECRET-PROMPT=*SAME übernimmt die Einstellung des Operanden, zu dem der hier definierte Operandenwert gehört (siehe ADD-OPERAND ...,SECRET-PROMPT= , Seite 156). Das Eingabefeld wird für geheime Operandenwerte dunkelgesteuert und die Protokollierung wird unterdrückt.
Bei SECRET-PROMPT=*NO wird der Operandenwert nicht als geheim behandelt.
Ist ein geheimer Operand mit einem nicht geheimen Wert vorbesetzt, so wird das Eingabefeld nicht dunkelgesteuert. Durch Eingabe des Zeichens ^ oder eines mit OUTPUT= *SECRET-PROMPT definierten Wertes kann die Dunkelsteuerung des Eingabefeldes angefordert werden.

REMOVE-POSSIBLE = *YES / *NO
Bestimmt, ob der Operandenwert gelöscht werden darf (siehe REMOVE, Seite 307).

DIALOG-ALLOWED = *YES / *NO
Bestimmt, ob der Operandenwert im Dialogbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Dialogbetrieb zugelassen ist.

DIALOG-PROC-ALLOWED = *YES / *NO
Bestimmt, ob der Operandenwert im Dialogbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

GUIDED-ALLOWED = *YES / *NO
Bestimmt, ob der Operandenwert im geführten Dialog angeboten wird. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im geführten Dialog zugelassen ist.

BATCH-ALLOWED = *YES / *NO
Bestimmt, ob der Operandenwert im Stapelbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Stapelbetrieb zugelassen ist.

BATCH-PROC-ALLOWED = *YES / *NO
Bestimmt, ob der Operandenwert im Stapelbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

STRUCTURE =
Bestimmt, ob der Operandenwert eine Struktur einleitet.

STRUCTURE = *NO
Der Operandenwert leitet keine Struktur ein.

STRUCTURE = *YES(...)
Der Operandenwert leitet eine Struktur ein.

SIZE = *SMALL / *LARGE
Bestimmt, ob in der MINIMUM- oder MEDIUM-Stufe des geführten Dialogs die Struktur in den übergeordneten Fragebogen integriert wird (*SMALL) oder ob für sie ein eigener Fragebogen angelegt wird (*LARGE).

FORM = *FLAT / *NORMAL
nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE =*NEW/*TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD). Falls LIST-ALLOWED=*YES ist, muss FORM=*NORMAL angegeben werden. Im Standardfall (*FLAT) wird die Struktur im Übergabebereich für die Implementierung linearisiert, die Operanden der Struktur werden in ein übergeordnetes Operanden-Array integriert. Im Fall *NORMAL erhält die Struktur ein eigenes Operanden-Array. In ihm werden die Operanden übergeben, für die als RESULT-OPERAND-LEVEL eine höhere

Strukturebene definiert ist als für den Operanden, zu dem der hier definierte struktureinleitende Operandenwert gehört (siehe ADD-OPERAND...,RESULT-OPERAND-LEVEL=,... und Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371).

MAX-STRUC-OPERAND = *STD / <integer 1..3000>
Anzahl der in der strukturierten Übergabe zu reservierenden Operandenpositionen. Standardmäßig wird das Operanden-Array so groß wie für die Struktur erforderlich angelegt. Für geplante künftige Erweiterungen lässt es sich aber auch größer anlegen. Dieser Operand bezieht sich auf die durch den Operandenwert eingeleitete Struktur und ist nur relevant, wenn im vorigen Operanden *NORMAL angegeben ist.

LIST-ALLOWED = *NO / *YES
Bestimmt, ob bei der Kommando- bzw. Anweisungseingabe für den Operandenwert die Angabe einer Liste zulässig ist. Das setzt voraus, dass der Operand, zu dem der Wert gehört, mit LIST-POSSIBLE=*YES definiert ist (siehe ADD-OPERAND). Für Anweisungen und Kommandos, die mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) und STRUCTURE=*YES mit FORM=*FLAT definiert sind, darf nur LIST-ALLOWED=*NO angegeben werden.

VALUE =
Bestimmt, welche Werte als Eingabe zulässig sind.

VALUE = *NO
Alle dem Operandentyp entsprechenden Werte sind zulässig. Einschränkungen gibt es nur, sofern diese bei der Festlegung des Operandentyps angegeben sind (z.B. Längenbegrenzungen). Für Operanden des Typs KEYWORD ist *NO nicht zulässig.

VALUE = list-poss(2000): <c-string 1..1800 with-low>(...)
Die erlaubten Werte sind auf die angegebenen Werte beschränkt. Im Gegensatz zu STANDARD-NAME und ALIAS-NAME kann der Benutzer angegebene Werte des Typs KEYWORD bei der Eingabe abkürzen. Für Werte vom Typ KEYWORD kann keine Liste von Einzelwerten verwendet werden (ein spezifisches ADD-VALUE muss für jeden einzelnen Wert eingegeben werden).

STANDARD-NAME = *NAME / *NO / list-poss(2000): <structured-name 1..30> / <c-string 1..30>
ist nur für Operandenwerte des Typs KEYWORD relevant. Er bestimmt den Standardnamen des Operandenwerts, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Dieser ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME darf im Gegensatz zum ALIAS-NAME nicht gelöscht werden und ist für die Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten.

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
ist nur für Operandenwerte des Typs KEYWORD relevant. Er bestimmt den Aliasnamen des Operandenwerts, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Dieser ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

GUIDED-ABBREVIATION = *NAME / <structured-name 1..30> / <c-string 1..30>
ist nur für Operandenwerte des Typs KEYWORD relevant. Er bestimmt den Namen, mit dem SDF in der mittleren Hilfestufe des geführten Dialogs den Operandenwert bezeichnet. Standardmäßig nimmt SDF-A als GUIDED-ABBREVIATION den Einzelwert, den Sie beim Operanden VALUE angegeben haben.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30> / <c-string 1..30>
nur für Operandenwerte vom Typ KEYWORD:
bestimmt die kürzest mögliche Abkürzung für den Operandenwert. Eine kürzere Abkürzung wird von SDF dann nicht akzeptiert.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar den richtigen Operandenwert auswählt, ihn aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem KEYWORD entstehen.
3. Die ALIAS-NAMEs und STANDARD-NAMEs des Operandenwerts dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Operandenwerts sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (keine Verlängerung) vorgenommen werden.

NULL-ABBREVIATION = *NO / *YES
ist nur für Operandenwerte des Typs KEYWORD relevant, die definiert sind mit STRUCTURE=*YES. Er bestimmt, ob der Operandenwert bei der Kommando- bzw. Anweisungseingabe vor der öffnenden Strukturklammer weggelassen werden kann, z.B. ein struktureinleitender Operandenwert, zu dem es keine alternativen Operandenwerte gibt.

OUTPUT =
Bestimmt, welcher Wert im Fall OUTPUT=*NORMAL (siehe Seite 194) an die Implementierung übergeben wird.

OUTPUT = *SAME
Der beim Operanden VALUE angegebene Wert wird übergeben.

OUTPUT = *EMPTY-STRING
Ein leerer String wird übergeben.

OUTPUT = *DROP-OPERAND
Die Übergabe des Operanden wird unterdrückt.

OUTPUT = <c-string 1..1800>
Der hier angegebene Wert wird übergeben.

OUTPUT = <x-string 1..3600>
Der hier angegebene Wert wird übergeben.

OUT-TYPE = *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST / *COMPOSED-NAME / *C-STRING / *DATE / *DEVICE/ *FIXED/ *FILENAME / *INTEGER / *KEYWORD / *KEYWORD-NUMBER / *LABEL / *NAME / *PARTIAL-FILENAME / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / *STRUCTURED-NAME / *TEXT / *TIME / *VSN / *X-STRING / *X-TEXT
*ist nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*
Bestimmt, ob und ggf. wie SDF den Datentyp des Operandenwertes ändert, wenn der Wert im Übergabebereich für die Implementierung abgelegt wird (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371). Standardmäßig verändert SDF den Datentyp nicht.

OVERWRITE-POSSIBLE = *NO / *YES
*ist nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*
Bestimmt, ob der eingegebene Operandenwert durch einen von der Implementierung dynamisch erzeugten Wert überschrieben wird (siehe Operand DEFAULT in den Makros CMDRST und CMDTST). Der vom Programm erzeugte Wert muss gültiger Operandenwert sein. Im geführten Dialog zeigt SDF den von der Implementierung erzeugten Wert im Fragebogen. Beispiel: Den Wert UNCHANGED in den MODIFY-Anweisungen von SDF-A überschreibt SDF-A mit dem aktuellen Wert.

OUTPUT =
Bestimmt, ob und ggf. wie SDF den eingegebenen Operandenwert an die Implementierung übergibt.

OUTPUT = *NORMAL(...)
SDF übergibt den Operandenwert an die Implementierung. Die Angaben AREA-LENGTH=, LEFT-JUSTIFIED= und FILLER= sind ab SDF-A Version 2.0 nicht mehr auf bestimmte Operandenwerte beschränkt.

AREA-LENGTH = *VARIABLE / <integer 1..3000>
Bestimmt die Länge des Feldes, in dem SDF den Operandenwert für die Implementierung ablegt. Das Feld muss lang genug sein, um den größten Wert aufnehmen zu können, der in der Ablaufphase eingegeben werden kann. Wird für AREA-LENGTH ein kleinerer Wert als für LONGEST-LENGTH eingegeben, gibt SDF eine Warnung aus und akzeptiert den für AREA-LENGTH angegebenen Wert.

Bei der Analyse eines Wertes, der länger als AREA-LENGTH und kürzer als LONGEST-LENGTH ist, können 2 Fälle auftreten:

1. Werte vom Typ C-STRING mit LONGEST-LENGTH ≤ 32, die zu einem geheimen Operanden gehören, werden von SDF komprimiert und in einem sedezimalen String mit der Länge AREA-LENGTH abgelegt. Dieses Verhalten kann zum Beispiel für Kennworte genutzt werden. Die Kennworte werden mithilfe eines Hash-Algorithmus komprimiert und sind durch die sedezimale Ablageform gegen unberechtigte Zugriffe gesichert.

   Hinweise:
   - Es wird derselbe Hash-Algorithmus wie bei der in SDF-P verfügbaren HASH-STRING-Funktion verwendet.
   - Der Kommandoserver oder das Programm, das den von SDF analysierten Wert verarbeitet, muss möglicherweise angepasst werden, wenn die Kennwort-Definition zur Unterstützung von Hash-Kennworten geändert wurde. Eventuell wird der von SDF zurückgegebene Hash-Wert von der Semantik-Analyse des Programmes oder Kommandoservers abgewiesen.

2. In allen von 1. abweichenden Fällen wird der Wert auf die bei AREA-LENGTH angegebene Länge gekürzt.

LEFT-JUSTIFIED = *STD / *YES / *NO
ist nur relevant, wenn für das Ablagefeld eine feste Länge bestimmt wurde. Mit LEFT-JUSTIFIED wird bestimmt, ob SDF den Operandenwert links- oder rechtsbündig in dem Feld ablegt. Bei numerischen Werten setzt SDF-A *STD in *NO um, bei allen anderen Werten in *YES.

FILLER = *STD / <c-string 1..1> / <x-string 1..2>
ist nur relevant, wenn für das Ablagefeld eine feste Länge bestimmt wurde. Mit FILLER wird bestimmt, mit welchem Zeichen SDF das Feld im Bedarfsfall auffüllt. Bei Werten vom Typ X-STRING oder INTEGER setzt SDF-A *STD in X'00' um, bei allen übrigen in C'␣' (Leerzeichen).

STRING-LITERALS = *NO / *HEX / *CHAR
Bestimmt, ob SDF den Operandenwert für die Übergabe an die Implementierung in den Datentyp X-STRING oder C-STRING umwandelt. Standardmäßig verändert SDF den Datentyp nicht. In diesem Fall ist bei Operandenwerten vom Typ C-STRING zu beachten, dass SDF nur den Inhalt des Strings übergibt (ohne Hochkomma und vorangestelltes C). Dieser Operand ist nur gültig, wenn VALUE=*NO angegeben wird.

HASH = *NO / *YES(...)
Legt fest, ob der Eingabewert mittels eines Hash-Algorithmus in einen Wert mit einer definierten Länge umgewandelt wird.

HASH = *YES(...)
ist nur erlaubt für Operandenwerte vom Datentyp C-STRING, die mit LONGEST-LENGTH ≤ 32 definiert werden.
Die anderen Operanden in der Struktur OUTPUT=*NORMAL(...) formatieren den Wert erst nach Ausführung der Hash-Funktion. Der Wert hat anschließend den Datentyp X-STRING und enthält dann eventuell auch nicht druckbare Zeichen.

OUTPUT-LENGTH = <integer 2..32>
Länge des Wertes, in den der Eingabewert umgewandelt wird.

OUTPUT = *SECRET-PROMPT
Der Operandenwert wird nicht an die Implementierung übergeben, sondern bewirkt, dass SDF den Benutzer auffordert, einen der zu dem Operanden gehörenden alternativen Werte einzugeben. Die folgende Eingabe erfolgt dann mit dunkelgesteuertem Eingabefeld und wird nicht protokolliert. Voraussetzungen sind:
– der Operand ist als geheim definiert
(siehe ADD-OPERAND...,SECRET-PROMPT=*YES)
– die Eingabe erfolgt im ungeführten Dialog oder in einer Vordergrund-Prozedur
– für den mit SECRET-PROMPT definierten Operandenwert ist als zulässige Eingabe ein Einzelwert angegeben (siehe Operand VALUE=<c-string>, im Normalfall ist er vom Typ KEYWORD)

Im geführten Dialog ergibt sich folgender Anwendungsfall:
Das Eingabefeld für einen geheimen Operanden, der mit einem nichtgeheimen Wert vorbesetzt ist, wird nicht dunkelgesteuert. Durch Eingabe eines mit OUTPUT=*SECRET-PROMPT definierten Wertes kann das Eingabefeld dunkelgesteuert werden.

PRIVILEGE =
Gibt an, welche Privilegien dem Operandenwert zugeordnet werden.

PRIVILEGE = *SAME
Der Operandenwert erhält die gleichen Privilegien wie der Operand, zu dem dieser Operandenwert definiert wird.

PRIVILEGE = *EXCEPT(...)
Der Operandenwert erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Operandenwert zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Operandenwert erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

CLOSE-CMD-OR-STMT
Kommando- oder Anweisungsdefinition schließen

Mit der Anweisung CLOSE-CMD-OR-STMT beenden Sie die Bearbeitung einer Kommando- oder Anweisungsdefinition und veranlassen, dass SDF-A diese Definition auf Konsistenz prüft. Eventuell noch nicht geschlossene Strukturen werden geschlossen. Falls Fehler festgestellt werden, gibt SDF-A entsprechende Meldungen aus und korrigiert die Fehler durch Standardmaßnahmen.

Wenn Sie mit einer anderen Anweisung, z.B. mit ADD-CMD, ADD-STMT oder END die Bearbeitung einer Kommando- oder Anweisungsdefinition beenden, so veranlassen Sie damit implizit die gleichen Konsistenzprüfungen. Eventuelle Fehlermeldungen zu der beendeten Definition erhalten Sie dann aber vermischt mit eventuellen Meldungen zu der eingegebenen Anweisung.

CLOSE-CMD-OR-STMT

Diese Anweisung hat keine Operanden.

CLOSE-STRUCTURE
Strukturen schließen

Mit der Anweisung CLOSE-STRUCTURE schließen Sie Strukturen in Kommando- und Anweisungsdefinitionen.

CLOSE-STRUCTURE
LEVEL = ***CUR**RENT / ***ALL**

LEVEL =
Bestimmt, welche Strukturen geschlossen werden.

LEVEL = *CURRENT
Die gerade bearbeitete Struktur wird nach dem „aktuellen“ Objekt geschlossen. Der diese Struktur einleitende Operandenwert wird „aktuelles“ Objekt.

LEVEL = *ALL
Die gerade bearbeitete Struktur wird nach dem „aktuellen“ Objekt geschlossen. Alle dieser Struktur übergeordneten Strukturen werden, sofern sie noch offen sind, ebenfalls geschlossen. Der Operandenwert, der die niedrigste dieser Strukturen einleitet, wird aktuelles Objekt.

COMPARE-SYNTAX-FILE
Objekte zweier Syntaxdateien vergleichen

Mit der Anweisung COMPARE-SYNTAX-FILE vergleichen Sie Objekte in zwei verschiedenen Syntaxdateien miteinander.
SDF-A sucht das zu vergleichende Objekt zuerst mit dem externen Namen in der ersten Syntaxdatei. Wenn es vorhanden ist, sucht SDF-A dieses Objekt mit seinem internen Namen in der zweiten Syntaxdatei. Deshalb können nur Objekte mit gleichen internen Namen verglichen werden. Im ausgegebenen Vergleichsprotokoll verwendet SDF-A immer den ersten Standardnamen des Objektes, d.h. bei gleichem internen Namen können in der Ausgabe durchaus unterschiedliche externe Namen für ein Objekt erscheinen.

(Teil 1 von 4)

COMPARE-**SYNT**AX-**FILE**

OBJECT = ***GLOB**AL-**INF**ORMATION / ***DOM**AIN(...) / ***COM**MAND(...) / ***PROG**RAM(...) /
***STATEM**ENT(...) / ***OPER**AND(...) / ***VAL**UE(...)

***DOM**AIN(...)

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***COM**MAND(...)

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***PROG**RAM(...)

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***STATEM**ENT(...)

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

***OPER**AND(...)

OPERAND-**L1** = <structured-name 1..20>

,**VAL**UE-**L1** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

Fortsetzung ➠

(Teil 2 von 4)

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**ORIG**IN = ***COM**MAND(...) / ***STATEM**ENT(...)

***COM**MAND(...)
NAME = <structured-name 1..30>

***STATEM**ENT(...)
NAME = <structured-name 1..30>
,**PROG**RAM = <structured-name 1..30>

***VAL**UE(...)

OPERAND-**L1** = <structured-name 1..20>

,**VAL**UE-**L1** = ***NO** / **COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

Fortsetzung ➠

(Teil 3 von 4)

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L5** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**ORIG**IN = ***COM**MAND(...) / ***STATEM**ENT(...)

***COM**MAND(...)
NAME = <structured-name 1..30>

***STATEM**ENT(...)
NAME = <structured-name 1..30>
,**PROG**RAM = <structured-name 1..30>

Fortsetzung ➠

(Teil 4 von 4)

```
,INPUT-FILE = <filename 1..54 without-gen-vers>(...)
    <filename1..54 without-gen-vers>(...)
        | TYPE = *SYSTEM / *USER / *GROUP
,COMPARE-FILE = <filename 1..54 without-gen-vers>(...)
    <filename 1..54 without-gen-vers>(...)
        | TYPE = *SYSTEM / *USER / *GROUP
,ATTACHED-INFORMATION = *YES / *NO
,OUTPUT = *SYSOUT / *SYSLST(...)
    *SYSLST(...)
        | SYSLST-NUMBER = *STD / <integer 1..99>
,COMPARE-ATTRIBUTES = *ALL / *USER-INTERFACE / *PRIVILEGES-ALLOWNESS
```

OBJECT =
Art des Objekts, dessen Definition zu vergleichen ist.

OBJECT = *GLOBAL-INFORMATION
Die Globalinformationen der Syntaxdateien werden verglichen.

OBJECT = *DOMAIN(...)
Die Definitionen von Anwendungsbereichen werden verglichen.

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen von allen bzw. den namentlich genannten Anwendungsbereichen werden verglichen.

OBJECT = *COMMAND(...)
Die Definitionen von Kommandos werden verglichen.

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen von allen bzw. den namentlich genannten Kommandos werden verglichen.

OBJECT = *PROGRAM(...)
Die Definitionen von Programmen werden verglichen.

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen von allen bzw. den namentlich genannten Programmen werden verglichen.

OBJECT = *STATEMENT(...)
Die Definitionen von Anweisungen werden verglichen.

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen von allen bzw. den namentlich genannten Anweisungen werden verglichen.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisungen gehören.

OBJECT = *OPERAND(...)
Bestimmt, dass die Definition eines Operanden verglichen wird. Steht dieser Operand in einer Struktur, so wird er durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig (oder gar nicht) angegeben werden. Ein Operand, der zur Identifizierung der zu vergleichenden Operandendefinition nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten VALUE-Li=*NO betrachtet SDF-A den durch OPERAND-Li bestimmten Operanden als den, dessen Definition zu vergleichen ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu VALUE-Li ein anderer Wert als *NO angegeben, so muss zu OPERAND-Li+1 ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = <structured-name 1..20>
Bestimmt den Operanden, dessen Definition zu vergleichen ist (VALUE-L1=*NO), oder einen Operanden, der auf dem Pfad zu diesem liegt (VALUE-L1≠*NO).
<structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO bedeutet, dass die Definition von OPERAND-L1 zu vergleichen ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition zu vergleichen ist. Ist der struktureinleitende Operandenwert vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD, ... , VALUE=<c-string>).

Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der struktureinleitende Operandenwert nicht vom Typ KEYWORD bzw. KEYWORD-NUMBER, so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO bedeutet, dass OPERAND-L2 für die Spezifizierung des Operanden, dessen Definition zu vergleichen ist, nicht relevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der, dessen Definition zu vergleichen ist (VALUE-L2=*NO), oder einer, der auf dem Pfad zu diesem liegt (VALUE-L2≠*NO).

VALUE-L2 = analog VALUE-L1
*NO bedeutet, dass VALUE-L2 für die Spezifizierung des Operanden nicht relevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition zu vergleichen ist. Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO bedeutet, dass OPERAND-L3 für die Spezifizierung des Operanden, dessen Definition zu vergleichen ist, nicht relevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der, dessen Definition zu vergleichen ist (VALUE-L3=*NO), oder einer, der auf dem Pfad zu diesem liegt (VALUE-L3≠*NO).

VALUE-L3= analog VALUE-L1
*NO bedeutet, dass VALUE-L3 für die Spezifizierung des Operanden nicht relevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition zu vergleichen ist. Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

VALUE-L4 = analog VALUE-L1
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

ORIGIN =
Kommando oder Anweisung, zu dem die zu vergleichende Operandendefinition gehört.

ORIGIN = *COMMAND(...)
Die Operandendefinition gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Die Operandendefinition gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *VALUE(...)
Die Definition eines Operandenwerts wird verglichen. Dieser Operandenwert wird durch den zu ihm führenden Pfad spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Gehört der Operandenwert zu einem Operanden, der außerhalb jeder Struktur steht, so liegt auf dem Pfad nur dieser Operand. Gehört der Operandenwert zu einem Operanden, der in einer Struktur steht, so liegen auf dem Pfad außerdem die übergeordneten Operanden und die dazugehörenden struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig angegeben werden. Ein Operand, der zur Identifizierung der zu vergleichenden Operandenwertdefinition nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten OPERAND-Li+1=*NO betrachtet SDF-A den durch VALUE-Li bestimmten Operandenwert als den, dessen Definition zu vergleichen ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu OPERAND-Li ein anderer Wert als *NO angegeben, so muss zu VALUE-Li ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = <structured-name 1..20>
Bestimmt den Operanden, zu dem der Operandenwert gehört, dessen Definition zu vergleichen ist (OPERAND-L2=*NO), oder einen Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L2≠*NO).

VALUE-L1=<u>*NO</u> / *COMMAND-REST / *INTEGER / *X-STRING/ *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Bestimmt den Operandenwert, dessen Definition zu vergleichen ist (OPERAND-L2=*NO), oder einen struktureinleitenden Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L2≠*NO). Ist VALUE-L1 vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,..., VALUE= <c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO bedeutet, dass die Definition von VALUE-L1 zu vergleichen ist. Andernfalls ist der Name des Operanden anzugeben, zu dem die zu vergleichende Operandenwertdefinition gehört (OPERAND-L3=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L3≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig sein.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO bedeutet, dass VALUE-L2 für die Spezifizierung des zu vergleichenden Operandenwerts nicht relevant ist. Andernfalls ist ein Operandenwert anzugeben; entweder der Operandenwert, dessen Definition zu vergleichen ist (OPERAND-L3=*NO), oder ein struktureinleitender Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L3≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO bedeutet, dass OPERAND-L3 für die Spezifizierung der zu vergleichenden Operandenwertdefinition nicht relevant ist. Andernfalls ist der Name des Operanden anzugeben, zu dem die zu vergleichende Operandenwertdefinition gehört (OPERAND-L4=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L4≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig sein.

VALUE-L3 = analog VALUE-L2
*NO bedeutet, dass VALUE-L3 für die Spezifizierung der zu vergleichenden Operandenwertdefinition nicht relevant ist. Andernfalls ist ein Operandenwert anzugeben; entweder der Operandenwert, dessen Definition zu vergleichen ist (OPERAND-L4=*NO), oder ein struktureinleitender Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L4≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L4 = analog VALUE-2
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L5= analog VALUE-2
siehe VALUE-L2.

ORIGIN =
Kommando oder Anweisung, zu dem die zu vergleichende Operandenwertdefinition gehört.

ORIGIN = *COMMAND(...)
Die Operandenwertdefinition gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Die Operandenwertdefinition gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

INPUT-FILE = <filename 1..54 without-gen-vers>(...)
Syntaxdatei, die die zu vergleichenden Definitionen bzw. die zu vergleichende Globalinformation enthält.

TYPE = *SYSTEM / *GROUP / *USER
Typ der Syntaxdatei mit den zu vergleichenden Definitionen bzw. der zu vergleichenden Globalinformation: Systemsyntaxdatei, Gruppensyntaxdatei oder Benutzersyntaxdatei.

COMPARE-FILE = <filename 1..54 without-gen-vers>(...)
Syntaxdatei, mit der die Definitionen bzw. die Globalinformation aus INPUT-FILE verglichen werden.

TYPE = *SYSTEM / *GROUP / *USER
Typ der Syntaxdatei:
Systemsyntaxdatei, Gruppensyntaxdatei oder Benutzersyntaxdatei

ATTACHED-INFO =
Bestimmt, welche der Definitionen ausgegeben werden, die zu dem ausgewählten Objekt gehören.

ATTACHED-INFO = *YES
Die Definition des ausgewählten Objekts wird einschließlich der Definitionen aller Objekte ausgegeben, die diesem Objekt zugeordnet sind (Anwendungsbereich mit zugeordneten Kommandos, Programm mit zugehörigen Anweisungen, Kommando oder Anweisung mit zugehörigen Operanden, Operand mit zugehörigen Operandenwerten, Operandenwert mit zugehöriger Struktur, Globalinformation mit sprachabhängigen Texten).

ATTACHED-INFO = *NO
Die Definition des ausgewählten Objekts wird ohne die Definitionen der Objekte ausgegeben, die diesem Objekt zugeordnet sind (Anwendungsbereich ohne Kommandozuordnungen, Programm ohne zugehörige Anweisungen, Kommando oder Anweisung ohne zugehörige Operanden, Operand ohne zugehörige Operandenwerte, Operandenwert ohne zugehörige Struktur, Globalinformation ohne sprachabhängige Texte).

OUTPUT =
Bestimmt das Ausgabemedium für das Vergleichsprotokoll.

OUTPUT = *SYSOUT
Die Ausgabe erfolgt in die logische Systemdatei SYSOUT, d.h. im Dialog in der Regel auf den Bildschirm.

OUTPUT = *SYSLST(...)
Die Ausgabe erfolgt in die logische Systemdatei SYSLST.

SYSLST-NUMBER = *STD / <integer 1..99>
Bestimmt die Nummer der logischen Systemdatei SYSLST. Bei Angabe von *STD erhält die logische Systemdatei SYSLST keine Nummer.

COMPARE-ATTRIBUTES =
Legt die Vergleichs-Attribute fest

COMPARE-ATTRIBUTES = *ALL
Die komplette Objekt-Definition wird verglichen.

COMPARE-ATTRIBUTES = *USER-INTERFACE
Verglichen werden
– die Hilfetexte des Kommandos, der Anweisung, des Anwendungsbereiches oder des Operanden
– die Zusätze zum Datentyp eines Operandenwertes (z.B. <integer 1..99>)
– die Kommentarzeile, die einem Programm zugeordnet ist.

COMPARE-ATTRIBUTES = *PRIVILEGES-ALLOWNESS
Verglichen werden
– die Eingabemodi (DIALOG-ALLOWED, DIALOG-PROC-ALLOWED, BATCH-ALLOWED usw.)
– die Privilegien, die dem Objekt zugeordnet sind.

Ausgaben der Anweisung COMPARE-SYNTAX-FILE

Wenn die zu vergleichenden Objekte identisch definiert sind, werden nur die Namen der beiden Syntaxdateien und anschließend die Meldung `END OF COMPARISON` ausgegeben.

Sind die zu vergleichenden Objekte nicht identisch definiert, dann gibt SDF-A folgende Informationen aus:

- Namen der beiden Syntaxdateien
- Identifizierung der Objekte, die unterschiedlich definiert sind
- Unterschiede in der Definition

Namen der beiden Syntaxdateien

Die Ziffer in der ersten Spalte identifiziert die Syntaxdatei:

1: Syntaxdatei, die bei INPUT-FILE angegeben wurde
2: Syntaxdatei, die bei COMPARE-FILE angegeben wurde

Fehlt die Ziffer in der ersten Spalte, dann betrifft die darauffolgende Information beide Syntaxdateien.

Identifizierung der Objekte, die unterschiedlich definiert sind

SDF-A sucht jedes Objekt zuerst mit dem externen Namen in der Syntaxdatei 1 (INPUT-FILE). Wenn es vorhanden ist, sucht SDF-A dieses Objekt mit seinem internen Namen in der Syntaxdatei 2 (COMPARE-FILE).

Der Identifikator im Vergleichsprotokoll beginnt immer mit dem Objekttyp, gefolgt von einem Doppelpunkt und der genauen Pfadangabe zum Objekt. Im folgenden Text bedeutet:

<domain-name>	Name des Anwendungsbereiches
<command-name>	Erster STANDARD-NAME des Kommandos (siehe ADD-CMD, Seite 135)
<program-name>	Name des Programmes
<statement-name>	Erster STANDARD-NAME der Anweisung (siehe ADD-STMT, Seite 164)
<operand-name>	Erster STANDARD-NAME des Operanden (siehe ADD-OPERAND, Seite 151)
<value-name>	Erster STANDARD-NAME des Operandenwertes, wenn dieser vom Datentyp KEYWORD(-NUMBER) ist; ansonsten der Datentyp des Operandenwertes in Kleinbuchstaben

DOMAIN:<domain-name>
Es gibt Unterschiede in der Definition eines Anwendungsbereichs.

COMMAND:<domain-name>.<command-name>
Es gibt Unterschiede in der Kommandodefinition und der Vergleich wurde mit OBJECT=*DOMAIN(...) durchgeführt.

COMMAND:<command-name>
Es gibt Unterschiede in der Kommandodefinition und der Vergleich wurde mit OBJECT=*COMMAND(...) durchgeführt.

PROGRAM:<program-name>
Es gibt Unterschiede in der Programmdefinition.

STATEMENT:<program-name>.<statement-name>
Es gibt Unterschiede in der Anweisungsdefinition.

OPERAND:<domain-name>.<command-name>.<operand-name>
Es gibt Unterschiede in der Operandendefinition eines Kommandos und der Vergleich wurde mit OBJECT=*DOMAIN(...) durchgeführt.

OPERAND:<command-name>.<operand-name>
Es gibt Unterschiede in der Operandendefinition eines Kommandos und der Vergleich wurde mit OBJECT=*COMMAND(...) durchgeführt.

OPERAND:<program-name>.<statement-name>.<operand-name>
Es gibt Unterschiede in der Operandendefinition einer Anweisung und der Vergleich wurde mit OBJECT=*PROGRAM(...) durchgeführt.

VALUE:<domain-name>.<command-name>.<operand-name>.<value-name>
Es gibt Unterschiede in der Operandenwertdefinition in einem Kommando und der Vergleich wurde mit OBJECT=*DOMAIN(...) durchgeführt.

VALUE:<command-name>.<operand-name>.<value-name>
Es gibt Unterschiede in der Operandenwertdefinition in einem Kommando und der Vergleich wurde mit OBJECT=*COMMAND(...) durchgeführt.

VALUE:<program-name>.<statement-name>.<operand-name>.<value-name>
Es gibt Unterschiede in der Operandenwertdefinition einer Anweisung und der Vergleich wurde mit OBJECT=*PROGRAM(...) durchgeführt.

Unterschiede zwischen den verglichenen Objekten

Direkt nach dem Objekt-Identifikator wird das Vergleichsresultat ausgegeben.
Zeilen ohne die Ziffer 1 oder 2 in der ersten Spalte betreffen beide Syntaxdateien und enthalten z.B. den Text 'NOT DEFINED' als Hinweis darauf, dass das zu vergleichende Objekt in beiden Syntaxdateien nicht definiert ist. Nachfolgend finden Sie einige Beispiele für Ausgabezeilen in einem Vergleichsprotokoll und eine kurze Erklärung dazu.

Beispiele für nicht definierte Objekte

`PROGRAM NOT DEFINED`
In beiden Syntaxdateien ist kein Programm definiert. Beim Vergleich wurde OBJECT=*PROGRAM(NAME=*ALL) angegeben.

`PROGRAM:SDF-A NOT DEFINED`
Das Programm SDF-A ist weder in Syntaxdatei 1 (INPUT-FILE) noch in Syntaxdatei 2 (COMPARE-FILE) definiert.

`1 OPERAND:MODIFY-SDF-OPTIONS.MODE.TEST.CHECK-PRIVILEGES NOT DEFINED`
In Syntaxdatei 1 ist der Operand CHECK-PRIVILEGE in der Struktur MODE=*TEST(...) im Kommando MODIFY-SDF-OPTIONS nicht definiert.

`2 COMMAND:SDF.RESET-INPUT-DEFAULTS NOT DEFINED`
In Syntaxdatei 2 ist das Kommando RESET-INPUT-DEFAULTS im Anwendungsbereich SDF nicht definiert.

```
1 COMMAND:MODIFY-SDF-OPTIONS NOT DEFINED
2 COMMAND:MODIFY-SDF-OPTIONS NOT DEFINED
```
Das Kommando MODIFY-SDF-OPTIONS hat in Syntaxdatei 2 nicht den gleichen internen Namen wie in Syntaxdatei1.

Beispiele für unterschiedliche Objekte

Unterschiede werden in Form des Teils der SDF-A-Anweisung dargestellt, mit der das betroffene Objekt definiert wurde; z.B. ADD-CMD bei Unterschieden zwischen Kommandos, ADD-DOMAIN bei Unterschieden zwischen Anwendungsbereichen oder SET-GLOBALS bei unterschiedlichen Globalinformationen.

```
1 COMMAND:MODIFY-SDF-OPTIONS HELP=(E('help from input-file'))
2 COMMAND:MODIFY-SDF-OPTIONS HELP=(E('help from compare-file'))
```
Die Hilfetexte für das Kommando MODIFY-SDF-OPTIONS unterscheiden sich.
Der Unterschied wird wie ein Teil der Anweisung ADD-CMD NAME=MODIFY-SDF-OPTIONS,...,HELP=(E('...')) dargestellt.

`1 COMMAND:MODIFY-SDF-OPTIONS HELP=(E('help from compare-file')) NOT DEFINED`
In Syntaxdatei 1 ist für das Kommando MODIFY-SDF-OPTIONS kein englischer Hilfetext definiert (der Hilfetext 'help from compare-file' stammt aus Syntaxdatei 2 und fehlt in Syntaxdatei 1).

```
1 VALUE:SDF.MODIFY-SDF-OPTIONS.MODE.TEST STRUCTURE=YES
2 VALUE:SDF.MODIFY-SDF-OPTIONS.MODE.TEST STRUCTURE=NO
```
Der Wert TEST im Operanden MODE des Kommandos MODIFY-SDF-OPTIONS (Anwendungsbereich SDF) ist in Syntaxdatei 1 mit STRUCTURE=*YES definiert, in Syntaxdatei 2 jedoch mit STRUCTURE=*NO.

COPY
Inhalte einer Syntaxdatei kopieren

Mit der Anweisung COPY kopieren Sie Inhalte einer Syntaxdatei. Die Information über die Sperrung von Objekten (siehe REMOVE, Seite 307) wird nicht kopiert. Die Kopien fügt SDF-A in die bearbeitete Syntaxdatei ein.

Definitionen von BS2000-Kommandos (durch System-Module implementiert) können Sie in eine Benutzersyntaxdatei nur dann kopieren, wenn diese durch die zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE, Seite 303) voll abgedeckt sind. Das Gleiche gilt für den Fall, dass Sie eine Operanden- oder Operandenwertdefinition in die Definition eines durch System-Module implementierten Kommandos kopieren.

Wenn Sie die Globalinformation kopieren, überschreibt SDF-A die in der geöffneten Syntaxdatei vorhandene Globalinformation.

Die SDF-Standardanweisungen können und müssen Sie nicht in Ihre Syntaxdatei kopieren. SDF stellt diese Anweisungen automatisch jedem Programm zur Verfügung, dessen Anweisungen in Syntaxdateien definiert sind. In der Syntaxdatei von SDF ab V4.0A sind alle von Fujitsu Siemens Computers freigegebenen Standardanweisungen zentral hinterlegt.

Bevor Sie die Definition eines Operanden oder Operandenwerts kopieren, müssen Sie dafür sorgen, dass das Objekt (z.B. ein anderer Operand) das aktuelle ist, nach dessen Definition Sie die Kopie in eine Kommando- bzw. Anweisungsdefinition einfügen wollen (siehe ADD-OPERAND bzw. ADD-VALUE). In dieser Umgebung darf der Operand bzw. der Operandenwert noch nicht definiert sein. Andernfalls lehnt SDF-A das Kopieren mit einer entsprechenden Meldung ab.

Das mit SDF bis V3.0A ausgelieferte Kommando SHOW-SYNTAX-VERSIONS kann mit COPY OBJECT=*COMMAND (NAME=*ALL) nicht in eine Benutzersyntaxdatei kopiert werden. Ab SDF V4.0A wird SHOW-SYNTAX-VERSIONS genau wie jedes andere Kommando behandelt und kann demzufolge auch in eine Benutzersyntaxdatei kopiert werden.

Es gibt bestimmte Operandenwerte, die kontextabhängig sind. Zum Beispiel wird ein Wert, der mit TYPE=*INTEGER(OUT-FORM=*STD) definiert ist, je nach Implementierungsform unterschiedlich konvertiert:

– Bei Kommandos, die mit IMPLEMENTOR=*PROCEDURE oder IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...) definiert sind, wird OUT-FORM=*STD in OUT-FORM=*CHAR umgesetzt.

– Bei Kommandos, die mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definiert sind, wird OUT-FORM=*STD in OUT-FORM=*BINARY umgesetzt.

Wird ein solches Objekt aus einem Kontext (z.B. ...,CMD-INTERFACE=*NEW bzw. *TRANSFER-AREA,...) in einen anderen Kontext (z.B. ...,CMD-INTERFACE=*STRING,...) kopiert, so müssen Sie die Anpassung an den Kontext selbst vornehmen. Dazu ist mit der Anweisung EDIT auf das entsprechende Objekt zu positionieren und das Objekt mit einer MODIFY-Anweisung zu ändern.

(Teil 1 von 3)

COPY

OBJECT = ***GLOB**AL-**INF**ORMATION / ***DOM**AIN(...) / ***COM**MAND(...) / ***PROG**RAM(...) /
***STATEM**ENT(...) / ***OPER**AND(...) / ***VAL**UE(...)

***DOM**AIN(...)
NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
***ALL**(...)
EXCEPT = ***NONE** / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>

***COM**MAND(...)
NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
***ALL**(...)
EXCEPT = ***NONE** / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>

***PROG**RAM(...)
NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
***ALL**(...)
EXCEPT = ***NONE** / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>

***STATEM**ENT(...)
NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
***ALL**(...)
EXCEPT = ***NONE** / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
,**PROG**RAM = <structured-name 1..30>

***OPER**AND(...)
OPERAND-**L1** = ***CUR**RENT / <structured-name 1..20>
,**VAL**UE-**L1** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>
,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

Fortsetzung ➠

(Teil 2 von 3)

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**ORIG**IN = ***CUR**RENT / ***COM**MAND(...) / ***STATEM**ENT(...)

***COM**MAND(...)
NAME = <structured-name 1..30>

***STATEM**ENT(...)
NAME = <structured-name 1..30>
,**PROG**RAM = <structured-name 1..30>

***VAL**UE(...)

OPERAND-**L1** = ***ABOV**E-**CUR**RENT / <structured-name 1..20>

,**VAL**UE-**L1** = ***CUR**RENT / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

Fortsetzung ➠

(Teil 3 von 3)

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L5** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**ORIG**IN = ***CUR**RENT / ***COM**MAND(...) / ***STATEM**ENT(...)

***COM**MAND(...)
NAME = <structured-name 1..30>

***STATEM**ENT(...)
NAME = <structured-name 1..30>
,**PROG**RAM = <structured-name 1..30>

,**FROM-F**ILE = ***CUR**RENT / ***TASK-H**IERARCHY / <filename 1..54 without-gen-vers>(...)

<filename 1..54 without-gen-vers>(...)
TYPE = ***CUR**RENT / ***USER** / ***GROUP** / ***SYST**EM

,**ATT**ACHED-**INF**O = ***YES** / ***NO** / ***ONLY**

,**OVERWR**ITE-**POSSIBLE** = ***NO** / ***YES** / ***EXTERN**AL-**ATTR**IBUTES

OBJECT =
Art des Objekts, dessen Definition zu kopieren ist.

OBJECT = *GLOBAL-INFORMATION
Bestimmt, dass die Globalinformation einer Syntaxdatei kopiert wird.

OBJECT = *DOMAIN(...)
Bestimmt, dass die Definitionen von Anwendungsbereichen kopiert werden.

NAME = *ALL(...)
Die Definitionen von allen Anwendungsbereichen werden kopiert.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Anwendungsbereiche werden nicht kopiert.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Anwendungsbereiche werden kopiert, bzw. die Definitionen der Anwendungsbereiche, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *COMMAND(...)
Bestimmt, dass die Definitionen von Kommandos kopiert werden.

NAME = *ALL(...)
Die Definitionen von allen Kommandos werden kopiert.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Kommandos werden nicht kopiert.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Kommandos werden kopiert, bzw. die Definitionen der Kommandos, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *PROGRAM(...)
Bestimmt, dass die Definitionen von Programmen kopiert werden.

NAME = *ALL(...)
Die Definitionen von allen Programmen werden kopiert.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Programme werden nicht kopiert.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Programme werden kopiert, bzw. die Definitionen der Programme, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *STATEMENT(...)
Bestimmt, dass die Definitionen von Anweisungen kopiert werden.

NAME = *ALL(...)
Die Definitionen von allen Anweisungen werden kopiert.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Anweisungen werden nicht kopiert.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Anweisungen werden kopiert, bzw. die Definitionen der Anweisungen, deren Name zum Wildcard-Suchmuster passt.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisungen gehören. Das Programm muss in der geöffneten Syntaxdatei bereits definiert sein.

OBJECT = *OPERAND(...)
Bestimmt, dass die Definition eines Operanden kopiert wird. Steht dieser Operand in einer Struktur, so wird er durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig (oder gar nicht) angegeben werden. Ein Operand, der zur Identifizierung der zu kopierenden Operandendefinition nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten VALUE-Li=*NO betrachtet SDF-A den durch OPERAND-Li bestimmten Operanden als den, dessen Definition zu kopieren ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu VALUE-Li ein anderer Wert als *NO angegeben, so muss zu OPERAND-Li+1 ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Bestimmt den Operanden, dessen Definition zu kopieren ist (VALUE-L1=*NO), oder einen Operanden, der auf dem Pfad zu diesem liegt (VALUE-L1≠*NO). *CURRENT bedeutet, dass OPERAND-L1 aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Die Angabe *NO bedeutet, dass die Definition von OPERAND-L1 zu kopieren ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition zu kopieren ist. Ist der struktureinleitende Operandenwert vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,..., VALUE= <c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der struktureinleitende Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L2 für die Spezifizierung des Operanden, dessen Definition zu kopieren ist, irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der, dessen Definition zu kopieren ist (VALUE-L2=*NO), oder einer, der auf dem Pfad zu diesem liegt (VALUE-L2≠*NO).

VALUE-L2 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition zu kopieren ist. Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung des Operanden, dessen Definition zu kopieren ist, irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der, dessen Definition zu kopieren ist (VALUE-L3=*NO), oder einer, der auf dem Pfad zu diesem liegt (VALUE-L3≠*NO).

VALUE-L3= analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung des Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition zu kopieren ist. Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

VALUE-L4 = analog VALUE-L1
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, zu dem die zu kopierende Operandendefinition gehört.

ORIGIN = *CURRENT
Die Operandendefinition gehört zu demselben Kommando (bzw. zu derselben Anweisung), in das SDF-A die Kopie einfügen soll.

ORIGIN = *COMMAND(...)
Die Operandendefinition gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Die Operandendefinition gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *VALUE(...)
Bestimmt, dass die Definition eines Operandenwerts kopiert wird. Dieser Operandenwert wird durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Gehört der Operandenwert zu einem Operanden, der außerhalb jeder Struktur steht, so liegt auf dem Pfad nur dieser Operand. Gehört der Operandenwert zu einem Operanden, der in einer Struktur steht, so liegen auf dem Pfad außerdem die übergeordneten Operanden und die zu diesen gehörenden struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig angegeben werden. Ein Operand, der zur Identifizierung der zu kopierenden Operandenwertdefinition nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten OPERAND-Li+1=*NO betrachtet SDF-A den durch VALUE-Li bestimmten Operandenwert als den, dessen Definition zu kopieren ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu OPERAND-Li ein anderer Wert als *NO angegeben, so muss zu VALUE-Li ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Bestimmt den Operanden, zu dem der Operandenwert, dessen Definition zu kopieren ist, gehört (OPERAND-L2=*NO), oder einen Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L2≠*NO).
*ABOVE-CURRENT bedeutet, dass ein zu OPERAND-L1 gehörender Wert aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1=*CURRENT / *COMMAND-REST / *INTEGER / *X-STRING/ *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Bestimmt den Operandenwert, dessen Definition zu kopieren ist (OPERAND-L2=*NO), oder einen struktureinleitenden Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L2≠*NO). *CURRENT bedeutet, dass VALUE-L1 aktuelles Objekt ist. Ist er nicht aktuelles Objekt und vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,..., VALUE=<c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass die Definition von VALUE-L1 zu kopieren ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der Operandenwert, dessen Definition zu kopieren ist, gehört (OPERAND-L3=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L3≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig sein.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des zu kopierenden Operandenwerts irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser ist entweder der Operandenwert, dessen Definition zu kopieren ist (OPERAND-L3=*NO), oder ein struktureinleitender Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L3≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung der zu kopierenden Operandenwertdefinition irrelevant ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der Operandenwert, dessen Definition zu kopieren ist, gehört (OPERAND-L4=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L4≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig sein.

VALUE-L3 = analog VALUE-L2
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung der zu kopierenden Operandenwertdefinition irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser ist entweder der Operandenwert, dessen Definition zu kopieren ist (OPERAND-L4=*NO), oder ein struktureinleitender Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L4≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L4 = analog VALUE-2
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L5= analog VALUE-2
siehe VALUE-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, zu dem die zu kopierende Operandenwertdefinition gehört.

ORIGIN = *CURRENT
Die Operandenwertdefinition gehört zu demselben Kommando (bzw. zu derselben Anweisung), in das SDF-A die Kopie einfügen soll.

ORIGIN = *COMMAND(...)
Die Operandenwertdefinition gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Die Operandenwertdefinition gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

FROM-FILE =
Syntaxdatei, die die zu kopierenden Definitionen bzw. die zu kopierende Globalinformation enthält.

FROM-FILE = *CURRENT
Die zurzeit bearbeitete Syntaxdatei enthält die zu kopierenden Definitionen. Das ist nur möglich beim Kopieren von Operanden- oder Operandenwertdefinitionen.

FROM-FILE = *TASK-HIERARCHY
Die zu kopierende Definition wird aus einer der Syntaxdateien geholt, die zur aktuellen Syntaxdatei-Hierarchie der Task gehören.

FROM-FILE = <filename 1..54 without-gen-vers>(...)
Die genannte Syntaxdatei enthält die zu kopierenden Definitionen bzw. die zu kopierende Globalinformation.

TYPE = *CURRENT / *USER / *GROUP / *SYSTEM
Die Syntaxdatei mit den zu kopierenden Definitionen bzw. der zu kopierende Globalinformation ist

*CURRENT	von der gleichen Art, wie die bearbeitete Syntaxdatei
*USER	eine Benutzersyntaxdatei
*GROUP	eine Gruppensyntaxdatei
*SYSTEM	eine Systemsyntaxdatei.

ATTACHED-INFO =
Bestimmt, welche der Definitionen, die zu dem spezifizierten Objekt gehören, kopiert werden. Für die Globalinformation, Programme und Anwendungsbereiche interpretiert SDF-A den Wert ONLY als *YES.

ATTACHED-INFO = *YES
Die Definition des spezifizierten Objekts wird einschließlich der Definitionen aller Objekte kopiert, die dem spezifizierten zugeordnet sind. (Anwendungsbereich mit zugeordneten Kommandos, Programm mit zugehörigen Anweisungen, Kommando oder Anweisung mit zugehörigen Operanden, Operand mit zugehörigen Operandenwerten, Operandenwert mit zugehöriger Struktur, Globalinformation mit sprachabhängigen Texten.)

ATTACHED-INFO = *NO
Die Definition des spezifizierten Objekts wird ohne die Definitionen der Objekte kopiert, die dem spezifizierten zugeordnet sind. (Anwendungsbereich ohne Kommandozuordnungen, Programm ohne zugehörige Anweisungen, Kommando oder Anweisung ohne zugehörige Operanden, Operand ohne zugehörige Operandenwerte, Operandenwert ohne zugehörige Struktur, Globalinformation ohne sprachabhängige Texte.)

ATTACHED-INFO = *ONLY
Ausschließlich die Definitionen der Objekte, die dem spezifizierten zugeordnet sind, werden kopiert. Die Definition des spezifizierten Objekts selbst wird nicht kopiert. (Beispiele: Operandenwerte ohne den Operanden, zu dem sie gehören; eine Struktur ohne den struktureinleitenden Operandenwert.)

OVERWRITE-POSSIBLE =
Bestimmt, ob die Definition eines Anwendungsbereichs, eines Kommandos, eines Programms oder einer Anweisung kopiert wird, wenn dieses Objekt bereits in der geöffneten Syntaxdatei definiert ist.

OVERWRITE-POSSIBLE = *NO
SDF-A lehnt das Kopieren eines Anwendungsbereichs, eines Kommandos, eines Programms oder einer Anweisung mit einer entsprechenden Meldung ab, wenn dieses Objekt bereits in einer bearbeiteten Syntaxdatei definiert ist. Das Kopieren der Globalinformation ist möglich.

OVERWRITE-POSSIBLE = *YES
SDF-A kopiert unabhängig davon, ob das Objekt bereits in der geöffneten Syntaxdatei definiert ist. Dabei ersetzt SDF-A ggf. die in der geöffneten Syntaxdatei vorhandene Definition durch die zu kopierende Definition. Ein Kommando oder eine Anweisung kann nur dann überschrieben werden, wenn:
– NAME oder STANDARD-NAME identisch sind und
– INTERNAL-NAME identisch ist.

OVERWRITE-POSSIBLE = *EXTERNAL-ATTRIBUTES
SDF-A kopiert nur die Objekte ohne die Definition der Objekte selbst. Die Definitionen der Objekte, die sich an den überschriebenen Anwendungsbereich oder an das Programm anschließen (z.B. Kommandos oder Anweisungen), bleiben erhalten. Die Angabe dieses Operanden ist nur möglich für das Kopieren von Anwendungsbereichen und Programmen (COPY OBJ =*DOMAIN...oder OBJ=*PROGRAM...). Der Operand ATTACHED-INFO erhält den Wert *NO, unabhängig davon, ob Sie einen anderen Wert angegeben haben.

DEFINE-ENVIRONMENT
Syntaxdatei-Format und Version festlegen

Mit der Anweisung DEFINE-ENVIRONMENT legen Sie fest, mit welcher SDF-A-Version Sie arbeiten. Durch die verwendete SDF-A-Version wird auch das Syntaxdatei-Format bestimmt, mit der die bearbeitete Syntaxdatei abgespeichert wird. Die Anweisung muss vor dem Erzeugen oder Öffnen einer Syntaxdatei ausgeführt werden, wenn Sie nicht mit der höchsten SDF-A-Version arbeiten wollen. Eine Syntaxdatei kann nie mit einer kleineren SDF-A-Version geändert werden als die SDF-A-Version, mit der sie erzeugt oder abgespeichert wurde.

DEFINE-**ENVIRON**MENT
SYNTAX-FILE-FORMAT = ***CURRENT** / ***V4.1** / ***V4** / ***V3**

SYNTAX-FILE-FORMAT =
Bestimmt die SDF-A-Version, mit der Sie arbeiten.

SYNTAX-FILE-FORMAT = *CURRENT / *V4.1
SDF-A V4.1 bleibt geladen. Die nachfolgend bearbeiteten Syntaxdateien werden im V4.1-Format abgespeichert.

SYNTAX-FILE-FORMAT = *V4
SDF-A V4.0 wird geladen. Danach steht der komplette Funktionsumfang von SDF-A V4.0 zur Verfügung. Die nachfolgend bearbeiteten Syntaxdateien werden im V4-Format abgespeichert.

SYNTAX-FILE-FORMAT = *V3
SDF-A V3.0 wird geladen. Danach steht der komplette Funktionsumfang von SDF-A V3.0 zur Verfügung. Die nachfolgend bearbeiteten Syntaxdateien werden im V3-Format abgespeichert.

EDIT
Auf ein Objekt der Syntaxdatei positionieren

Mit der Anweisung EDIT erklären Sie die Globalinformation der Syntaxdatei oder einen Anwendungsbereich, ein Programm, ein Kommando, eine Anweisung, einen Operanden oder einen Operandenwert zum „aktuellen“ Objekt (siehe Seite 151). Mit EDIT lässt sich auch auf ein Objekt positionieren, das nicht in der geöffneten Syntaxdatei steht, sondern in der Gruppen- bzw. Systemsyntaxdatei, die Sie beim Öffnen der zu bearbeitenden Syntaxdatei mit dem Operanden GROUP- bzw. SYSTEM-DESCRIPTION angegeben haben.

Wenn sie mit einer MODIFY-Anweisung die Definition eines Objekts ändern wollen, müssen Sie zuvor dafür sorgen, dass dieses das „aktuelle“ ist. Wollen Sie mit einer ADD- oder COPY-Anweisung die Definition eines Operanden oder Operandenwerts in eine Kommando- oder Anweisungsdefinition einfügen, so müssen Sie ebenfalls dafür sorgen, dass das Objekt, nach dessen Definition eingefügt werden soll, das „aktuelle“ ist. Wenn Sie (im geführten Dialog) mit der Anweisung SET-GLOBALS die Globalinformation ändern, werden Ihnen die aktuellen Werte nur dann angezeigt, wenn die Globalinformation aktuelles Objekt ist.

(Teil 1 von 4)

EDIT

OBJECT = ***GLOB**AL-**INF**ORMATION / ***PRIV**ILEGE(...) / ***DOM**AIN(...) / ***COM**MAND(...) / ***PROG**RAM(...) / ***STATEM**ENT(...) / ***OPER**AND(...) / ***VAL**UE(...)

***PRIV**ILEGE(...)
  NAME = <structured-name 1..30>

***DOM**AIN(...)
  NAME = <structured-name 1..30>

***COM**MAND(...)
  NAME = <structured-name 1..30>

***PROG**RAM(...)
  NAME = <structured-name 1..30>

***STATEM**ENT(...)
  NAME = <structured-name 1..30>
  ,**PROG**RAM = <structured-name 1..30>

Fortsetzung ➠

(Teil 2 von 4)

```
*OPERAND(...)
    OPERAND-L1 = *CURRENT / <structured-name 1..20>
    ,VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME /
                <composed-name 1..30>
    ,OPERAND-L2 = *NO / <structured-name 1..20>
    ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME /
                <composed-name 1..30>
    ,OPERAND-L3 = *NO / <structured-name 1..20>
    ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME /
                <composed-name 1..30>
    ,OPERAND-L4 = *NO / <structured-name 1..20>
    ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME /
                <composed-name 1..30>
    ,OPERAND-L5 = *NO / <structured-name 1..20>
    ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
        *COMMAND(...)
            NAME = <structured-name 1..30>
        *STATEMENT(...)
            NAME = <structured-name 1..30>
            ,PROGRAM = <structured-name 1..30>
```

Fortsetzung ➠

(Teil 3 von 4)

***VAL**UE(...)

OPERAND-**L1** = ***ABOV**E-**CUR**RENT / <structured-name 1..20>

,**VAL**UE-**L1** = ***CUR**RENT / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

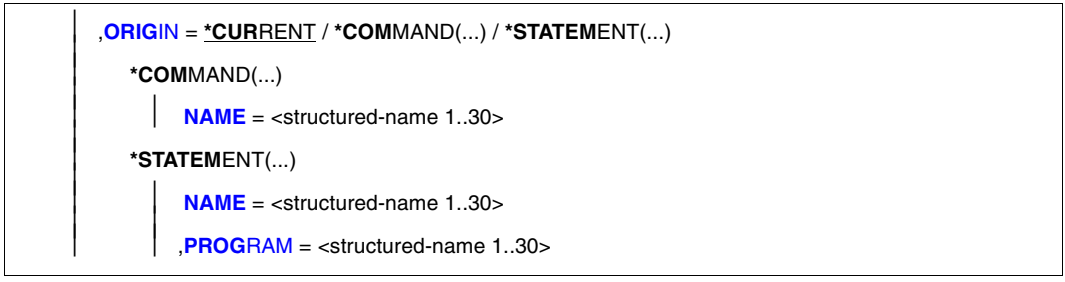
,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L5** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** /
<composed-name 1..30>

Fortsetzung ➠

(Teil 4 von 4)



OBJECT =
Art des Objekts, das zum aktuellen erklärt wird.

OBJECT = *GLOBAL-INFORMATION
Die Globalinformation der Syntaxdatei wird aktuelles Objekt.

OBJECT = *PRIVILEGE(...)
Das angegebene Privileg wird aktuelles Objekt. Dieser Operandenwert ist reserviert für die Systemsoftwareentwicklung von Fujitsu Siemens Computers.

NAME = <structured-name 1..30>
Name des Privilegs.

OBJECT = *DOMAIN(...)
Ein Anwendungsbereich wird aktuelles Objekt.

NAME = <structured-name 1..30>
Name des Anwendungsbereichs.

OBJECT = *COMMAND(...)
Ein Kommando wird aktuelles Objekt.

NAME = <structured-name 1..30>
Name des Kommandos.

OBJECT = *PROGRAM(...)
Ein Programm wird aktuelles Objekt.

NAME = <structured-name 1..30>
Name des Programms.

OBJECT = *STATEMENT(...)
Eine Anweisung wird aktuelles Objekt.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *OPERAND(...)
Ein Operand wird aktuelles Objekt. Steht der Operand, der aktuelles Objekt wird, in einer Struktur, so wird er durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig (oder gar nicht) angegeben werden. Ein Operand, der zur Identifizierung des zum aktuellen Objekt werdenden Operanden nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten VALUE-Li=*NO betrachtet SDF-A den durch OPERAND-Li bestimmten Operanden als den, der aktuelles Objekt wird. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu VALUE-Li ein anderer Wert als *NO angegeben, so muss zu OPERAND-Li+1 ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Bestimmt den zum aktuellen Objekt werdenden Operanden (VALUE-L1=*NO) oder einen Operanden, der auf dem Pfad zu diesem liegt (VALUE-L1≠*NO). *CURRENT bedeutet, dass OPERAND-L1 aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Die Angabe *NO bedeutet, dass OPERAND-L1 aktuelles Objekt wird. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet, die den zum aktuellen Objekt werdenden Operanden unmittelbar oder mittelbar enthält. Ist der struktureinleitende Operandenwert vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der struktureinleitende Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L2 für die Spezifizierung des zum aktuellen Objekt werdenden Operanden irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig ist. Dieser Operand wird entweder aktuelles Objekt (VALUE-L2=*NO) oder er liegt auf dem Pfad zu dem Operanden, der aktuelles Objekt wird (VALUE-L2≠*NO).

VALUE-L2 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des zum aktuellen Objekt werdenden Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet, die den zum aktuellen Objekt werdenden Operanden unmittelbar oder mittelbar enthält. Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung des zum aktuellen Objekt werdenden Operanden irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig ist. Dieser Operand wird entweder aktuelles Objekt (VALUE-L3=*NO) oder er liegt auf dem Pfad zu dem Operanden, der aktuelles Objekt wird (VALUE-L3≠*NO).

VALUE-L3 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung des zum aktuellen Objekt werdenden Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet, die den zum aktuellen Objekt werdenden Operanden unmittelbar oder mittelbar enthält. Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

VALUE-L4 = analog VALUE-L1
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, in der der spezifizierte Operand aktuelles Objekt wird.

ORIGIN = *CURRENT
Der Operand gehört zu dem Kommando (bzw. zu der Anweisung), das zurzeit entweder selbst aktuelles Objekt ist oder einen Operanden oder Operandenwert enthält, der aktuelles Objekt ist.

ORIGIN = *COMMAND(...)
Der Operand gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Der Operand gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *VALUE(...)
Ein Operandenwert wird aktuelles Objekt. Der Operandenwert, der aktuelles Objekt wird, wird durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Gehört der zum aktuellen Objekt werdende Operandenwert zu einem Operanden, der außerhalb jeder Struktur steht, so liegt auf dem Pfad nur dieser Operand. Gehört der zum aktuellen Objekt werdende Operandenwert zu einem Operanden, der in einer Struktur steht, so liegen auf dem Pfad außerdem die übergeordneten Operanden und die zu diesen gehörenden struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig angegeben werden. Ein Operand, der zur Identifizierung des zum aktuellen Objekt werdenden Operandenwerts nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten OPERAND-Li+1=*NO betrachtet SDF-A den durch VALUE-Li bestimmten Operandenwert als den, der aktuelles Objekt wird. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu OPERAND-Li ein anderer Wert als *NO angegeben, so muss zu VALUE-Li ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Bestimmt den Operanden, zu dem der zum aktuellen Objekt werdende Operandenwert gehört (OPERAND-L2=*NO), oder einen Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L2≠*NO). *ABOVE-CURRENT bedeutet, dass ein zu OPERAND-L1 gehörender Wert aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1= *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING/ *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Bestimmt den Operandenwert, der aktuelles Objekt wird (OPERAND-L2=*NO), oder einen struktureinleitenden Operandenwert, der auf dem Pfad zu dem zum aktuellen Objekt werdenden Operandenwert liegt (OPERAND-L2≠*NO). *CURRENT bedeutet, dass VALUE-L1 aktuelles Objekt ist. Ist VALUE-L1 nicht aktuelles Objekt und vom

Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass VALUE-L1 aktuelles Objekt wird. Andernfalls ist der Name des Operanden anzugeben, zu dem der Operandenwert gehört, der aktuelles Objekt wird (OPERAND-L3=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L3≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig sein.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des zum aktuellen Objekt werdenden Operandenwerts irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser wird entweder aktuelles Objekt (OPERAND-L3=*NO) oder ist ein struktureinleitender Operandenwert, der auf dem Pfad zu dem Operandenwert liegt, der aktuelles Objekt wird (OPERAND-L3≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung des zum aktuellen Objekt werdenden Operandenwerts irrelevant ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der Operandenwert gehört, der aktuelles Objekt wird (OPERAND-L4=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L4≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig sein.

VALUE-L3 = analog VALUE-L2
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung des zum aktuellen Objekt werdenden Operandenwerts irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser wird entweder aktuelles Objekt (OPERAND-L4=*NO) oder ist ein struktureinleitender Operandenwert, der auf dem Pfad zu dem Operandenwert liegt, der aktuelles Objekt wird (OPERAND-L4≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L4 = analog VALUE-2
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L5 = analog VALUE-2
siehe VALUE-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, in der der spezifizierte Operandenwert aktuelles Objekt wird.

ORIGIN = *CURRENT
Der Operandenwert gehört zu dem Kommando (bzw. der Anweisung), das zurzeit entweder selbst aktuelles Objekt ist oder einen Operanden oder Operandenwert enthält, der aktuelles Objekt ist.

ORIGIN = *COMMAND(...)
Der Operandenwert gehört zu einem Kommando.

NAME=<structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Der Operandenwert gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

END
Programmlauf beenden

Mit der Anweisung END beenden Sie die Eingabe an SDF-A. Die Anweisung schließt auch implizit die zuletzt geöffnete Syntaxdatei.

END

Diese Anweisung hat keine Operanden.

MODIFY-CMD
Kommandodefinition ändern

Mit der Anweisung MODIFY-CMD ändern Sie in der geöffneten Syntaxdatei die Definition eines Kommandos. Dieses Kommando muss „aktuelles“ Objekt sein (siehe Seite 151). Anschließend ist der erste Operand dieses Kommandos „aktuelles“ Objekt.

Die Anweisung MODIFY-CMD gleicht weitgehend der Anweisung ADD-CMD. Default-Wert für alle Operanden von MODIFY-CMD ist *UNCHANGED, d.h. es werden nur die Teile der Kommandodefinition verändert, die explizit angegeben sind. Im geführten Dialog wird im Operandenfragebogen statt *UNCHANGED die aktuelle Operandenbesetzung angezeigt. Wenn Sie zu einem Operanden einen Wert ungleich *UNCHANGED angeben, so werden die alten Angaben in der Kommandodefinition durch die neuen ersetzt. Das gilt auch dann, wenn Listenangabe zulässig ist, d.h. die Werteliste wird nicht um den angegebenen Wert ergänzt, sondern durch ihn ersetzt. Für den STANDARD-NAME gibt es eine Sonderregelung. Wenn das Kommando in einer zugewiesenen Referenzdatei definiert ist (siehe OPEN-SYNTAX-FILE), bleiben die alten Angaben für STANDARD-NAME erhalten und der in MODIFY-CMD angegebene Name kommt additiv hinzu.

Mit Ausnahme des RESULT-INTERNAL-NAME müssen alle für das Kommando vergebenen Namen eindeutig in Bezug auf den gesamten Kommandovorrat sein.

Die Definitionen der zu dem Kommando gehörenden Operanden und Operandenwerte werden nicht mit der Anweisung MODIFY-CMD, sondern mit der Anweisung MODIFY-OPERAND bzw. MODIFY-VALUE geändert.

(Teil 1 von 4)

MODIFY-**CMD**

NAME = ***UNCH**ANGED / <structured-name 1..30>

,**RES**ULT-**INTERN**AL-**NAME** = ***UNCH**ANGED / ***SAME** / <alphanum-name 1..8>

,**STAND**ARD-**NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..30>

,**ALIAS-NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): <structured-name 1..30>

,**MIN**IMAL-**ABBREV**IATION = ***UNCH**ANGED / ***NO** / <structured-name 1..30>

,**HELP** = ***UNCH**ANGED / ***NO** / list-poss(2000): <name 1..1>(...)

<name 1..1>(...)

TEXT = ***UNCH**ANGED / <c-string 1..500 with-low>

,**DOM**AIN = ***UNCH**ANGED / ***NO** / list-poss(2000): <structured-name 1..30>

Fortsetzung

(Teil 2 von 4)

,**IMPL**EMENTOR = ***UNCH**ANGED / ***PROC**EDURE(...) **/ *TPR**(...) / ***APPLIC**ATION(...) / ***BY-TPR**(...)

***PROC**EDURE(...)

NAME = ***UNCH**ANGED / <c-string 1..280> / ***BY-IMON**(...)

***BY-IMON**(...)

LOGICAL-ID = ***UNCH**ANGED / <filename 1..30 without-cat-user-gen-vers>

,**INSTALLATION-UNIT** = ***UNCH**ANGED / <text 1..30 without-sep>

,**VERSION** = ***UNCH**ANGED / ***STD** / <product-version>

,**DEFAULT-PATH-NAME** = ***UNCH**ANGED / <filename 1..54>

,**ELEMENT** = ***UNCH**ANGED / ***NONE** / <composed-name 1..64>

,**CALL-TYPE** = ***UNCH**ANGED / ***CALL-PROC**EDURE / ***INCL**UDE-**PROC**EDURE / ***ENTER-PROC**EDURE

,**CALL-OPT**IONS = ***UNCH**ANGED / ***NONE** / <c-string 1..1800 with-low>

,**UNLOAD-PROG**RAM = ***UNCH**ANGED / ***NO** / ***YES**

***TPR**(...)

ENTRY = ***UNCH**ANGED / <name 1..8>

,**INTERF**ACE = ***UNCH**ANGED / ***SPL** / ***ASS** / ***ISL**(...)

***ISL**(...)

VERSION = ***UNCH**ANGED / <integer 1..2>

,**CMD-INTERFACE** = ***UNCH**ANGED / ***STRING**(...) / ***TRANSFER-AREA**(...) / ***NEW**(...)

***STRING**(...)

OUT-CMD-NAME = ***UNCH**ANGED / ***SAME** / <structured-name 1..30>

***TRANSFER-AREA**(...)

MAX-STRUC-OPERAND = ***UNCH**ANGED / ***STD** / <integer 1..3000>

,**CMD-VERSION** = ***UNCH**ANGED / ***NONE** / <integer 1..999>

***NEW**(...)

MAX-STRUC-OPERAND = ***UNCH**ANGED / ***STD** / <integer 1..3000>

,**LOG**GING = ***UNCH**ANGED / ***BY-SDF** / ***BY-IMPL**EMENTOR

,**INP**UT-**FORM** = ***UNCH**ANGED / ***INVAR**IANT / ***STAND**ARD / ***NONE**

,**SCI** = ***UNCH**ANGED / ***YES** / ***NO**

***APPLIC**ATION(...)

LOGGING = ***UNCH**ANGED / ***BY-SDF** / ***BY-IMPL**EMENTOR

***BY-TPR**(...)

TPR-CMD = ***UNCH**ANGED / <structured-name 1..30>

Fortsetzung ➠

(Teil 3 von 4)

,**REM**OVE**-POSSIBLE** = ***UNCH**ANGED / ***YES** / ***NO**

,**DIAL**OG**-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**DIAL**OG**-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**GUID**ED**-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**BATCH-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**BATCH-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**CMD-ALLOW**ED = ***UNCH**ANGED / ***NO**(...) / ***YES**(...)

***NO**(...)
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***YES**(...)
UNLOAD = ***UNCH**ANGED / ***YES** / ***NO**
PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

Fortsetzung

(Teil 4 von 4)

```
,NEXT-INPUT = *UNCHANGED / *CMD / *STMT / *DATA / *ANY
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) /
             list-poss(64): <structured-name 1..30>
  *EXCEPT(...)
    | EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
  *ADD(...)
    | ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
  *REMOVE(...)
    | REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *UNCHANGED / <structured-name 1..30>
(externer) Kommandoname, der bei der Kommandoeingabe anzugeben ist. Der Benutzer kann ihn im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Kommandoeingabe abkürzen.

RESULT-INTERNAL-NAME = *UNCHANGED / *SAME / <alphanum-name 1..8>
Dieser Operand ist lediglich für einige der durch System-Module implementierten Kommandos von Bedeutung.
Die Implementierung durch System-Module ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten. Der Operand RESULT-INTERNAL-NAME wird deshalb hier nicht beschrieben.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30>
zusätzlicher externer Kommandoname, der bei der Kommandoeingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME kann im Gegensatz zum ALIAS-NAME nicht gelöscht werden, solange das Kommando mit diesem Namen in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE) existiert. Wird der ursprüngliche, in der Dokumentation des Kommandos genannte externe Name zum Standardnamen erklärt, so ist sichergestellt, dass das Kommando ungeachtet aller Namensänderungen mit dem ursprünglichen Namen eingegeben werden kann. Die Angabe *NAME bewirkt, dass SDF-A als STANDARD-NAME den externen Kommandonamen nimmt, den Sie beim Operanden NAME angegeben haben.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
zusätzlicher externer Kommandoname, der bei der Kommandoeingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30>
zusätzlicher externer Kommandoname, der die kürzest mögliche Abkürzung für das Kommando festlegt. Eine kürzere Abkürzung wird dann nicht akzeptiert, selbst wenn sie in Bezug auf andere Kommandos eindeutig wäre.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar das richtige Kommando auswählt, es aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem Kommandonamen (NAME) entstehen.
3. Die ALIAS-NAMEs und STANDARD-NAMEs des Kommandos dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Kommandonamens sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung, aber keine Verlängerung vorgenommen werden.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für das Kommando gibt.

HELP = *UNCHANGED
keine Änderung bezüglich der Hilfetexte.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Hilfetext. *UNCHANGED ist nur zulässig, wenn für den Sprachschlüssel bereits ein Hilfetext definiert ist.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

DOMAIN = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
Bestimmt, ob und ggf. welchen Anwendungsbereichen das Kommando zugeordnet wird.

IMPLEMENTOR =
Art der Kommandoimplementierung

IMPLEMENTOR = *UNCHANGED
keine Änderung bezüglich der Kommandoimplementierung.

IMPLEMENTOR = *PROCEDURE(...)
Das Kommando ist durch eine Prozedur implementiert. Die Eingabe des Kommandos bewirkt den Aufruf der Prozedur.

NAME = *UNCHANGED / <c-string 1..280> / *BY-IMON(...)
Name der aufzurufenden Prozedur.

NAME = <c-string 1..280>
Name der Datei, in der die Prozedur steht. Wenn SDF-P geladen ist, kann auch der Name einer Listenvariablen, die die Prozedur enthält, angegeben werden. Die Angabe erfolgt dann in der Form '*VARIABLE(VARIABLE-NAME=varname)'.
Bibliothekselemente können mit '*LIBRARY-ELEMENT(LIBRARY=library, ELEMENT=element)' angegeben werden.
Bei Angabe von 'library(element)' wird der Wert von CALL-OPTIONS ignoriert. Diese Schreibweise sollte deshalb nicht mehr verwendet werden.

Der Benutzer ist selbst dafür verantwortlich, dass die Zeichenkette zur Angabe des Prozedurbehälters korrekt aufgebaut ist. Ein Fehler an dieser Stelle wird erst beim Aufruf des geänderten Kommandos sichtbar.

NAME = *BY-IMON(...)
Der Name der Prozedur bzw. der Bibliothek, die diese Prozedur als Bibliothekselement enthält, wird durch den Aufruf von IMON-GPN, dem Installation-Pathmanager bereitgestellt (siehe Benutzerhandbuch „IMON“ [13]).

LOGICAL-ID = *UNCHANGED / <filename 1..30 without-cat-user-gen-vers>
Logischer Name der kommandoimplementierenden Prozedur bzw. der Bibliothek, die diese Prozedur als Bibliothekselement enthält (z.B. SYSSPR).

INSTALLATION-UNIT = *UNCHANGED /<text 1..30 without-sep>
Name der Installation-Unit, z.B. SDF-A.

VERSION = *UNCHANGED / *STD / <product-version>
Version der Installation-Unit (vgl. „product-version“ auf Seite 15 bzw. Seite 186).
Bei Angabe von *STD wird die Version verwendet, die mit dem Kommando SELECT-PRODUCT-VERSION ausgewählt wurde. Falls dieses Kommando für die betreffende Installation-Unit noch nicht ausgeführt wurde, wird die höchste Version verwendet.

DEFAULT-PATH-NAME = *UNCHANGED / <filename 1..54>
Vollständiger Dateiname einer Prozedur, die aufgerufen wird, falls IMON-GPN nicht verfügbar ist oder wenn LOGICAL-ID, INSTALLATION-UNIT oder VERSION im System nicht bekannt sind. Ist die Prozedur in einem Bibliothekselement gespeichert, bezeichnet der hier angegebene Dateiname die Bibliothek, aus der die im Operanden ELEMENT angegebene Prozedur aufgerufen wird.
Bei anderen Fehlern wird das Kommando mit einer Fehlermeldung abgewiesen, d.h. die hier angegebene Prozedur wird nicht aufgerufen.

ELEMENT = *UNCHANGED / *NONE / <composed-name 1..64>
Gibt an, ob die Prozedur in einem Bibliothekselement gespeichert ist.

ELEMENT = <composed-name 1..64>
Name des Bibliothekselements, das die Prozedur enthält. Der Elementname wird an IMON-GPN übergeben.

CALL-TYPE = *UNCHANGED / *CALL-PROCEDURE / *INCLUDE-PROCEDURE / *ENTER-PROCEDURE
Legt fest, ob die Prozedur mit CALL-PROCEDURE, INCLUDE-PROCEDURE oder ENTER-PROCEDURE aufgerufen wird. Mit CALL- und ENTER-PROCEDURE können sowohl S- als auch Nicht-S-Prozeduren aufgerufen werden. Mit INCLUDE-PROCEDURE können nur S-Prozeduren aufgerufen werden (siehe Handbücher „SDF-P“ [12] und „Kommandos, Band 1-5“ [4]).

Beispiel:

Damit die Kommandoprozedur mit
```
/CALL-PROCEDURE NAME=*LIB-ELEM(LIBRARY=xxx,ELEMENT=yyy)
```
aufgerufen wird, muss das Kommando so definiert sein:

```
//ADD-CMD ...,IMPLEMENTOR=*PROC(NAME='*LIB-ELEM(LIBRARY=xxx,
          ELEMENT=yyy)',CALL-TYPE=*CALL-PROCEDURE...
```

CALL-OPTIONS = *UNCHANGED / *NONE / <c-string 1..1800 with-low>
Gibt eine Zeichenkette an, die weitere Operanden (z.B. LOGGING) für den Prozeduraufruf mit CALL-/INCLUDE- oder ENTER-PROCEDURE in der folgenden Form enthält:
`CALL-OPTIONS='operandx=wertx,operandy=werty, ...'`
Der Operand PROCEDURE-PARAMETERS der Kommandos CALL-/INCLUDE-/ENTER-PROCEDURE darf nicht in dieser Zeichenkette enthalten sein.

UNLOAD-PROGRAM = *UNCHANGED / *YES / *NO
Gibt an, ob ein Programm entladen werden soll, wenn das im Operanden NAME definierte Kommando in dem Programm über den CMD-Makro ausgeführt wird.
Die aufgerufene Prozedur selbst darf kein Kommando enthalten, das mit CMD-ALLOWED=*YES(UNLOAD=*YES) definiert ist, insbesondere darf in der Prozedur kein TU-Programm aufgerufen werden.

IMPLEMENTOR = *TPR(...)
Das Kommando ist durch System-Module implementiert. Diese Möglichkeit der Kommandoimplementierung ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten. Die Struktur *TPR wird deshalb hier nicht beschrieben.

IMPLEMENTOR = *APPLICATION(...)
Das Kommando wird durch eine $CONSOLE-Anwendung implementiert. Diese Möglichkeit ist der Systembetreuung vorbehalten. Die Voraussetzungen dafür sind im Handbuch „Einführung in die Systembetreuung“ [6] beschrieben. Die erforderlichen Kommandos CONNECT-CMD-SERVER und DISCONNECT-CMD-SERVER finden Sie im Handbuch „Kommandos, Band 1-5“ [4].

LOGGING = *UNCHANGED / *BY-SDF / *BY-IMPLEMENTOR
Der Operand ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten und wird deshalb hier nicht beschrieben.

IMPLEMENTOR = *BY-TPR(...)
Ein bereits existierendes Kommando dient als Vorlage für das neue Kommando.

TPR-CMD = *UNCHANGED / <structured-name 1..30>
Name eines mit IMPLEMENTOR=*TPR(...) definierten Kommandos, das in der Syntaxdatei-Hierarchie bekannt ist (Beispiel siehe Seite 141).

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Bestimmt, ob das Kommando gelöscht werden darf (siehe REMOVE, Seite 307). Wenn in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE, Seite 303) das Kommando mit REMOVE-POSSIBLE=*NO definiert ist, so lehnt SDF-A eine Änderung in *YES ab.

DIALOG-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Bestimmt, ob das Kommando im Dialogbetrieb zugelassen ist.

DIALOG-ALLOWED = *YES(...)
Das Kommando ist im Dialogbetrieb für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

DIALOG-ALLOWED = *NO(...)
Das Kommando ist im Dialogbetrieb für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Bestimmt, ob das Kommando im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

DIALOG-PROC-ALLOWED = *YES(...)
Das Kommando ist im Dialogbetrieb innerhalb einer Prozedur für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

DIALOG-PROC-ALLOWED = *NO(...)
Das Kommando ist im Dialogbetrieb innerhalb einer Prozedur für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

GUIDED-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Bestimmt, ob das Kommando im geführten Dialog zugelassen ist.

GUIDED-ALLOWED = *YES(...)
Das Kommando ist im geführten Dialog für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

GUIDED-ALLOWED = *NO(...)
Das Kommando ist im geführten Dialog für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

BATCH-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Bestimmt, ob das Kommando im Stapelbetrieb zugelassen ist.

BATCH-ALLOWED = *YES(...)
Das Kommando ist im Stapelbetrieb für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

BATCH-ALLOWED = *NO(...)
Das Kommando ist im Stapelbetrieb für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

BATCH-PROC-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Bestimmt, ob das Kommando im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

BATCH-PROC-ALLOWED = *YES(...)
Das Kommando ist im Stapelbetrieb innerhalb einer Prozedur für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

BATCH-PROC-ALLOWED = *NO(...)
Das Kommando ist im Stapelbetrieb innerhalb einer Prozedur für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist das Kommando gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Das Kommando ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

CMD-ALLOWED =
Bestimmt, ob das Kommando mit dem CMD-Makro aufgerufen werden kann.

CMD-ALLOWED = *UNCHANGED
keine Änderung bezüglich des Aufrufs mit CMD-Makro.

CMD-ALLOWED = *YES(...)
Das Kommando kann mit dem CMD-Makro aufgerufen werden. Der Aufruf mit dem CMD-Makro ist für alle Benutzeraufträge zugelassen, die mindestens eines der unter PRIVILEGE angegebenen Privilegien besitzen. Einschränkungen hinsichtlich der zugelassenen Betriebsart (DIALOG-ALLOWED, DIALOG-PROC-ALLOWED, BATCH-ALLOWED, BATCH-PROC-ALLOWED) gelten nicht, wenn das Kommando mit dem CMD-Makro aufgerufen wird.

UNLOAD = *NO / *YES
Bestimmt, ob das aufrufende Programm entladen wird. Bei Kommandos, die durch eine Kommandoprozedur implementiert sind, wird das aufrufende Programm entladen wenn für den Aufruf der Kommandoprozedur UNLOAD-PROGRAM=*YES (siehe Operand IMPLEMENTOR=*PROCEDURE(...), Seite 240) vereinbart wurde.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist der Aufruf des Kommandos zugelassen (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Der Aufruf des Kommandos ist für Benutzeraufträge mit genau den gleichen Privilegien zugelassen, die für das Kommando selbst definiert sind (siehe Operand PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Aufruf des Kommandos ist nur für Benutzeraufträge mit genau den Privilegien zugelassen, die Sie in dieser Liste angeben.

CMD-ALLOWED = *NO(...)
Der Aufruf des Kommandos mit dem CMD-Makro ist für alle Benutzeraufträge gesperrt, die lediglich die unter PRIVILEGE angegebenen Privilegien besitzen.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
Für Benutzeraufträge mit den hier angegebenen Privilegien ist der Aufruf des Kommandos gesperrt (mögliche Privilegien siehe Handbuch „SECOS“ [10]).

PRIVILEGE = *SAME
Der Aufruf des Kommandos ist für Benutzeraufträge mit genau den gleichen Privilegien gesperrt, für die auch das Kommando selbst gesperrt ist (siehe Operand EXCEPT-PRIVILEGE auf Seite 248).

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Aufruf des Kommandos ist für Benutzeraufträge mit den Privilegien gesperrt, die Sie in dieser Liste angeben. Hat ein Benutzerauftrag mindestens ein Privileg, das nicht in dieser Liste enthalten ist, dann darf er das Kommando ausführen.

NEXT-INPUT =
Bestimmt, was für eine Eingabe nach dem Kommando erwartet wird. Diese Angabe benötigt SDF zur Steuerung des geführten Dialogs.

NEXT-INPUT = *UNCHANGED
keine Änderung bezüglich der nachfolgenden Eingabe.

NEXT-INPUT = *CMD
Ein Kommando wird für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Kommando.

NEXT-INPUT = *STMT
Eine Anweisung wird für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Anweisung. Beispiel: Ein mittels Prozedur implementiertes Kommando startet ein Programm, das als Erstes eine Anweisung einliest.

NEXT-INPUT = *DATA
Daten werden für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Daten. Beispiel: Ein mittels Prozedur implementiertes Kommando startet ein Programm, das als Erstes Daten einliest.

NEXT-INPUT = *ANY
Die Art der nachfolgenden Eingabe ist nicht vorhersehbar.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) / list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Kommando zugeordnet werden.

PRIVILEGE = *ALL
Das Kommando erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Das Kommando erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Kommando zugeordnet werden.

PRIVILEGE = *ADD(...)
Das Kommando erhält zusätzlich zu den bereits zugeordneten Privilegien die bei *ADD(...) angegebenen Privilegien.

ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Kommando zusätzlich zugeordnet werden.

PRIVILEGE = *REMOVE(...)
Dem Kommando werden die bei *REMOVE(...) angegebenen Privilegien entzogen.

REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Kommando entzogen werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Kommando erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

MODIFY-CMD-ATTRIBUTES
Kommandoattribute ändern

Mit der Anweisung MODIFY-CMD-ATTRIBUTES ändern Sie für mehrere Kommandos gleichzeitig die sicherheitsrelevanten Attribute (Eingabemodi und Privilegien).

(Teil 1 von 2)

MODIFY-**CMD-ATT**RIBUTES

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

,**DIAL**OG-**ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**DIAL**OG-**PROC-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**GUID**ED-**ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**BATCH-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

***NO**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**BATCH-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

***YES**(...)

PRIVILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

Fortsetzung ➠

(Teil 2 von 2)

```
    *NO(...)
      |  PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
,CMD-ALLOWED = *UNCHANGED / *NO(...) / *YES(...)
    *NO(...)
      |  PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
    *YES(...)
      |  UNLOAD = *UNCHANGED / *YES / *NO
      |  ,PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) /
             list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
    *ADD(...)
      |  ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
    *REMOVE(...)
      |  REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Namen der Kommandos, deren Attribute zu ändern sind.

Die Beschreibung aller anderen Operanden der Anweisung MODIFY-CMD-ATTRIBUTES finden Sie bei der Anweisung MODIFY-CMD ab Seite 242.

MODIFY-DOMAIN
Anwendungsbereichsdefinition ändern

Mit der Anweisung MODIFY-DOMAIN ändern Sie in der geöffneten Syntaxdatei die Definition eines Anwendungsbereichs. Dieser Anwendungsbereich muss „aktuelles“ Objekt sein (siehe Seite 225).

Die Anweisung MODIFY-DOMAIN gleicht weitgehend der Anweisung ADD-DOMAIN, Default-Wert für alle Operanden von MODIFY-DOMAIN ist *UNCHANGED, d.h. es werden nur die Teile der Definition verändert, die explizit angegeben sind. Im geführten Dialog wird im Operandenfragebogen statt *UNCHANGED die aktuelle Operandenbesetzung angezeigt. Wenn Sie zu einem Operanden einen Wert ungleich *UNCHANGED angeben, so werden die alten Angaben in der Definition durch die neuen ersetzt. Das gilt auch dann, wenn Listenangabe zulässig ist, d.h. die Werteliste wird nicht um den angegebenen Wert ergänzt, sondern durch ihn ersetzt. Der für den Anwendungsbereich vergebene Name muss eindeutig in Bezug auf alle übrigen Anwendungsbereiche sein.

```
MODIFY-DOMAIN

 NAME = *UNCHANGED / <structured-name 1..30>
,HELP = *UNCHANGED / *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
        TEXT = *UNCHANGED / <c-string 1..500 with-low>
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
        EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *UNCHANGED / <structured-name 1..30>
Name des Anwendungsbereichs, der im geführten Dialog benutzt wird.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für den Anwendungsbereich gibt.

HELP = *UNCHANGED
Keine Änderung bezüglich der Hilfetexte.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Hilfetext. *UNCHANGED ist nur zulässig, wenn für den Sprachschlüssel bereits ein Hilfetext definiert ist.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) /
list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Anwendungsbereich zugeordnet werden.

PRIVILEGE = *ALL
Der Anwendungsbereich erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Der Anwendungsbereich erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Anwendungsbereich zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Anwendungsbereich erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

MODIFY-OPERAND
Operandendefinition ändern

Mit der Anweisung MODIFY-OPERAND ändern Sie in der geöffneten Syntaxdatei die Definition eines Operanden. Dieser Operand muss „aktuelles“ Objekt sein. Anschließend ist der erste Operandenwert dieses Operanden „aktuelles“ Objekt (siehe Seite 151).

Die Anweisung MODIFY-OPERAND gleicht weitgehend der Anweisung ADD-OPERAND. Default-Wert für alle Operanden von MODIFY-OPERAND ist *UNCHANGED, d.h. es werden nur die Teile der Operandendefinition verändert, die explizit angegeben sind. Im geführten Dialog wird im Operandenfragebogen statt *UNCHANGED die aktuelle Operandenbesetzung angezeigt. Wenn Sie einen Wert ungleich *UNCHANGED angeben, so werden die alten Angaben in der Operandendefinition durch die neuen ersetzt. Das gilt auch dann, wenn Listenangabe zulässig ist, d.h. die Werteliste wird nicht um den angegebenen Wert ergänzt, sondern durch ihn ersetzt.

Für den STANDARD-NAME gibt es eine Sonderregelung. Wenn der Operand in einer zugewiesenen Referenzdatei definiert ist (siehe OPEN-SYNTAX-FILE), bleiben die alten Angaben für STANDARD-NAME erhalten und der in MODIFY-OPERAND angegebene Name kommt additiv hinzu. Alle für den Operanden vergebenen Namen müssen in Bezug auf die übrigen Operanden derselben Ebene (bzw. derselben Struktur) eindeutig sein. Die Definitionen der zu dem Operanden gehörenden Operandenwerte werden nicht mit der Anweisung MODIFY-OPERAND, sondern mit der Anweisung MODIFY-VALUE geändert.

(Teil 1 von 2)

MODIFY-**OPER**AND

NAME = ***UNCH**ANGED / <structured-name 1..20>

,**INTERN**AL-**NAME** = ***UNCH**ANGED / ***STD** / <alphanum-name 1..8>

,**STAND**ARD-**NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..20>

,**ALIAS-NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): <structured-name 1..20>

,**MIN**IMAL-**ABBREV**IATION = ***UNCH**ANGED / ***NO** / <structured-name 1..30>

,**HELP** = ***UNCH**ANGED / ***NO** / list-poss(2000): <name 1..1>(...)

<name 1..1>(...)

TEXT = ***UNCH**ANGED / <c-string 1..500 with-low>

,**DEF**AULT = ***UNCH**ANGED / ***NONE** / ***JV**(...) / ***VAR**IABLE(...) / <c-string 1..1800 with-low>(...)

***JV**(...)

JV-NAME = ***UNCH**ANGED / <filename 1..54 without-gen-vers>

,**ALTER**NATE-**DEF**AULT = ***UNCH**ANGED / ***NONE** / <c-string 1..1800>(...)

<c-string 1..1800>(...)

ANALYSE-**DEF**AULT = ***UNCH**ANGED / ***YES** / ***NO**

***VAR**IABLE(...)

VARIABLE-**NAME** = ***UNCH**ANGED / <composed-name 1..255>

,**ALTER**NATE-**DEF**AULT = ***UNCH**ANGED / ***NONE** / <c-string 1..1800>(...)

<c-string 1..1800>(...)

ANALYSE-**DEF**AULT = ***UNCH**ANGED / ***YES** / ***NO**

<c-string 1..1800 with-low>(...)

ANALYSE-**DEF**AULT = ***UNCH**ANGED / ***YES** / ***NO**

,**SECRET-PROMPT** = ***UNCH**ANGED / ***YES** / ***NO**

,**STRUCT**URE-**IMPLICIT** = ***UNCH**ANGED / ***YES** / ***NO**

,**REM**OVE-**POSSIBLE** = ***UNCH**ANGED / ***YES** / ***NO**

,**DIAL**OG-**ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**DIAL**OG-**PROC-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**GUID**ED-**ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**BATCH-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**BATCH-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

Fortsetzung

(Teil 2 von 2)

```
,LIST-POSSIBLE = *UNCHANGED / *NO / *YES(...)
    *YES(...)
        LIMIT = *UNCHANGED / *STD / <integer 1..3000>
        ,FORM = *UNCHANGED / *NORMAL / *OR
,LINES-IN-FORM = *UNCHANGED / <integer 1..15>
,PRESENCE = *UNCHANGED / *NORMAL / *EXTERNAL-ONLY / *INTERNAL-ONLY
,RESULT-OPERAND-LEVEL = *UNCHANGED / <integer 1..5>
,RESULT-OPERAND-NAME = *UNCHANGED / *SAME / <structured-name 1..20> / *POSITION(...) /
                       *LABEL / *COMMAND-NAME
    *POSITION(...)
        POSITION = *UNCHANGED / <integer 1..3000>
,CONCATENATION-POS = *UNCHANGED / *NO / <integer 1..20>
,VALUE-OVERLAPPING = *UNCHANGED / *YES / *NO
,OVERWRITE-POSSIBLE = *UNCHANGED / *NO / *YES
,PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
        EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *UNCHANGED / <structured-name 1..20>
(externer) Operandenname, der bei der Kommando- bzw. Anweisungseingabe anzugeben ist (siehe aber Operand PRESENCE=*INTERNAL-ONLY). Der Benutzer kann ihn im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Eingabe abkürzen.

INTERNAL-NAME = *UNCHANGED / *STD / <alphanum-name 1..8>
interner Operandenname. SDF identifiziert einen Operanden, der in mehreren Syntaxdateien mit unterschiedlichem externen Namen definiert ist, mithilfe des internen Operandennamens als denselben Operanden. Standardmäßig nimmt SDF-A als internen Operandennamen die ersten acht Zeichen (ohne Bindestrich) des externen Namens, den Sie beim Operanden NAME angegeben haben.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..20>
zusätzlicher externer Operandenname, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME darf im Gegensatz zum ALIAS-NAME nicht gelöscht werden, solange der Operand mit diesem Namen in einer der zugewiesenen Syntaxdateien (siehe OPEN-SYNTAX-FILE) existiert. Wird der ursprüngliche, in der Kommando- bzw. Programmdokumentation genannte externe Name zum Standardnamen erklärt, so ist sichergestellt, dass der

Operand ungeachtet aller Namensänderungen mit dem ursprünglichen Namen eingegeben werden kann. Die Angabe *NAME bewirkt, dass SDF-A als STANDARD-NAME den externen Operandennamen nimmt, den Sie beim Operanden NAME angegeben haben.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..20>
zusätzlicher externer Operandenname, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30>
zusätzlicher externer Operandenname, der die kürzest mögliche Abkürzung für den Operanden festlegt. Eine kürzere Abkürzung wird dann nicht akzeptiert, selbst wenn sie in Bezug auf andere Operanden eindeutig wäre.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar den richtigen Operanden auswählt, ihn aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem Operandennamen (NAME) entstehen.
3. Die ALIAS-NAMEs und STANDARD-NAMEs des Operanden dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Operandennamens sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (aber keine Verlängerung) vorgenommen werden.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für das Kommando gibt.

HELP = *UNCHANGED
keine Änderung bezüglich der Hilfetexte.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Hilfetext. *UNCHANGED ist nur zulässig, wenn für den Sprachschlüssel bereits ein Hilfetext definiert ist.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

DEFAULT =
Bestimmt, ob es für den Operanden einen Default-Wert gibt.

DEFAULT = *UNCHANGED
Keine Änderung bezüglich des Default-Werts.

DEFAULT = *NONE
Es gibt keinen Default-Wert. Der Operand ist ein Pflichtoperand.

DEFAULT = *JV(...)
Der Operand ist wahlfrei. Sein Default-Wert ist in der Job-Variablen gespeichert, deren Name angegeben wird. Wird eine Job-Variable als Default-Wert verwendet, wird der Default-Wert von SDF immer zum Zeitpunkt der Ausführung analysiert. Ist der Zugriff auf die Job-Variable nicht möglich, so wird der mit ALTERNATE-DEFAULT definierte alternative Default-Wert verwendet. Gibt es keinen alternativen Default-Wert, ist der Operand als Pflichtoperand zu betrachten (entspricht DEFAULT=*NONE). DEFAULT=*JV(...) darf deshalb nicht zusammen mit PRESENCE=*INTERNAL-ONLY angegeben werden.

JV-NAME = *UNCHANGED
Der Name der Job-Variablen bleibt unverändert.

JV-NAME = <filename 1..54 without-gen-vers>
Name der Job-Variablen

ALTERNATE-DEFAULT = *UNCHANGED / *NONE / <c-string 1..1800 with-low>(...)
Alternativer Default-Wert, der verwendet wird, wenn beim Zugriff auf die Jobvariable Fehler auftreten.

ALTERNATE-DEFAULT = *NONE
Es gibt keinen alternativen Default-Wert.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternativer Default-Wert. Für die Angabe dieses alternativen Default-Werts gelten die gleichen Regeln wie für die Definition eines Operanden. Er kann beispielsweise auch eine in Klammern eingeschlossene Liste sein. Der alternative Default-Wert muss durch eine zu dem Operanden gehörende Operandenwertdefinition (siehe ADD-VALUE) abgedeckt sein. Deckt ein mit STAR=*MANDATORY definiertes Schlüsselwort diesen Default-Wert ab, so muss der Default-Wert mit einem Stern eingegeben werden.

ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
Bestimmt, ob SDF-A den angegebenen Wert bereits bei Abschluss der Kommando- bzw. Anweisungsdefinition syntaktisch analysiert. Das beschleunigt die Kommando- bzw. Anweisungsanalyse zur Laufzeit, setzt aber voraus, dass der Wert weder eine Struktur einleitet noch aus einer Liste besteht.

DEFAULT = *VARIABLE(...)
Der Operand ist wahlfrei. Sein Default-Wert ist in der S-Variablen (siehe Benutzerhandbuch „SDF-P“ [12]) gespeichert, deren Name angegeben wird.
Wird eine S-Variable als Default-Wert verwendet, dann wird der Default-Wert von SDF immer zum Zeitpunkt der Ausführung analysiert. Ist der Zugriff auf die S-Variable nicht möglich, so wird der mit ALTERNATE-DEFAULT definierte alternative Default-Wert verwendet. Gibt es keinen alternativen Default-Wert, ist der Operand als Pflichtoperand zu betrachten (entspricht DEFAULT=*NONE). DEFAULT=*VARIABLE(...) darf deshalb nicht zusammen mit PRESENCE=*INTERNAL-ONLY angegeben werden.

VARIABLE-NAME = *UNCHANGED
Der Name der S-Variablen bleibt unverändert.

VARIABLE-NAME = <composed-name 1..255>
Name der S-Variablen.

ALTERNATE-DEFAULT = *UNCHANGED / *NONE / <c-string 1..1800 with-low>(...)
Alternativer Default-Wert, der verwendet wird, wenn beim Zugriff auf die S-Variable Fehler auftreten.

ALTERNATE-DEFAULT = *NONE
Es gibt keinen alternativen Default-Wert.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternativer Default-Wert. Für die Angabe dieses alternativen Default-Werts gelten die gleichen Regeln wie für die Eingabe eines Operanden. Er kann beispielsweise auch eine in Klammern eingeschlossene Liste sein. Der alternative Default-Wert muss durch eine zu dem Operanden gehörende Operandenwertdefinition (siehe ADD-VALUE) abgedeckt sein. Deckt ein mit STAR=*MANDATORY definiertes Schlüsselwort diesen Default-Wert ab, so muss der Default-Wert mit einem Stern eingegeben werden.

ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
Bestimmt, ob SDF-A den angegebenen Wert bereits bei Abschluss der Kommando- bzw. Anweisungsdefinition syntaktisch analysiert. Das beschleunigt die Kommando- bzw. Anweisungsanalyse zur Laufzeit, setzt aber voraus, dass der Wert weder eine Struktur einleitet noch aus einer Liste besteht.

DEFAULT = <c-string 1..1800 with-low>(...)
Der Operand ist wahlfrei und hat den angegebenen Default-Wert. Für die Angabe dieses Default-Werts gelten die gleichen Regeln wie für die Eingabe eines Operanden. Er kann beispielsweise auch eine in Klammern eingeschlossene Liste sein. Der Default-Wert muss durch eine zu dem Operanden gehörende Operandenwertdefinition (siehe ADD-VALUE) abgedeckt sein. Deckt ein mit STAR=*MANDATORY definiertes Schlüsselwort den Default-Wert ab, so muss auch der Default-Wert mit einem Stern angegeben werden.

ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
Bestimmt, ob SDF-A den angegebene Standartwert bereits bei Abschluss der Kommando- bzw. Anweisungsdefinition syntaktisch analysiert.

Das beschleunigt die Kommando- bzw. Anweisungsanalyse zur Laufzeit, setzt aber voraus, dass der Default-Wert weder eine Struktur einleitet noch aus einer Liste besteht.

SECRET-PROMPT = *UNCHANGED / *NO / *YES
Bestimmt, ob der Operand als geheimer Operand behandelt wird. Das Eingabefeld für den Wert eines geheimen Operanden wird dunkelgesteuert und seine Protokollierung wird unterdrückt (siehe auch ADD-VALUE...,OUTPUT=*SECRET-PROMPT und ADD-VALUE...,SECRET-PROMPT=*SAME/*NO).

STRUCTURE-IMPLICIT = *UNCHANGED / *NO / *YES
ist nur relevant für Operanden, die in einer Struktur stehen, und bestimmt, ob bei der Kommando- bzw. Anweisungseingabe durch globale Angabe des Operandennamens implizit die Struktur mit ausgewählt wird, die den Operanden enthält (Einzelheiten siehe ADD-OPERAND, Seite 157).

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operand gelöscht werden darf (siehe REMOVE). Wenn der Operand zu einem Kommando gehört und in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE) mit REMOVE-POSSIBLE=*NO definiert ist, so lehnt SDF-A eine Änderung in *YES ab.

DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operand im Dialogbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Dialogbetrieb zugelassen ist.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operand im Dialogbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operand im geführten Dialog zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im geführten Dialog zugelassen ist.
Zur Durchsetzung sicherheitsrelevanter Aspekte ist GUIDED-ALLOWED=*NO nicht geeignet, da so definierte Operanden im Prozedurfehlerdialog sowie bei /SHOW-CMD bzw. //SHOW-STMT mit FORM=*UNGUIDED angezeigt werden.

BATCH-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operand im Stapelbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Stapelbetrieb zugelassen ist.

BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operand im Stapelbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass das Kommando bzw. die Anweisung und ggf. der struktureinleitende Operandenwert im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

LIST-POSSIBLE =
Bestimmt, ob an der Operandenposition eine Liste zugelassen ist. Für welche Operandenwerte diese Liste zulässig ist, wird mit der Anweisung ADD-VALUE festgelegt.

LIST-POSSIBLE = *UNCHANGED
Keine Änderung bezüglich der Zulässigkeit einer Liste.

LIST-POSSIBLE = *NO
Es ist keine Liste zugelassen.

LIST-POSSIBLE = *YES(...)
Es ist eine Liste zugelassen.

LIMIT = *UNCHANGED / *STD / <integer 1..3000>
Bestimmt die maximale Anzahl der Listenelemente. Standardmäßig setzt SDF-A den Wert 2000 ein (siehe auch Seite 383).

FORM = *UNCHANGED / *NORMAL / OR
Bestimmt, ob die Listenelemente einzeln adressiert (NORMAL) oder mit logischem OR zu einem einzigen Wert verknüpft an die Implementierung übergeben werden (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371). Letzteres ist nur sinnvoll bei Listenelementen vom Datentyp KEYWORD, für die sedezimale Übergabewerte definiert sind (siehe ADD-VALUE...,VALUE=<c-string>(...,OUTPUT=<x-string>...). Diese Angabe ist nur relevant, wenn der definierte Operand zu einer Anweisung oder zu einem mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierten Kommando (siehe ADD-CMD) gehört. Die hier gemachte Angabe muss konsistent mit dem in der Implementierung definierten Übergabebereich sein.

LINES-IN-FORM = *UNCHANGED / <integer 1..15>
Bestimmt die Anzahl der Eingabezeilen im Fragebogen des geführten Dialogs.

PRESENCE =
Bestimmt, ob der Operand unterdrückt wird.

PRESENCE = *UNCHANGED
Keine Änderung bezüglich der Unterdrückung des Operanden.

PRESENCE = *NORMAL
Der Operand wird nicht unterdrückt.

PRESENCE = *EXTERNAL-ONLY
Die Übergabe des Operanden an die Implementierung wird unterdrückt (z.B. ein nicht mehr benötigter Operand, der aus Kompatibilitätsgründen an der Benutzeroberfläche erhalten bleiben muss, oder ein Operand, der lediglich dazu dient, weitere Operanden in einer Struktur zu gruppieren).

PRESENCE = *INTERNAL-ONLY
Der Operand wird an der Benutzeroberfläche unterdrückt. In Verbindung mit der dann obligatorischen Definition eines Default-Werts (siehe Operand DEFAULT) lässt sich so einem implementierten Parameter ein fester Wert zuordnen, ohne dass deshalb im Kommando- bzw. Anweisungsformat ein Operand für den Benutzer sichtbar ist. Hängt an dem Operanden eine Struktur, so werden alle in ihr enthaltenen Unteroperanden in die höhere Ebene integriert. PRESENCE=*INTERNAL-ONLY darf nicht zusammen mit DEFAULT=*JV(...) oder DEFAULT=*VARIABLE(...) angegeben werden.

RESULT-OPERAND-LEVEL = *UNCHANGED / <integer 1..5>
Bestimmt die Strukturebene, auf der der Operand an die Implementierung übergeben wird. Bei einem Operanden, der in keiner Struktur steht, muss dieser Wert eins sein. Für einen Operanden, der in einer Struktur steht, gilt: Der RESULT-OPERAND-LEVEL ist gleich oder niedriger als die Strukturebene, auf der der Operand im Eingabeformat des Kommandos bzw. der Anweisung steht. Er ist kleiner, gleich oder um eins größer als der RESULT-OPERAND-LEVEL des Operanden, zu dem der struktureinleitende Operandenwert gehört. Bei Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW bzw. *TRANSFER-AREA,...) definierten Kommandos siehe auch //ADD-VALUE, Operand STRUCTURE=*YES(...,FORM=...).

RESULT-OPERAND-NAME =
Bestimmt, wie die Implementierung den Operanden in dem Übergabebereich oder String identifiziert, den SDF an sie übergibt. Anmerkung: Einen Übergabebereich (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371) benutzt SDF bei Anweisungen und bei mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierten Kommandos (siehe ADD-CMD). Einen String übergibt SDF bei Kommandos, die mit IMPLEMENTOR=*PROCEDURE(...) oder IMPLEMENTOR=*TPR (...,CMD-INTERFACE=*STRING,...) definiert sind (siehe ADD-CMD).

RESULT-OPERAND-NAME = *UNCHANGED
keine Änderung bezüglich des RESULT-OPERAND-NAME.

RESULT-OPERAND-NAME = *SAME
Ist nur zulässig, wenn die Operandenübergabe mittels String erfolgt.
Der Operand hat in dem zu übergebenden String den gleichen Namen, den Sie ihm mit NAME= gegeben haben.

RESULT-OPERAND-NAME = <structured-name 1..20>
Ist nur zulässig, wenn die Operandenübergabe mittels String erfolgt.
Der Operand hat in dem zu übergebenden String den angegebenen Namen.

RESULT-OPERAND-NAME = *POSITION(...)
Der Operand hat in dem Übergabebereich (vgl. Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371) oder in dem zu übergebenden String eine bestimmte Position. Ein Name ist ihm nicht zugeordnet.

POSITION = *UNCHANGED / <integer 1..3000>
Position. Bei Operanden, die zu einer Struktur gehören, ist die Positionsangabe strukturrelativ. Das bedeutet, dass der erste Operand in der Struktur die Position 1 hat, wenn für den aktuellen Operanden ein höheres RESULT-OPERAND-LEVEL definiert wurde, als der ihm übergeordnete Operand hat.

RESULT-OPERAND-NAME = *LABEL
*Ist nur zulässig für Operanden in mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE= *STRING,...) definierten Kommandos (siehe ADD-CMD).*
Der Operandenwert ist für die Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert und wird deshalb hier nicht beschrieben.

RESULT-OPERAND-NAME = *COMMAND-NAME
*Ist nur zulässig für Operanden in mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE= *STRING,...) definierten Kommandos (siehe ADD-CMD).*
Der Operandenwert ist für die Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert und wird deshalb hier nicht beschrieben.

CONCATENATION-POS = *UNCHANGED / *NO / <integer 1..20>
Bestimmt, ob und ggf. wie der Operand mit anderen Eingabeoperanden auf einen einzigen Operanden in dem an die Implementierung zu übergebenden String abgebildet wird. Die Eingabeoperanden werden miteinander verkettet. Welche Position sie bei der Verkettung erhalten ist hier anzugeben. Voraussetzung ist, dass die Übergabe an die Implementierung in Stringform erfolgt (Kommandos, die definiert sind mit IMPLEMENTOR= *PROCEDURE(...) oder IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...), siehe ADD-CMD). Alle miteinander zu verkettenden Eingabeoperanden müssen denselben RESULT-OPERAND-NAME haben.

VALUE-OVERLAPPING = *UNCHANGED / *YES
Bestimmt, ob eine Überlappung von Datentypen in der Definition der Operandenwerte zugelassen werden soll. Bei der Kommando- bzw. Anweisungseingabe überprüft SDF den eingegebenen Wert anhand der Datentypdefinitionen in der Reihenfolge, wie diese für den Operanden angegeben sind. Ist der Datentyp KEYWORD(-NUMBER), so überprüft SDF, ob der eingegebene Wert eindeutig ist hinsichtlich weiterer Definitionen vom Typ KEYWORD(-NUMBER). Zusätzlich überprüft SDF die definierten Eigenschaften (z.B. Länge, Wertebereich) des eingegebenen Wertes. Treffen diese Eigenschaften nicht zu, wird mit dem nächsten definierten Datentyp die Überprüfung fortgesetzt. Die Überlappung von Datentypen wird ab der SDF-Version 1.3 unterstützt. Wird eine Syntaxdatei mit Überlappung von Datentypen in einer älteren SDF-Version benützt, führt dies zu Fehlern. Sich ausschließende Datentypen siehe Seite 635. Mit der Anweisung MODIFY-OPERAND kann

eine zugelassene Überlappung von Datentypen nicht rückgängig gemacht werden (*NO nicht möglich). Eine Änderung kann nur dadurch erreicht werden, indem man den Operanden löscht und ihn und seine Werte anschließend neu definiert.

OVERWRITE-POSSIBLE = *UNCHANGED / *NO / *YES
*Ist nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(..., CMD-INTERFACE= *NEW/ *TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*
Bestimmt, ob der Default-Wert des Operanden durch einen von der Implementierung dynamisch erzeugten Wert überschrieben wird (siehe Operand DEFAULT in den Makros CMDCST, CMDRST und CMDTST). Der vom Programm erzeugte Wert muss gültiger Operandenwert sein. Im geführten Dialog zeigt SDF den von der Implementierung erzeugten Wert im Fragebogen.

PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) /
list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Operanden zugeordnet werden.

PRIVILEGE = *SAME
Der Operand erhält die gleichen Privilegien, die auch für das zugehörige Kommando oder die Anweisung festgelegt sind. Gehört der Operand zu einer Struktur, erhält der Operand die gleichen Privilegien wie der Operandenwert, der diese Struktur einleitet.

PRIVILEGE = *EXCEPT(...)
Der Operand erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Operanden zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Operand erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

MODIFY-PROGRAM
Programmdefinition ändern

Mit der Anweisung MODIFY-PROGRAM ändern Sie in der geöffneten Syntaxdatei die Definition eines Programms. Dieses Programm muss „aktuelles“ Objekt sein (siehe Seite 151). Der für das Programm vergebene Name muss eindeutig in Bezug auf alle in der Syntaxdatei definierten Programme sein.

Die Anweisung MODIFY-PROGRAM gleicht weitgehend der Anweisung ADD-PROGRAM.

MODIFY-**PROG**RAM

NAME = ***UNCH**ANGED / <structured-name 1..30>

,**PRIV**ILEGE = ***UNCH**ANGED / ***ALL** / ***EXC**EPT(...) / list-poss(64): <structured-name 1..30>

***EXC**EPT(...)

EXCEPT-**PRIV**ILEGE = list-poss(64): <structured-name 1..30>

,**COM**MENT-**L**INE = ***UNCH**ANGED / ***NONE** / ***STD** / <c-string 1..50 with-low>

NAME = *UNCHANGED / <structured-name 1..30>
(externer) Programmname, der im geführten Dialog angezeigt wird. Dieser Name ist frei wählbar (muss nicht mit dem Modul- oder Phasennamen übereinstimmen). Im geführten Dialog wird im Operandenfragebogen statt *UNCHANGED der aktuelle Programmname angezeigt.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Programm zugeordnet werden.

PRIVILEGE = *ALL
Das Programm erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Das Programm erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Programm zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Das Programm erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

COMMENT-LINE =
Gibt an, welche Programm-Kommentarzeile im geführten Dialog angezeigt werden soll. Die Programm-Kommentarzeile erscheint ganz oben in den Fragebögen des geführten Dialogs.

COMMENT-LINE = *UNCHANGED
Keine Änderung bezüglich der Programm-Kommentarzeile.

COMMENT-LINE = *NONE
Keine Programm-Kommentarzeile wird angezeigt.

COMMENT-LINE = *STD
In der Programm-Kommentarzeile werden die Programmversion und das Erzeugungsdatum des Programmes angezeigt. Bindemodule (Object Module, Typ-R-Elemente) haben keine interne Version. Deshalb wird an Stelle des Erzeugungsdatums das Ausführungsdatum angezeigt.

COMMENT-LINE = <c-string 1..50 with-low>
Zeichenkette, die als Programm-Kommentarzeile ausgegeben wird.

MODIFY-STMT
Anweisungsdefinition ändern

Mit der Anweisung MODIFY-STMT ändern Sie in der geöffneten Syntaxdatei die Definition einer Anweisung. Diese Anweisung muss „aktuelles“ Objekt sein. Anschließend ist der erste Operand dieser Anweisung „aktuelles“ Objekt (siehe Seite 151). Die Anweisung MODIFY-STMT gleicht weitgehend der Anweisung ADD-STMT.

Default-Wert für alle Operanden von MODIFY-STMT ist *UNCHANGED, d.h. es werden nur die Teile der Anweisungsdefinition verändert, die explizit angegeben sind.
Im geführten Dialog wird im Operandenfragebogen statt *UNCHANGED die aktuelle Operandenbesetzung angezeigt. Wenn Sie zu einem Operanden einen Wert ungleich *UNCHANGED angeben, so werden die alten Angaben in der Anweisungsdefinition durch die neuen ersetzt. Das gilt auch dann, wenn Listenangabe zulässig ist, d.h. die Werteliste wird nicht um den angegebenen Wert ergänzt, sondern durch ihn ersetzt.
Für den STANDARD-NAME gibt es eine Sonderregelung. Wenn das Kommando in einer zugewiesenen Referenzdatei definiert ist (siehe OPEN-SYNTAX-FILE), bleiben die alten Angaben für STANDARD-NAME erhalten und der in MODIFY-STMT angegebene Name kommt additiv hinzu. Alle für die Anweisung vergebenen Namen müssen eindeutig in Bezug auf die Menge der zu dem Programm gehörenden Anweisungen sein.

Die Definitionen der zu der Anweisung gehörenden Operanden und Operandenwerte werden nicht mit der Anweisung MODIFY-STMT, sondern mit der Anweisung MODIFY-OPERAND bzw. MODIFY-VALUE geändert.

MODIFY-**STMT**

NAME = ***UNCH**ANGED / <structured-name 1..30>

,**PROG**RAM = ***UNCH**ANGED / <structured-name 1..30>

,**STAND**ARD-**NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..30>

,**ALIAS-NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): <structured-name 1..30>

,**MIN**IMAL-**ABBREV**IATION = ***UNCH**ANGED / ***NO** / <structured-name 1..30>

,**HELP** = ***UNCH**ANGED / ***NO** / list-poss(2000): <name 1..1>(...)

<name 1..1>(...)

TEXT = ***UNCH**ANGED / <c-string 1..500 with-low>

,**MAX-STRUC-OPER**AND = ***UNCH**ANGED / ***STD** / <integer 1..3000>

,**REM**OVE-**POSSIBLE** = ***UNCH**ANGED / ***YES** / ***NO**

,**DIAL**OG-**ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**DIAL**OG-**PROC-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**GUID**ED-**ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**BATCH-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**BATCH-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**NEXT-IN**PUT = ***UNCH**ANGED / ***STMT** / ***DATA** / ***ANY**

,**PRIV**ILEGE = ***UNCH**ANGED / ***ALL** / ***EXC**EPT(...) / ***ADD**(...) / ***REMOVE**(...) /
list-poss(64): <structured-name 1..30>

***EXC**EPT(...)

EXCEPT-**PRIV**ILEGE = list-poss(64): <structured-name 1..30>

***ADD**(...)

ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>

***REMOVE**(...)

REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>

,**STMT-VERS**ION = ***UNCH**ANGED / ***NONE** / <integer 1..999>

NAME = *UNCHANGED / <structured-name 1..30>

(externer) Anweisungsname, der bei der Anweisungseingabe anzugeben ist. Der Benutzer kann ihn im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Anweisungseingabe abkürzen.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30>
zusätzlicher externer Anweisungsname, der bei der Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME darf im Gegensatz zum ALIAS-NAME nicht gelöscht werden. Er ist der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorbehalten und wird deshalb hier nicht beschrieben.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
zusätzlicher externer Anweisungsname, der bei der Anweisungseingabe alternativ benutzt werden kann. Er ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30>
zusätzlicher externer Anweisungsname, der die kürzest mögliche Abkürzung für die Anweisung festlegt. Eine kürzere Abkürzung wird dann nicht akzeptiert, selbst wenn sie in Bezug auf andere Anweisungen eindeutig wäre.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar die richtige Anweisung auswählt, sie aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem Anweisungsnamen (NAME) entstehen.
3. Die ALIAS-NAMEs und STANDARD-NAMEs der Anweisung dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Anweisungsnamens sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (aber keine Verlängerung) vorgenommen werden.

HELP =
Bestimmt, ob und ggf. welche Hilfetexte es für die Anweisung gibt.

HELP = *UNCHANGED
Keine Änderung bezüglich der Hilfetexte.

HELP = *NO
Es gibt keine Hilfetexte.

HELP = list-poss(2000): <name 1..1>(...)
Es gibt Hilfetexte in den angegebenen Sprachen (E=Englisch, D=Deutsch). SDF benutzt bevorzugt die Sprache, die für die Meldungsausgabe festgelegt ist.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Hilfetext. UNCHANGED ist nur zulässig, wenn für den Sprachschlüssel bereits ein Hilfetext definiert ist.
Der Hilfetext kann die spezielle Zeichenfolge „\n“ für den Zeilenumbruch enthalten.

MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
Anzahl der in der strukturierten Übergabe auf oberster Ebene zu reservierenden Operandenpositionen. Bei der Angabe STD wird das Operanden-Array so groß wie erforderlich angelegt. Für geplante künftige Erweiterungen lässt es sich aber auch größer anlegen.

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Bestimmt, ob die Anweisung gelöscht werden darf (vgl. REMOVE, Seite 307). Dieser Operand kann für die Standardanweisungen nicht geändert werden.

DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob die Anweisung im Dialogbetrieb zugelassen ist. Dieser Operand kann für die Standardanweisungen nicht geändert werden.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob die Anweisung im Dialogbetrieb innerhalb einer Prozedur zugelassen ist. Dieser Operand kann für die Standardanweisungen nicht geändert werden.

GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob die Anweisung im geführten Dialog zugelassen ist. Dieser Operand kann für die Standardanweisungen nicht geändert werden.

BATCH-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob die Anweisung im Stapelbetrieb zugelassen ist. Dieser Operand kann für die Standardanweisungen nicht geändert werden.

BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob die Anweisung im Stapelbetrieb innerhalb einer Prozedur zugelassen ist. Dieser Operand kann für die Standardanweisungen nicht geändert werden.

NEXT-INPUT =
Bestimmt, was für eine Eingabe nach der Anweisung erwartet wird. Diese Angabe benötigt SDF zur Steuerung des geführten Dialogs.

NEXT-INPUT = *UNCHANGED
Keine Änderung bezüglich der nachfolgenden Eingabe.

NEXT-INPUT = *STMT
Eine Anweisung wird für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Anweisung.

NEXT-INPUT = *DATA
Daten werden für die nachfolgende Eingabe erwartet. SDF interpretiert Eingaben im NEXT-Feld des geführten Dialogs als Daten.

NEXT-INPUT = *ANY
Die Art der nachfolgenden Eingabe ist nicht vorhersehbar.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) / list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien der Anweisung zugeordnet werden.

PRIVILEGE = *ALL
Die Anweisung erhält alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

PRIVILEGE = *EXCEPT(...)
Die Anweisung erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht der Anweisung zugeordnet werden.

PRIVILEGE = *ADD(...)
Die Anweisung erhält zusätzlich zu den bereits zugeordneten Privilegien die bei *ADD(...) angegebenen Privilegien.

ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien der Anweisung zusätzlich zugeordnet werden.

PRIVILEGE = *REMOVE(...)
Der Anweisung werden die bei *REMOVE(...) angegebenen Privilegien entzogen.

REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien der Anweisung entzogen werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Die Anweisung erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

STMT-VERSION = *UNCHANGED / *NONE / <integer 1..999>
Version der Anweisung. Der Wert wird im normierten Übergabebereich übergeben. *NONE bedeutet, dass ein NULL-Wert im normierten Übergabebereich eingetragen wird. STMT-VERSION wird ignoriert, falls das Programm, zu dem die Anweisung gehört, nicht mit den neuen Makros CMDRST und CMDTST arbeitet bzw. noch das alte Format des normierten Übergabebereiches verwendet. Mehr zum Format des normierten Übergabebereiches und zu den SDF-Makros finden Sie im Kapitel „SDF-Programmschnittstelle“ ab Seite 371ff und Seite 385ff.

MODIFY-STMT-ATTRIBUTES
Anweisungsattribute ändern

Mit der Anweisung MODIFY-STMT-ATTRIBUTES ändern Sie für mehrere Anweisungen eines Programmes gleichzeitig die sicherheitsrelevanten Attribute (Eingabemodi und Privilegien).

MODIFY-**STMT-ATT**RIBUTES

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

,**DIAL**OG-**ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**DIAL**OG-**PROC-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**GUID**ED-**ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**BATCH-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**BATCH-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES** / ***NO**

,**PRIV**ILEGE = ***UNCH**ANGED / ***ALL** / ***EXC**EPT(...) / ***ADD**(...) / ***REMOVE**(...) /
list-poss(64): <structured-name 1..30>

***EXC**EPT(...)

EXCEPT-**PRIV**ILEGE = list-poss(64): <structured-name 1..30>

***ADD**(...)

ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>

***REMOVE**(...)

REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Namen der Anweisungen, deren Attribute zu ändern sind.

PROGRAM = <structured-name 1..30>
Externer Name des Programmes, zu dem die Anweisungen gehören.
Dieser Name kann mit MODIFY-STMT-ATTRIBUTES nicht geändert werden.

Die Beschreibung aller anderen Operanden der Anweisung MODIFY-STMT-ATTRIBUTES finden Sie bei der Anweisung MODIFY-STMT ab Seite 270.

MODIFY-VALUE
Operandenwertdefinition ändern

Mit der Anweisung MODIFY-VALUE ändern Sie in der geöffneten Syntaxdatei die Definition eines Operandenwerts. Dieser Operandenwert muss „aktuelles“ Objekt sein. Falls der Wert eine Struktur einleitet, ist anschließend der erste Operand der Struktur aktuelles Objekt (siehe Seite 151). Andernfalls kommt es darauf an, ob zu dem Operanden, zu dem der geänderte Wert gehört, noch weitere Werte definiert sind. Falls ja, so wird der nächste zu diesem Operanden gehörende Wert aktuelles Objekt. Gibt es keinen weiteren Wert, so wird ggf. der in der Kommando- bzw. Anweisungsstruktur folgende Operand aktuelles Objekt. Wird der letzte Wert einer Struktur verändert, dann wird der Wert aktuelles Objekt, der auf den die Struktur einführenden Wert folgt. Ist dieser Wert nicht vorhanden, so wird der darauffolgende Operand aktuelles Objekt.

Die Anweisung MODIFY-VALUE gleicht weitgehend der Anweisung ADD-VALUE. Default-Wert für alle Operanden von MODIFY-VALUE ist *UNCHANGED, d.h. es werden nur die Teile der Operandenwertdefinition verändert, die explizit spezifiziert sind. Im geführten Dialog wird im Operandenfragebogen statt *UNCHANGED die aktuelle Operandenbesetzung angezeigt. Wenn Sie einen Wert ungleich *UNCHANGED angeben, so werden die alten Angaben in der Operandenwertdefinition durch die neuen ersetzt. Das gilt auch dann, wenn Listenangabe zulässig ist, d.h. die Werteliste wird nicht um den angegebenen Wert ergänzt, sondern durch ihn ersetzt. Für den STANDARD-NAME gibt es eine Sonderregelung. Wenn der Operandenwert in einer zugewiesenen Referenzdatei definiert ist (siehe OPEN-SYNTAX-FILE, Seite 303), bleiben die alten Angaben für STANDARD-NAME erhalten und der in MODIFY-VALUE angegebene Name kommt additiv hinzu.

Alle für den Operandenwert vergebenen Namen müssen in Bezug auf seine Umgebung eindeutig sein.

(Teil 1 von 6)

MODIFY-**VAL**UE

TYPE = ***UNCH**ANGED / ***ALPHA**NUMERIC-**NAME**(...) / ***CAT-ID** / ***COM**MAND-**REST**(...) /
***COMPOSED-NAME**(...) / ***C-STR**ING(...) / ***DATE**(...) / ***DEV**ICE(...) / ***FIXED**(...) /
***FILENAME**(...) / ***INTEG**ER(...) / ***KEYW**ORD(...) / ***KEYW**ORD-**NUM**BER(...) / ***LABEL**(...) /
***NAME**(...) / ***PAR**TIAL-**FILENAME**(...) / ***POS**IX-**PATH**NAME(...) / ***POS**IX-**FILENAME**(...) /
***PROD**UCT-**VERS**ION(...) / ***STRUCT**URED-**NAME**(...) / ***TEXT**(...) / ***TIME**(...) / ***VSN**(...) /
***X-STR**ING(...) / ***X-TEXT**(...)

***ALPHA**NUMERIC-**NAME**(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>
,**WILDC**ARD = ***UNCH**ANGED / ***NO** / ***YES**(...)

***YES**(...)

TYPE = ***UNCH**ANGED / ***SEL**ECTOR / ***CONSTR**UCTOR
,LONGEST-**LOG**ICAL-**LEN** = ***UNCH**ANGED / ***LONG**EST-**L**ENGTH / <integer 1..255>

***COM**MAND-**REST**(...)

LOWER-**C**ASE = ***UNCH**ANGED / **YES** / ***NO**

***COMPOSED-NAME**(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>

,**UNDER**SCORE = ***UNCH**ANGED / ***NO** / ***YES**

,**WILDC**ARD = ***UNCH**ANGED / ***NO** / ***YES**(...)

***YES**(...)

TYPE = ***UNCH**ANGED / ***SEL**ECTOR / ***CONSTR**UCTOR
,LONGEST-**LOG**ICAL-**LEN** = ***UNCH**ANGED / ***LONG**EST-**L**ENGTH / <integer 1..1800>

***C-STR**ING(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>

,**LOW**ER-**C**ASE = ***UNCH**ANGED / **YES** / ***NO**

***DATE**(...)

COMPLETION = ***UNCH**ANGED **/ *NO** / ***YES**

Fortsetzung ➠

(Teil 2 von 6)

```
*DEVICE(...)
    CLASS-TYPE = *UNCHANGED / list-poss(2000): *DISK(...) / *TAPE(...)
      *DISK(...)
        EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
        ,SCOPE = *UNCHANGED / *ALL / *STD-DISK
      *TAPE(...)
        EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
    ,ALIAS-ALLOWED = *UNCHANGED / *YES / *NO
    ,VOLUME-TYPE-ONLY = *UNCHANGED / *YES / *NO
    ,RESULT-VALUE = *UNCHANGED / *BY-NAME / *BY-CODE
*FIXED(...)
    LOWEST = *UNCHANGED / *ANY / <fixed -2147483648..2147483647>
    ,HIGHEST = *UNCHANGED / *ANY / <fixed -2147483648..2147483647>
*FILENAME(...)
    SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
    ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
    ,CATALOG-ID = *UNCHANGED / *YES / *NO
    ,USER-ID = *UNCHANGED / *YES / *NO
    ,GENERATION = *UNCHANGED / *YES / *NO
    ,VERSION = *UNCHANGED / *YES / *NO
    ,WILDCARD = *UNCHANGED / *NO / *YES(...)
      *YES(...)
        TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
        ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..80>
    ,PATH-COMPLETION = *UNCHANGED / *YES / *NO
    ,TEMPORARY-FILE = *UNCHANGED / *YES / *NO
*INTEGER(...)
    LOWEST = *UNCHANGED / *ANY / <integer -2147483648..2147483647>
    ,HIGHEST = *UNCHANGED / *ANY / <integer -2147483648..2147483647>
    ,OUT-FORM = *UNCHANGED / *BINARY / *PACKED / *UNPACKED / *CHAR / *STD
```

Fortsetzung ➠

(Teil 3 von 6)

***KEYWORD**(...)

STAR = ***UNCH**ANGED / ***OPT**IONAL / ***MAND**ATORY

***KEYWORD-NUM**BER(...)

STAR = ***UNCH**ANGED / ***OPT**IONAL / ***MAND**ATORY

***LABEL**(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..8>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..8>

***NAME**(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>

,**UNDER**SCORE = ***UNCH**ANGED / ***YES** / ***NO**

,**LOW**ER-**C**ASE = ***UNCH**ANGED / **YES** / ***NO**

,**WILDC**ARD = ***UNCH**ANGED / ***NO** / ***YES**(...)

***YES**(...)

TYPE = ***UNCH**ANGED / ***SEL**ECTOR / ***CONSTR**UCTOR

,LONGEST-**LOG**ICAL-**LEN** = ***UNCH**ANGED / ***LONG**EST-**L**ENGTH / <integer 1..255>

***PAR**TIAL-**FILENAME**(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 2..79>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 2..79>

,**CAT**ALOG-**ID** = ***UNCH**ANGED / ***YES** / ***NO**

,**USER-ID** = ***UNCH**ANGED / ***YES** / ***NO**

,**WILDC**ARD = ***UNCH**ANGED / ***NO** / ***YES**(...)

***YES**(...)

TYPE = ***UNCH**ANGED / ***SEL**ECTOR / ***CONSTR**UCTOR

,LONGEST-**LOG**ICAL-**LEN** = ***UNCH**ANGED / ***LONG**EST-**L**ENGTH / <integer 2..79>

***POS**IX-**PATH**NAME(...)

SHORTEST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1023>

,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1023>

,**WILDCARD** = ***UNCH**ANGED / ***YES** / ***NO**

,**QUOTES** = ***UNCH**ANGED / ***OPT**IONAL / ***MAND**ATORY

Fortsetzung ➠

(Teil 4 von 6)

***POS**IX-**FILENAME**(...)

- **SHORT**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>
- ,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>
- ,**WILDCARD** = ***UNCH**ANGED / ***YES** / ***NO**
- ,**QUOTES** = ***UNCH**ANGED / ***OPT**IONAL / ***MAND**ATORY

***PROD**UCT-**VERS**ION(...)

- **USER-INT**ERFACE = ***UNCH**ANGED / ***NO** / ***YES**(...) ***ANY**(...)
  - ***YES**(...)
    - **CORR**ECTION-**STA**TE = ***UNCH**ANGED /***YES** / ***NO** / ***ANY**
  - ***ANY**(...)
    - **CORR**ECTION-**STA**TE = ***UNCH**ANGED / ***ANY** / ***NO** / ***YES**

***STRUCT**URED-**NAME**(...)

- **SHORT**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>
- ,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..255>
- ,**WILDC**ARD = ***UNCH**ANGED / ***NO** / ***YES**(...)
  - ***YES**(...)
    - **TYPE** = ***UNCH**ANGED / ***SEL**ECTOR / ***CONSTR**UCTOR
    - **,LONG**EST-**LOG**ICAL-**LEN** = ***UNCH**ANGED / ***LONG**EST-**L**ENGTH / <integer 1..255>

***TEXT**(...)

- **SHORT**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>
- ,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>
- ,**LOW**ER-**C**ASE = ***UNCH**ANGED / ***YES** / ***NO**
- ,**SEP**ARATORS = ***UNCH**ANGED / ***YES** / ***NO**

***TIME**(...)

- **OUT-FORM** = ***UNCH**ANGED / ***STD** / ***BIN**ARY / ***CHAR**

***VSN**(...)

- **SHORT**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..6>
- ,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..6>

***X-STR**ING(...)

- **SHORT**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>
- ,**LONG**EST-**L**ENGTH = ***UNCH**ANGED / ***ANY** / <integer 1..1800>

Fortsetzung ➠

(Teil 5 von 6)

```
    *X-TEXT(...)
        |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
        |  ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
        |  ,ODD-POSSIBLE = *UNCHANGED / *YES / *NO
,INTERNAL-NAME = *UNCHANGED / *STD / <alphanum-name 1..8>
,REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
,SECRET-PROMPT = *UNCHANGED / *SAME / *NO
,DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
,DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,STRUCTURE = *UNCHANGED / *NO / *YES(...)
    *YES(...)
        |  SIZE = *UNCHANGED / *SMALL / *LARGE
        |  ,FORM = *UNCHANGED / *NORMAL / *FLAT
        |  ,MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
,LIST-ALLOWED = *UNCHANGED / *YES / *NO
,VALUE = *UNCHANGED / *NO / list-poss(2000): <c-string 1..1800 with-low>(...)
    <c-string 1..1800 with-low>(...)
        |  STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME /
        |                  <structured-name 1..30> / <c-string 1..30>
        |  ,ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
        |  ,GUIDED-ABBREVIATION = *UNCHANGED / *NAME / <structured-name 1..30> / <c-string 1..30>
        |  ,MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30> / <c-string 1..30>
        |  ,NULL-ABBREVIATION = *UNCHANGED / *YES / *NO
        |  ,OUTPUT = *UNCHANGED / SAME / *EMPTY-STRING / *DROP-OPERAND /
        |            <c-string 1..1800> / <x-string 1..3600>
```

Fortsetzung ➠

(Teil 6 von 6)

```
      ,OUT-TYPE = *UNCHANGED / *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST /
                  *COMPOSED-NAME / *C-STRING / *DATE / *DEVICE / *FIXED / *FILENAME /
                  *INTEGER / *KEYWORD / *KEYWORD-NUMBER / *LABEL / *NAME /
                  *PARTIAL-FILENAME / *PRODUCT-VERSION / *POSIX-PATHNAME /
                  *POSIX-FILENAME / *STRUCTURED-NAME / *TEXT / *TIME / *VSN /
                  *X-STRING / *X-TEXT
      ,OVERWRITE-POSSIBLE = *UNCHANGED / *YES / *NO
,OUTPUT = *UNCHANGED / *SECRET-PROMPT / *NORMAL(...)
   *NORMAL(...)
      AREA-LENGTH = *UNCHANGED / *VARIABLE / <integer 1..3000>
      ,LEFT-JUSTIFIED = *UNCHANGED / *STD / *YES / *NO
      ,FILLER = *UNCHANGED / *STD / <c-string 1..1> / <x-string 1..2>
      ,STRING-LITERALS = *UNCHANGED / *HEX / *CHAR / *NO
      ,HASH = *UNCHANGED / *NO / *YES(...)
         *YES(...)
            OUTPUT-LENGTH = *UNCHANGED / <integer 2..32>
,PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
      EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

TYPE =
Bestimmt, von welchem Datentyp der Operandenwert ist. Die verschiedenen zu einem Operanden definierten Werte dürfen sich hinsichtlich des Datentyps nicht ausschließen (siehe Seite 635). (Ist dies in Ausnahmefällen nicht möglich, so muss in der Anweisung ADD-OPERAND oder MODIFY-OPERAND für VALUE-OVERLAPPING der Wert *YES angegeben werden.) Ansonsten ist es beispielsweise nicht möglich, für einen Operanden einen Wert vom Typ NAME und einen alternativen Wert vom Typ STRUCTURED-NAME zu definieren. Lediglich der Datentyp KEYWORD darf in mehr als einer alternativen Operandenwertdefinition angegeben sein. Bei der Kommando- bzw. Anweisungseingabe prüft SDF, ob ein eingegebener Operandenwert von dem für ihn definierten Typ ist. Bei der folgenden Beschreibung der Datentypen wird wiederholt der Begriff „alphanumerisches Zeichen“ benutzt. Dieses kann ein Buchstabe (A,B,C,..,Z), eine Ziffer (0,1,2,...,9) oder ein Sonderzeichen (@,#,$) sein.

TYPE = *UNCHANGED
Keine Änderung bezüglich des Datentyps.

TYPE = *ALPHANUMERIC-NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp ALPHANUMERIC-NAME ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD = *UNCHANGED / *NO / *YES(...)
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..255>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *CAT-ID
Bestimmt, dass der Operandenwert vom Datentyp CAT-ID ist. Dieser ist definiert als eine Folge von maximal 4 Zeichen (A-Z, 0-9; Sonderzeichen $, @, # sind nicht erlaubt). Die Begrenzungszeichen ':' werden nicht mitgezählt. Eine 4 Zeichen lange CAT-ID darf nicht mit der Zeichenfolge 'PUB' beginnen.

TYPE = *COMMAND-REST(...)
Bestimmt, dass der Operandenwert vom Datentyp COMMAND-REST ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *COMPOSED-NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp COMPOSED-NAME ist. Dieser Datentyp entspricht weitgehend dem Datentyp FILENAME, hat jedoch folgende Abweichungen:
– nicht erlaubt sind: catid, userid, version, generation, wildcard
– keine Einschränkung der Länge auf maximal 54 Zeichen
– es muss nicht zwingend ein Buchstabe vorhanden sein

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

UNDERSCORE = *UNCHANGED / *NO / *YES
Bestimmt, ob der Unterstrich '_' akzeptiert wird.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..1800>
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..1800>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *C-STRING(...)
Bestimmt, dass der Operandenwert vom Datentyp C-STRING ist. Dieser ist definiert als eine Folge von EBCDIC-Zeichen, die in Hochkommas eingeschlossen ist. Ihr kann der Buchstabe C vorangestellt sein. Ein Hochkomma als Wert innerhalb der begrenzenden Hochkommas ist bei der Eingabe doppelt anzugeben.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

LOWER-CASE = *UNCHANGED / *NO / *YES
Bestimmt, ob in Hochkommas eingeschlossene Kleinbuchstaben erhalten bleiben.

TYPE = *DATE(...)
Bestimmt, dass der Operandenwert vom Datentyp DATE ist. Dieser ist definiert als eine Folge von einer vierstelligen und zwei zweistelligen Zahlen, die durch Bindestrich miteinander verbunden sind (<Jahr>-<Monat>-<Tag>). Das Jahr kann auch mit zwei Ziffern an Stelle von vier angegeben werden.

COMPLETION = *UNCHANGED / *NO / *YES
Bestimmt, ob eine zweistellige Jahresangabe komplettiert wird. Bei Angabe von *YES ergänzt SDF zweistellige Jahresangaben (Datum der Form jj-mm-tt) zu:
– 20jj-mm-tt falls jj < 60
– 19jj-mm-tt falls jj ≥ 60.

TYPE = *DEVICE(...)
Bestimmt, dass der Operandenwert vom Datentyp DEVICE ist. Für Operanden, deren Operandenwert mit dem Datentyp DEVICE definiert ist, wird dem Benutzer im geführten Dialog eine Liste der im System verfügbaren Platten- oder Bandgeräte angeboten (siehe auch Handbuch „Systeminstallation“ [9]).

CLASS-TYPE = *UNCHANGED / list-poss(2000): *DISK(...) / *TAPE(...)
gibt die Geräteklasse an.

CLASS-TYPE = *DISK(...)
Das Gerät gehört zur Klasse der Plattengeräte.

EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
Bestimmt, welche Plattengeräte nicht in der Liste der verfügbaren Geräte erscheinen sollen.

SCOPE = *UNCHANGED / *ALL / *STD-DISK
Legt fest, ob alle Plattengeräte in der Liste der verfügbaren Geräte erscheinen oder nur die über DVS festgelegten Standard-Plattengeräte (siehe Handbuch „Einführung in das DVS“ [7]).
In BS2000/OSD-BC < V3.0 gilt immer SCOPE=*ALL.

SCOPE = *ALL
Alle Plattengeräte erscheinen in der Liste.

SCOPE = *STD-DISK
Nur die Plattengeräte, die im DVS als Standard-Plattengeräte festgelegt sind, erscheinen in der Liste.

CLASS-TYPE = *TAPE(...)
Das Gerät gehört zur Klasse der Bandgeräte.

EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
Bestimmt, welche Geräte nicht in der Liste der verfügbaren Geräte erscheinen sollen.

ALIAS-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob die Angabe des Alias-Namens des Gerätes erlaubt ist.

VOLUME-TYPE-ONLY = *UNCHANGED / *NO / *YES
Bestimmt, ob der Volumetyp akzeptiert wird.

RESULT-VALUE = *UNCHANGED / *BY-NAME / *BY-CODE
Bestimmt, in welcher Form SDF die Information an die Implementierung übergibt.

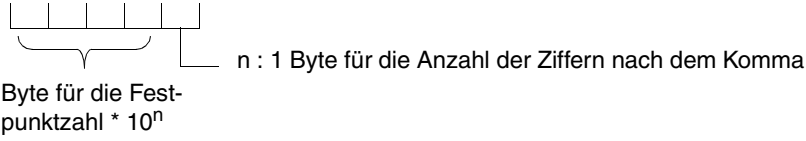
RESULT-VALUE = *BY-NAME
SDF übergibt den externen Gerätenamen. Der externe Gerätename ist 8 Zeichen lang.

RESULT-VALUE = *BY-CODE
SDF übergibt den internen Gerätecode. Der interne Gerätecode ist 2 Byte lang.

TYPE = *FIXED(...)
Bestimmt, dass der Operandenwert vom Datentyp FIXED ist. Dieser ist wie folgt definiert:

[Zeichen][Ziffern].[Ziffern]
[Zeichen] entspricht + oder –
[Ziffern] entspricht 0..9

FIXED muss aus mindestens einer Ziffer, darf aber höchstens aus 10 Zeichen (Ziffern und ein '.') bestehen. Im normierten Übergabebereich wird der Wert im folgenden Format abgelegt:



LOWEST = *UNCHANGED / *ANY / <fixed -2147483648 .. 2147483647>
Bestimmt, ob es für die Festpunktzahl eine untere Grenze gibt und ggf. welche.

HIGHEST = *UNCHANGED / *ANY / <fixed -2147483648 .. 2147483647>
Bestimmt, ob es für die Festpunktzahl eine obere Grenze gibt und ggf. welche.

TYPE = *FILENAME(...)
Bestimmt, dass der Operandenwert vom Datentyp FILENAME ist. Für die einzugebende Zeichenfolge gilt die Definition, die im Handbuch „Einführung in das DVS“ [7] für den vollqualifizierten Dateinamen angegeben ist.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

CATALOG-ID = *UNCHANGED / *YES / *NO
Bestimmt, ob die Katalogkennung als Teil des Dateinamens angegeben werden darf.

USER-ID = *UNCHANGED / *YES / *NO
Bestimmt, ob die Benutzerkennung als Teil des Dateinamens angegeben werden darf.

GENERATION = *UNCHANGED / *YES / *NO
Bestimmt, ob eine Generationsnummer als Teil des Dateinamens angegeben werden darf.

VERSION = *UNCHANGED / *YES / *NO
Bestimmt, ob eine Versionsbezeichnung als Teil des Dateinamens angegeben werden darf.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden. Der Datentyp <filename x..y with-wild> beinhaltet auch teilqualifizierte Dateinamen, d.h. es ist nicht notwendig, für den Operanden zusätzlich einen Wert vom Datentyp <partial-filename> zu definieren.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..80>
Gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..80>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

PATH-COMPLETION = *UNCHANGED / *NO / *YES
Legt fest, ob der Dateiname bei der Übergabe an die Implementierung zum vollständigen Pfadnamen ergänzt wird. Das schließt auch die Ersetzung von Aliasnamen durch die ACS-Funktion ein.
PATH-COMPLETION=*YES darf nur angegeben werden, wenn CATALOG-ID und USER-ID erlaubt sind und wenn Wildcards im Dateinamen nicht erlaubt sind.

TEMPORARY-FILE = *UNCHANGED / *YES / *NO
Gibt an, ob temporäre Dateinamen erlaubt sind.

TYPE = *INTEGER(...)
Bestimmt, dass der Operandenwert vom Datentyp INTEGER ist. Dieser ist definiert als eine Folge von Ziffern, der ein Vorzeichen vorangestellt sein kann.

LOWEST = *UNCHANGED / *ANY / <integer -2147483648 .. 2147483647>
Bestimmt, ob es für die Ganzzahl eine untere Grenze gibt und ggf. welche.

HIGHEST = *UNCHANGED / *ANY / <integer -2147483648 .. 2147483647>
Bestimmt, ob es für die Ganzzahl eine obere Grenze gibt und ggf. welche.

OUT-FORM = *UNCHANGED / *STD / *BINARY / *PACKED / *UNPACKED / *CHAR
Bestimmt, in welchem Format SDF die Ganzzahl an die Implementierung übergibt. Erfolgt die Übergabe in einem Übergabebereich (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371), so setzt SDF-A *STD in *BINARY um. Bei Übergabe einer Zeichenkette (Kommandos, die mit IMPLEMENTOR=*PROCEDURE(...) oder IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...) definiert sind, siehe ADD-CMD) setzt SDF-A OUT-FORM=*STD in *CHAR um.

TYPE = *KEYWORD(...)
Bestimmt, dass der Operandenwert vom Datentyp KEYWORD ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen. Diese Zeichenfolge kann in mehrere Teilzeichenfolgen gegliedert sein, die durch Bindestrich miteinander verbunden sind. Teilzeichenfolgen dürfen den Punkt enthalten. Der Punkt darf jedoch nicht am Beginn oder Ende einer Teilzeichenfolge stehen. Die gesamte Zeichenfolge, d.h. ggf. die erste Teilzeichenfolge muss mit einem Buchstaben oder Sonderzeichen beginnen. Der Wertebereich für einen Operandenwert vom Typ KEYWORD ist auf einen oder eine endliche Anzahl von genau festgelegten Einzelwerten beschränkt (siehe Operand VALUE in dieser Anweisung).
Ab SDF-A/SDF Version 2.0 wird auch der Wert '*...' akzeptiert. Dieser Wert kann verwendet werden, wenn für einen Operanden eine Liste von Schlüsselwörtern definiert werden muss (z.B. die Definition für alle externen Geräte). Damit können neue Schlüsselwörter (z.B. neue Gerätenamen) eingefügt werden, ohne dass die Syntaxdatei verändert werden muss, denn mit '*...' akzeptiert SDF zusätzliche Werte und analysiert sie als Schlüsselwörter. Der Datentyp KEYWORD darf maximal 30 Zeichen lang sein.

STAR =
Bestimmt, ob bei der Eingabe der Zeichenfolge ein Stern vorangestellt werden muss.

STAR = *UNCHANGED
keine Änderung bezüglich des Sterns.

STAR = *OPTIONAL
Ein Stern darf, muss aber nicht vorangestellt werden.
Falls ein Stern vorangestellt ist, wird er bei der Übergabe an die Implementierung unterdrückt.

STAR = *MANDATORY
Ein Stern muss vorangestellt werden (erforderlich zur Unterscheidung von alternativen Datentypen).

TYPE = *KEYWORD-NUMBER(...)
Bestimmt, dass der Operandenwert vom Datentyp KEYWORD-NUMBER ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

STAR = *UNCHANGED
keine Änderung bezüglich des Sterns.

STAR = *OPTIONAL
Ein Stern darf, muss aber nicht vorangestellt werden.

STAR = *MANDATORY
Ein Stern muss vorangestellt werden (erforderlich zur Unterscheidung von alternativen Datentypen).

TYPE = *LABEL(...)
Bestimmt, dass der Operandenwert vom Datentyp LABEL ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp NAME ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen, die mit einem Buchstaben oder Sonderzeichen beginnt.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

UNDERSCORE = *UNCHANGED / *NO / *YES
Bestimmt, ob '_' (Unterstrich) akzeptiert wird.

LOWER-CASE = *UNCHANGED / *NO / *YES
Legt fest, ob Kleinbuchstaben erhalten bleiben.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..255>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *PARTIAL-FILENAME(...)
Bestimmt, dass der Operandenwert vom Datentyp PARTIAL-FILENAME ist. Für die einzugebende Zeichenfolge gilt die Definition, die im Handbuch „Einführung in das DVS“ [7] für den teilqualifizierten Dateinamen angegeben ist.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 2..79>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 2..79>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

CATALOG-ID = *UNCHANGED / *YES / *NO
Bestimmt, ob die Katalogkennung als Teil des Dateinamens angegeben werden darf.

USER-ID = *UNCHANGED / *YES / *NO
Bestimmt, ob die Benutzerkennung als Teil des Dateinamens angegeben werden darf.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 2..79>
gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 2..79>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *POSIX-PATHNAME(...)
Bestimmt, dass der Operandenwert vom Datentyp POSIX-PATHNAME ist. Für die einzugebende Zeichenfolge gelten folgende Konventionen:
– erlaubt sind Buchstaben, Ziffern und die Zeichen '_' , '-' , '.' und '/' , wobei '/' immer als Trennzeichen zwischen den Verzeichnissen und Dateinamen dient. Steuerzeichen sind nicht erlaubt.
– Ein POSIX-PATHNAME besteht aus POSIX-FILENAMEs, die durch '/' getrennt sind. Ein POSIX-PATHNAME darf insgesamt nicht länger als 1023 Zeichen sein.
– Das Zeichen '\' (Backslash) entwertet Metazeichen in allen POSIX-spezifischen Namen. Es wird nicht mitgezählt. Das Metazeichen * wird ebenfalls nicht mitgezählt.

Metazeichen sind Zeichen, die für Wildcard-Ausdrücke verwendet werden. Folgende Metazeichen können auftreten:

*****	kein, ein oder mehrere beliebige Zeichen
?	genau ein beliebiges Zeichen
[S]	genau ein Zeichen aus der mit S bestimmten Zeichenmenge
[!S]	genau ein Zeichen, das aber nicht Element der mit S bestimmten Zeichenmenge sein darf
wobei S	eine Menge von Zeichen (z.B. abcd) oder ein Bereich von Zeichen (z.B. a-d) oder eine Kombination aus Mengen- und Bereichsangaben sein kann.

POSIX-PATHNAMEs, die andere als die oben genannten Zeichen oder am Anfang '?' oder '!' enthalten, müssen bei der Eingabe in Hochkommas eingeschlossen sein (wie C-STRINGs). Im normierten Übergabebereich bzw. im Übergabe-String werden die Hochkommas von SDF wieder entfernt, da sie nicht zum Dateinamen gehören. Hochkommas, die zum Dateinamen gehören, müssen verdoppelt werden.
In Prozeduren sollte zur Vermeidung von Kompatibilitätsproblemen immer die C-STRING-Syntax verwendet werden.

SHORTEST-LENGTH = *ANY / <integer 1..1023>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..1023>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD = *YES / *NO
Bestimmt, ob Metazeichen (siehe oben) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

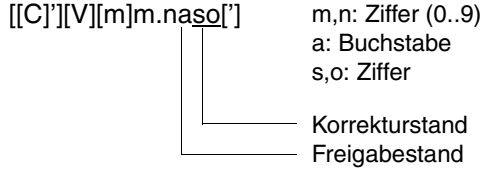
TYPE = *POSIX-FILENAME(...)
Bestimmt, dass der Operandenwert vom Datentyp POSIX-FILENAME ist. Für die einzugebende Zeichenfolge gelten die gleichen Konventionen wie für POSIX-PATHNAMEs (siehe Seite 290), mit folgenden Einschränkungen:
– '/' (Slash) darf nicht enthalten sein.
– Die Länge ist auf maximal 255 Zeichen begrenzt.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD = *YES / *NO
Bestimmt, ob Metazeichen (siehe POSIX-PATHNAME) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

TYPE = *PRODUCT-VERSION(...)
Bestimmt, dass der Operandenwert vom Datentyp PRODUCT-VERSION ist. Die Produktversion hat folgendes Format:



Die optionalen Zeichen C, V und Hochkomma werden im SDF-Übergabebereich unterdrückt.

USER-INTERFACE = *UNCHANGED / *NO / *YES(...) / *ANY(...)
Legt fest, ob der Freigabestand der Benutzerschnittstelle angegeben werden muss oder darf.

USER-INTERFACE = *NO
Der Freigabestand der Benutzerschnittstelle und der Korrekturstand dürfen nicht angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:

[[C]'][V][m]m.n['].

USER-INTERFACE = *YES(...)
Der Freigabestand der Benutzerschnittstelle muss angegeben werden.

CORRECTION-STATE = *UNCHANGED / *YES / *NO / *ANY
Legt fest, ob auch der Korrekturstand angegeben werden muss oder darf.

CORRECTION-STATE = *YES
Der Korrekturstand muss angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.naso['].

CORRECTION-STATE = *NO
Der Korrekturstand darf nicht angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.na['].

CORRECTION-STATE = *ANY
Der Korrekturstand kann angegeben werden.

USER-INTERFACE = *ANY(...)
Der Freigabestand der Benutzerschnittstelle kann angegeben werden.

CORRECTION-STATE = *UNCHANGED / *ANY / *NO / *YES
Legt fest, ob auch der Korrekturstand angegeben werden muss oder darf.

CORRECTION-STATE = *ANY
Der Korrekturstand kann angegeben werden.

CORRECTION-STATE = *NO
Der Korrekturstand darf nicht angegeben werden. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.na['].

CORRECTION-STATE = *YES
Der Korrekturstand muss angegeben werden, wenn der Freigabestand angegeben wird. Angaben zum Datentyp PRODUCT-VERSION haben dann folgendes Format:
[[C]'][V][m]m.naso['].

TYPE = *STRUCTURED-NAME(...)
Bestimmt, dass der Operandenwert vom Datentyp STRUCTURED-NAME ist. Dieser ist definiert als eine Folge von alphanumerischen Zeichen. Diese Zeichenfolge kann in mehrere Teilzeichenfolgen gegliedert sein, die durch Bindestrich miteinander verbunden sind. Die gesamte Zeichenfolge, d.h. ggf. die erste Teilzeichenfolge muss mit einem Buchstaben oder Sonderzeichen beginnen.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

WILDCARD = *UNCHANGED / *NO / *YES(...)
Bestimmt, ob Platzhalterzeichen („Wildcards“, siehe SDF-Metasyntax Seite 18) an Stelle von Zeichen(-folgen) innerhalb des Namens bei der Kommando- bzw. Anweisungseingabe angegeben werden dürfen.

WILDCARD = *NO
Als Eingabe für den Operandenwert sind keine Wildcards zulässig.

WILDCARD = *YES(...)
Wildcards dürfen angegeben werden.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Gibt an, ob die Zeichenfolge ein Wildcard-Suchmuster oder ein Wildcard-Konstruktor sein kann. Wildcard-Konstruktoren werden zur Namensbildung aus Zeichenfolgen genutzt, die mithilfe eines Wildcard-Suchmusters entstanden sind.

TYPE = *SELECTOR
Die Zeichenfolge kann ein Wildcard-Suchmuster sein. Der Datentyp erhält den Zusatz with-wild (siehe SDF-Metasyntax Seite 18).

TYPE = *CONSTRUCTOR
Die Zeichenfolge kann ein Wildcard-Konstruktor sein. Der Datentyp erhält den Zusatz with-wild-constr (siehe SDF-Metasyntax Seite 20).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
gibt die maximale Länge des Namens an, zu dem der Wildcard-Ausdruck (Suchmuster oder Konstruktor) noch als passend erkannt werden soll.

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
Die maximale Länge des zum Wildcard-Ausdruck passenden Namens ist gleich der beim Operanden LONGEST-LENGTH angegebenen Länge (aus Gründen der Kompatibilität).

LONGEST-LOGICAL-LEN = <integer 1..255>
Bestimmt, dass der zum Wildcard-Ausdruck passende Name die angegebene Länge nicht überschreiten darf.

TYPE = *TEXT(...)
Bestimmt, dass der Operandenwert vom Datentyp TEXT ist. Dieser Datentyp ist nur für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers vorgesehen. Er wird deshalb hier nicht beschrieben.

TYPE = *TIME(...)
Bestimmt, dass der Operandenwert vom Datentyp TIME ist. Dieser ist definiert als eine Folge von ein, zwei oder drei vorzeichenlosen Zahlen, die durch Doppelpunkt miteinander verbunden sind (<Stunden>[:<Minuten> [:<Sekunden>]]). Die Angabe für Stunden, Minuten und Sekunden darf maximal zweistellig sein. Zahlen, die weniger Stellen haben, dürfen Nullen vorangestellt sein. Der Wertebereich für Minuten und Sekunden liegt zwischen 0 und 59.

OUT-FORM = *UNCHANGED / *STD / *BINARY / *CHAR
Bestimmt, in welchem Format SDF die Zahlendarstellung der Zeitangabe an die Implementierung übergibt.

OUT-FORM = *STD
Erfolgt die Übergabe in einem Übergabebereich (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371), wird die Zeitangabe im Binär-Format übergeben. Bei Übergabe in einer Zeichenkette (bei Kommandos, die mit IMPLEMENTOR= *PROCEDURE(...) oder IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...) definiert sind, siehe ADD-CMD) wird die Zeitangabe im Character-Format übergeben.

OUT-FORM = *BINARY
Die Zeitangabe wird im Binär-Format übergeben.

OUT-FORM = *CHAR
Die Zeitangabe wird im Character-Format übergeben.

TYPE = *VSN(...)
Bestimmt, dass der Operandenwert vom Datentyp VSN ist. Ab V1.4A kann SDF zwischen zwei Typen unterscheiden:

1. VSN mit einem Punkt:
   Diese VSN muss aus 6 Zeichen bestehen. Neben einem einzelnen Punkt werden nur die Buchstaben A-Z und die Ziffern 0-9 akzeptiert.
   Eine solche VSN hat das Format `pvsid.sequence-number`, wobei `pvsid` aus 2 bis 4 Zeichen und `sequence-number` aus 1 bis 3 Zeichen besteht.
   Diesem untergeordneten Typ von VSN darf PUB nicht vorangestellt werden:
   z.B. ist PUBA.0 oder PUB.02 ungültig. Der Punkt kann 3., 4. oder 5. Zeichen der VSN sein.

2. VSN ohne Punkt: Diese VSN besteht aus einer Zeichenfolge von maximal 6 Zeichen. Nur die Buchstaben A-Z, die Ziffern 0-9 und die Sonderzeichen $, @,# sind erlaubt. Eine solche VSN kann mit PUB beginnen. In diesem Fall dürfen die folgenden Zeichen keine Sonderzeichen sein (z.B. wird PUB@1 oder PUB$## zurückgewiesen). SDF macht keine weiteren Unterschiede zwischen Public oder Private VSN oder PUBRES.

   SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..6>
   Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..6>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

TYPE = *X-STRING(...)
Bestimmt, dass der Operandenwert vom Datentyp X-STRING ist. Dieser ist definiert als eine Folge von Sedezimalzeichen (Ziffern 0-9, Großbuchstaben A-F), die in Hochkommas eingeschlossen ist. Ihr vorangestellt ist der Buchstabe X.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Byte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Byte).

TYPE = *X-TEXT(...)
Bestimmt, dass der Operandenwert vom Datentyp X-TEXT ist. Dieser Datentyp entspricht weitgehend dem Datentyp X-STRING, hat jedoch keine Hochkommas und kein vorangestelltes 'X'.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss und ggf. welche (Angabe in Halbbyte).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf und ggf. welche (Angabe in Halbbyte).

ODD-POSSIBLE = *UNCHANGED / *YES / *NO
Bestimmt, ob eine ungerade Zahl von Zeichen akzeptiert wird.

INTERNAL-NAME = *UNCHANGED / *STD / <alphanum-name 1..8>
Interner Operandenwertname. SDF identifiziert einen Operandenwert mithilfe dieses Namens. Bei Angabe von *STD nimmt SDF-A die ersten acht Zeichen (ohne Bindestrich) des Datentyps, den Sie beim Operanden TYPE angegeben haben. Bei Operandenwerten vom Datentyp KEYWORD nimmt SDF-A bei Angabe von *STD die ersten acht Zeichen (ohne Bindestrich) des ersten Werts, den Sie beim Operanden VALUE angegeben haben.

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operandenwert gelöscht werden darf (siehe REMOVE). Wenn der Operandenwert zu einem Kommando gehört und in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE) mit REMOVE-POSSIBLE=*NO definiert ist, so lehnt SDF-A eine Änderung in *YES ab.

SECRET-PROMPT = *UNCHANGED / *SAME / *NO
Gibt an, ob der Operandenwert als geheim behandelt werden soll.
SECRET-PROMPT=*SAME übernimmt die Einstellung des Operanden, zu dem der hier definierte Operandenwert gehört (siehe ADD-OPERAND ...,SECRET-PROMPT=, Seite 156). Das Eingabefeld wird für geheime Operandenwerte dunkelgesteuert und die Protokollierung wird unterdrückt.
Bei SECRET-PROMPT=*NO wird der Operandenwert nicht als geheim behandelt.
Ist ein geheimer Operand mit einem nicht geheimen Wert vorbesetzt, so wird das Eingabefeld nicht dunkelgesteuert. Durch Eingabe des Zeichens ^ oder eines mit OUTPUT= *SECRET-PROMPT definierten Wertes kann die Dunkelsteuerung des Eingabefeldes angefordert werden.

DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operandenwert im Dialogbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Dialogbetrieb zugelassen ist.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operandenwert im Dialogbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Dialogbetrieb innerhalb einer Prozedur zugelassen ist.

GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operandenwert im geführten Dialog angeboten wird. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im geführten Dialog zugelassen ist.

BATCH-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operandenwert im Stapelbetrieb zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Stapelbetrieb zugelassen ist.

BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Bestimmt, ob der Operandenwert im Stapelbetrieb innerhalb einer Prozedur zugelassen ist. Die Angabe *YES setzt voraus, dass der Operand, zu dem der Wert gehört, im Stapelbetrieb innerhalb einer Prozedur zugelassen ist.

STRUCTURE =
Bestimmt, ob der Operandenwert eine Struktur einleitet.

STRUCTURE = *UNCHANGED
keine Änderung bezüglich der Struktureinleitung.

STRUCTURE = *NO
Der Operandenwert leitet keine Struktur ein.

STRUCTURE = *YES(...)
Der Operandenwert leitet eine Struktur ein.

SIZE = *UNCHANGED / *SMALL / *LARGE
Bestimmt, ob in der MINIMUM- oder MEDIUM-Stufe des geführten Dialogs die Struktur in den übergeordneten Fragebogen integriert wird (*SMALL) oder ob für sie ein eigener Fragebogen angelegt wird (*LARGE).

FORM = *UNCHANGED / *FLAT / *NORMAL
*Ist nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*
Bei Angabe von *FLAT wird die Struktur im Übergabebereich für die Implementierung linearisiert, die Operanden der Struktur werden in ein übergeordnetes Operanden-Array integriert. Im Fall *NORMAL erhält die Struktur ein eigenes Operanden-Array. In ihm werden die Operanden übergeben, für die als RESULT-OPERAND-LEVEL eine höhere Strukturebene definiert ist als für den Operanden, zu dem der hier definierte struktureinleitende Operandenwert gehört (siehe ADD-OPERAND...,RESULT-OPERAND-LEVEL=,... und Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371).

MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
Anzahl der in der strukturierten Übergabe zu reservierenden Operandenpositionen. Bei Angabe von *STD wird das Operanden-Array so groß wie für die Struktur erforderlich angelegt. Für geplante künftige Erweiterungen lässt es sich aber auch größer anlegen. Dieser Operand bezieht sich auf die durch den Operandenwert eingeleitete Struktur und ist nur relevant, wenn im vorigen Operanden *NORMAL angegeben ist.

LIST-ALLOWED = *UNCHANGED / *NO / *YES
Bestimmt, ob bei der Kommando- bzw. Anweisungseingabe für den Operandenwert die Angabe einer Liste zulässig ist. Das setzt voraus, dass der Operand, zu dem der Wert gehört, mit LIST-POSSIBLE=*YES definiert ist (siehe ADD-OPERAND).

VALUE =
Bestimmt, welche Werte als Eingabe zulässig sind.

VALUE = *UNCHANGED
keine Änderung bezüglich der zulässigen Eingabewerte.

VALUE = *NO
Alle dem Operandentyp entsprechenden Werte sind zulässig. Einschränkungen gibt es nur, sofern diese bei der Festlegung des Operandentyps angegeben sind (z.B. Längenbegrenzungen). Für Operanden des Typs KEYWORD ist *NO nicht zulässig.

VALUE = list-poss(2000): <c-string 1..1800 with-low>(...)
Der Operandenwert muss einen der angegebenen Werte haben (zwingend für Werte vom Typ KEYWORD). Einen angegebenen Wert vom Typ KEYWORD kann der Benutzer im Gegensatz zum STANDARD-NAME und zum ALIAS-NAME bei der Eingabe abkürzen. Ist der Operandenwert vom Typ KEYWORD, so ist Listenangabe nicht zulässig.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30> / <c-string 1..30>
Ist nur für Operandenwerte des Typs KEYWORD relevant.
Bestimmt den Standardnamen des Operandenwerts, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Dieser ist bei der Eingabe nicht abkürzbar. Ein STANDARD-NAME darf im Gegensatz zum ALIAS-NAME nicht gelöscht werden, solange der Operandenwert mit diesem Namen in einer der zugewiesenen Referenzsyntaxdateien (siehe OPEN-SYNTAX-FILE) existiert. Wird der ursprüngliche, in der Kommando- bzw. Programmdokumentation als Schlüsselwort genannte Einzelwert zum Standardnamen erklärt, so ist sichergestellt, dass der Operandenwert ungeachtet aller Änderungen mit dem ursprünglichen Schlüsselwort eingegeben werden kann. Bei Angabe von *NAME nimmt SDF-A als STANDARD-NAME den Einzelwert, den Sie beim Operanden VALUE angegeben haben.

GUIDED-ABBREVIATION = *UNCHANGED / *NAME / <structured-name 1..30> / <c-string 1..30>
Ist nur für Operandenwerte des Typs KEYWORD relevant.
Bestimmt den Namen, mit dem SDF in der mittleren Hilfestufe des geführten Dialogs den Operandenwert bezeichnet.Bei Angabe von *NAME nimmt SDF-A als GUIDED-ABBREVIATION den Einzelwert, den Sie beim Operanden VALUE angegeben haben.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
Ist nur für Operandenwerte des Typs KEYWORD relevant.
Bestimmt den Aliasnamen des Operandenwerts, der bei der Kommando- bzw. Anweisungseingabe alternativ benutzt werden kann. Dieser ist bei der Eingabe nicht abkürzbar. Ein ALIAS-NAME darf im Gegensatz zum STANDARD-NAME gelöscht werden.

MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30> / <c-string 1..30>
Nur für Operandenwerte vom Typ KEYWORD:
Bestimmt die kürzest mögliche Abkürzung für den Operandenwert. Eine kürzere Abkürzung wird von SDF dann nicht akzeptiert.
Folgendes ist zu beachten:

1. Die Überprüfung der Minimal-Abkürzung erfolgt erst *nachdem* SDF die Eingabe auf Eindeutigkeit überprüft hat. Es kann also passieren, dass SDF zwar den richtigen Operandenwert auswählt, ihn aber dann ablehnt, weil bei der Eingabe die Abkürzung kürzer als die vorgeschriebene Minimal-Abkürzung gewählt wurde.
2. Die Minimal-Abkürzung muss aus dem KEYWORD entstehen.

3. Die ALIAS-NAMEs und STANDARD-NAMEs des Operandenwerts dürfen nicht kürzer als die Minimal-Abkürzung sein, wenn sie Abkürzung des Operandenwerts sind.
4. In einer Syntax-Datei-Hierarchie darf nur eine Verkürzung der Minimal-Abkürzung (keine Verlängerung) vorgenommen werden.

NULL-ABBREVIATION = *UNCHANGED / *NO / *YES
*Ist nur für Operandenwerte des Typs KEYWORD relevant, die mit STRUCTURE=*YES definiert sind.*
Bestimmt, ob der Operandenwert bei der Kommando- bzw. Anweisungseingabe vor der öffnenden Strukturklammer weggelassen werden kann, z.B. ein struktureinleitender Operandenwert, zu dem es keine alternativen Operandenwerte gibt.

OUTPUT =
Bestimmt, welcher Wert im Fall OUTPUT=*NORMAL (siehe Seite 300) an die Implementierung übergeben wird.

OUTPUT = *UNCHANGED
keine Änderung bezüglich des zu übergebenden Werts.

OUTPUT = *SAME
Der beim Operanden VALUE angegebene Wert wird übergeben.

OUTPUT = *EMPTY-STRING
Ein leerer String wird übergeben.

OUTPUT = *DROP-OPERAND
Die Übergabe des Operanden wird unterdrückt.

OUTPUT = <c-string 1..1800>
Der hier angegebene Wert wird übergeben.

OUTPUT = <x-string 1..3600>
Der hier angegebene Wert wird übergeben.

OUT-TYPE = *UNCHANGED / *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST / *COMPOSED-NAME / *C-STRING / *DATE / *DEVICE / *FIXED / *FILENAME / *INTEGER / *KEYWORD-NUMBER / *NAME / *PARTIAL-FILENAME /*POSIX-PATHNAME / *POSIX-FILENAME / *PRODUCT-VERSION / *STRUCTURED-NAME / *TEXT / *TIME / *VSN / *X-STRING / *X-TEXT
*Ist nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*
Bestimmt, ob und ggf. wie SDF den Datentyp des Operandenwertes ändert, wenn der Wert im Übergabebereich für die Implementierung abgelegt wird (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff). Bei Angabe von *SAME verändert SDF den Datentyp nicht.

OVERWRITE-POSSIBLE = *UNCHANGED / *NO / *YES

*Ist nur relevant für Anweisungen und mit IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) definierte Kommandos (siehe ADD-CMD).*

Bestimmt, ob der eingegebene Operandenwert durch einen von der Implementierung dynamisch erzeugten Wert überschrieben wird (siehe Operand DEFAULT in den Makros CMDRST und CMDTST). Eine der für den Operanden existierenden Operandenwertdefinitionen muss den von der Implementierung erzeugten Wert abdecken. Im geführten Dialog zeigt SDF den von der Implementierung erzeugten Wert im Fragebogen. Beispiel: Den Wert UNCHANGED in den MODIFY-Anweisungen von SDF-A überschreibt SDF-A mit dem aktuellen Wert.

OUTPUT =

Bestimmt, ob und ggf. wie SDF den eingegebenen Operandenwert an die Implementierung übergibt.

OUTPUT = *UNCHANGED

Keine Änderung bezüglich der Übergabe des Operandenwerts.

OUTPUT = *NORMAL(...)

SDF übergibt den Operandenwert an die Implementierung. Die Angaben AREA-LENGTH=, LEFT-JUSTIFIED= und FILLER= sind ab SDF-A Version 2.0 nicht mehr auf bestimmte Operandenwerte beschränkt.

AREA-LENGTH = *UNCHANGED / *VARIABLE / <integer 1..3000>

Bestimmt die Länge des Feldes, in dem SDF den Operandenwert für die Implementierung ablegt. Das Feld muss lang genug sein, um den größten Wert aufnehmen zu können, der in der Ablaufphase eingegeben werden kann. Wird für AREA-LENGTH ein kleinerer Wert als für LONGEST-LENGTH eingegeben, gibt SDF eine Warnung aus und akzeptiert den für AREA-LENGTH angegebenen Wert.

Bei der Analyse eines Wertes, der länger als AREA-LENGTH und kürzer als LONGEST-LENGTH ist, können 2 Fälle auftreten:

1. Werte vom Typ C-STRING mit LONGEST-LENGTH ≤ 32, die zu einem geheimen Operanden gehören, werden von SDF komprimiert und in einem sedezimalen String mit der Länge AREA-LENGTH abgelegt. Dieses Verhalten kann zum Beispiel für Kennworte genutzt werden. Die Kennworte werden mithilfe eines Hash-Algorithmus komprimiert und sind durch die sedezimale Ablageform gesichert gegen unberechtigte Zugriffe.

   Hinweise:

   – Es wird derselbe Hash-Algorithmus wie bei der in SDF-P verfügbaren HASH-STRING-Funktion verwendet.
   – Der Kommandoserver oder das Programm, das den von SDF analysierten Wert verarbeitet, muss möglicherweise angepasst werden, wenn die Kennwort-Definition zur Unterstützung von Hash-Kennworten geändert wurde. Eventuell wird der von SDF zurückgegebene Hash-Wert von der Semantik-Analyse des Programmes oder Kommandoservers abgewiesen.

2. In allen von 1. abweichenden Fällen wird der Wert auf die bei AREA-LENGTH angegebene Länge gekürzt.

LEFT-JUSTIFIED = *UNCHANGED / *STD / *YES / *NO
ist nur relevant, wenn für das Ablagefeld eine feste Länge bestimmt wurde. Mit LEFT-JUSTIFIED wird bestimmt, ob SDF den Operandenwert links- oder rechtsbündig in dem Feld ablegt. Bei numerischen Werten setzt SDF-A *STD in *NO um, bei allen anderen Werten in *YES.

FILLER = *UNCHANGED / *STD / <c-string 1..1> / <x-string 1..2>
ist nur relevant, wenn für das Ablagefeld eine feste Länge bestimmt wurde. Mit FILLER wird bestimmt, mit welchem Zeichen SDF das Feld im Bedarfsfall auffüllt. Bei Werten vom Typ X-STRING oder INTEGER setzt SDF-A *STD in X'00' um, bei allen übrigen in C'␣' (Leerzeichen).

STRING-LITERALS = *UNCHANGED / *NO / *HEX / *CHAR
Bestimmt, ob SDF den Operandenwert für die Übergabe an die Implementierung in den Datentyp X-STRING oder C-STRING umwandelt. Bei Angabe von *NO verändert SDF den Datentyp nicht. In diesem Fall ist bei Operandenwerten vom Typ C-STRING zu beachten, dass SDF nur den Inhalt des Strings übergibt (ohne Hochkomma und vorangestelltes C). Dieser Operand ist nur gültig, wenn VALUE=*NO angegeben wird.

HASH = *UNCHANGED / *NO / *YES(...)
Legt fest, ob der Eingabewert mittels eines Hash-Algorithmus in einen Wert mit einer definierten Länge umgewandelt wird.

HASH = *YES(...)
Ist nur erlaubt für Operandenwerte vom Datentyp C-STRING, die mit LONGEST-LENGTH Î 32 definiert werden.
Die anderen Operanden in der Struktur OUTPUT=*NORMAL(...) formatieren den Wert erst nach Ausführung der Hash-Funktion. Der Wert hat anschließend den Datentyp X-STRING und enthält dann eventuell auch nicht druckbare Zeichen.

OUTPUT-LENGTH = <integer 2..32>
Länge des Wertes, in den der Eingabewert umgewandelt wird.

OUTPUT = *SECRET-PROMPT
Der Operandenwert wird nicht an die Implementierung übergeben, sondern bewirkt, dass SDF den Benutzer auffordert, einen der zu dem Operanden gehörenden alternativen Werte einzugeben. Die folgende Eingabe erfolgt dann mit dunkelgesteuertem Eingabefeld und wird nicht protokolliert. Voraussetzungen sind:
– der Operand ist als geheim definiert
(siehe ADD-OPERAND...,SECRET-PROMPT=*YES)
– die Eingabe erfolgt im ungeführten Dialog oder in einer Vordergrundprozedur
– für den mit SECRET-PROMPT definierten Operandenwert ist als zulässige Eingabe ein Einzelwert angegeben (siehe Operand VALUE=<c-string>, im Normalfall ist er vom Typ KEYWORD)

Im geführten Dialog ergibt sich folgender Anwendungsfall:
Das Eingabefeld für einen geheimen Operanden, der mit einem nichtgeheimen Wert vorbesetzt ist, wird nicht dunkelgesteuert. Durch Eingabe eines mit OUTPUT=*SECRET-PROMPT definierten Wertes kann das Eingabefeld dunkelgesteuert werden.

PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) /
list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien dem Operandenwert zugeordnet werden.

PRIVILEGE = *SAME
Der Operandenwert erhält die gleichen Privilegien wie der Operand, zu dem dieser Operandenwert gehört.

PRIVILEGE = *EXCEPT(...)
Der Operandenwert erhält mit Ausnahme der bei *EXCEPT(...) angegebenen Privilegien alle zurzeit definierten Privilegien sowie alle Privilegien, die zu einem späteren Zeitpunkt definiert werden.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Gibt an, welche Privilegien nicht dem Operandenwert zugeordnet werden.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Der Operandenwert erhält nur genau die Privilegien, die Sie in dieser Liste angeben.

OPEN-SYNTAX-FILE
Syntaxdatei öffnen

Mit der Anweisung OPEN-SYNTAX-FILE öffnen Sie eine Syntaxdatei zur Bearbeitung mit SDF-A. Sie können vor dem Öffnen der Syntaxdatei mit der Anweisung DEFINE-ENVIRONMENT festlegen, welches Format die Syntaxdatei haben soll. Jede weitere OPEN-SYNTAX-FILE-Anweisung bewirkt implizit, dass SDF-A die zuvor geöffnete Syntaxdatei schließt.

Eine Syntaxdatei kann mit (Standardeinstellung) oder ohne Hierarchie geöffnet werden. Eine Veränderung der Hierarchie zwischen zwei Operationen kann zu Fehlern führen. Es wird daher empfohlen, eine Syntaxdatei immer in der gleichen Hierarchie zu bearbeiten.

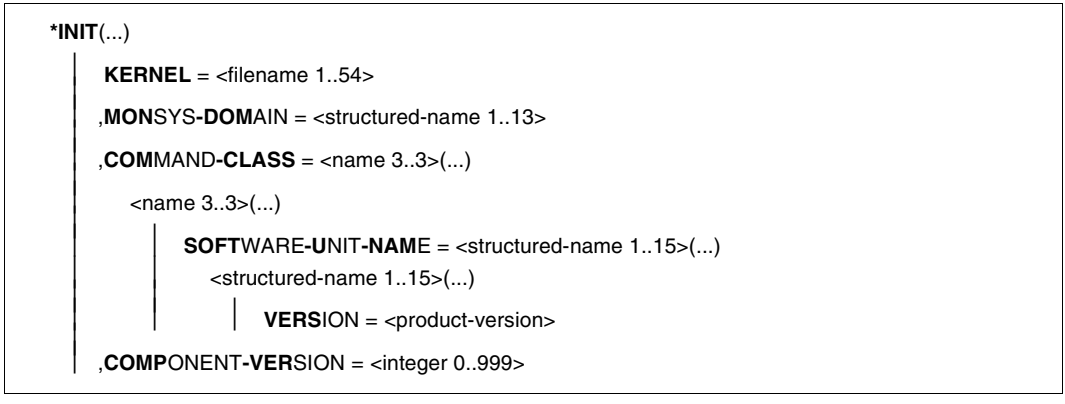
(Teil 1 von 2)

OPEN-SYNTAX-FILE

FILE = <filename 1..54>
,**TYPE** = ***USER**(...) / ***GROUP**(...) / ***SYST**EM(...)
  ***USER**(...)
    GROUP-DESCRIPTIONS = ***CURR**ENT / ***NO** / <filename 1..54>
    ,**SYSTEM-DESCR**IPTIONS = ***CURR**ENT / ***NO** / <filename 1..54>
    ,**USER-CONTR**OL = ***NO** / <filename 1..54>
  ***GROUP**(...)
    SYSTEM-DESCRIPTIONS = ***CURR**ENT / ***NO** / <filename 1..54>
    ,**SYS**TEM-**CONTR**OL = ***NO** / <filename 1..54>
  ***SYST**EM(...)
    SYSTEM-**CONTR**OL = ***NO** / <filename 1..54>
,**MODE** = ***UPD**ATE(...) / ***CRE**ATE / ***READ** / ***INIT**(...)
  ***UPD**ATE(...)
    COMPONENT-**VER**SION = ***UNCH**ANGED / <integer 0..999>
    ,**SOFT**WARE-**U**NIT-**NAM**E = ***NOCHECK**(...) / <structured-name 1..15>(...)
      ***NOCHECK**(...)
        VERSION = ***UNCH**ANGED / <product-version>
      <structured-name 1..15>(...)
        VERSION = ***UNCH**ANGED / <product-version>

Fortsetzung ➠

(Teil 2 von 2)



FILE = <filename 1..54>
Name der zu öffnenden Syntaxdatei.

TYPE =
Art der zu öffnenden Syntaxdatei.

TYPE = *USER(...)
Eine Benutzersyntaxdatei ist zu öffnen.

GROUP-DESCRIPTIONS =
Bestimmt, ob SDF-A beim Bearbeiten der zu öffnenden Benutzersyntaxdatei auf den Inhalt einer Gruppensyntaxdatei zugreifen soll.

GROUP-DESCRIPTIONS = *CURRENT
SDF-A soll auf die Gruppensyntaxdatei zugreifen, die für die aktuelle Task aktiviert ist.

GROUP-DESCRIPTIONS = *NO
SDF-A soll auf keine Gruppensyntaxdatei zugreifen.

GROUP-DESCRIPTIONS = <filename 1..54>
Name der Gruppensyntaxdatei, auf die SDF-A zugreifen soll.

SYSTEM-DESCRIPTIONS =
Bestimmt, ob SDF-A beim Bearbeiten der zu öffnenden Benutzersyntaxdatei auf den Inhalt einer Systemsyntaxdatei zugreifen soll.

SYSTEM-DESCRIPTIONS = *CURRENT
SDF-A soll auf die derzeit aktivierte Systemsyntaxdatei zugreifen.

SYSTEM-DESCRIPTIONS = *NO
SDF-A soll auf keine Systemsyntaxdatei zugreifen.

SYSTEM-DESCRIPTIONS = <filename 1..54>
Name der Systemsyntaxdatei, auf die SDF-A zugreifen soll.

USER-CONTROL =
Bestimmt, ob SDF-A überwachen soll, dass Definitionen von benutzereigenen Kommandos/Anweisungen nicht in unzulässiger Weise geändert werden. Für diese Überwachung benötigt SDF-A eine Benutzersyntaxdatei, in der die Definitionen der benutzereigenen Kommandos/Anweisungen in ihrer Originalfassung abgelegt sind.

USER-CONTROL = *NO
Die Überwachung soll nicht stattfinden.

USER-CONTROL = <filename 1..54>
Name der Benutzersyntaxdatei, die die zur Überwachung benötigten Kommando-/Anweisungsdefinitionen enthält.

TYPE = *GROUP(...)
Eine Gruppensyntaxdatei ist zu öffnen.

SYSTEM-DESCRIPTIONS =
Bestimmt, ob SDF-A beim Bearbeiten der zu öffnenden Gruppensyntaxdatei auf den Inhalt einer Systemsyntaxdatei zugreifen soll.

SYSTEM-DESCRIPTIONS = *CURRENT
SDF-A soll auf die derzeit aktivierte Systemsyntaxdatei zugreifen.

SYSTEM-DESCRIPTIONS = *NO
SDF-A soll auf keine Systemsyntaxdatei zugreifen.

SYSTEM-DESCRIPTIONS = <filename 1..54>
Name der Systemsyntaxdatei, auf die SDF-A zugreifen soll.

SYSTEM-CONTROL =
Bestimmt, ob SDF-A überwachen soll, dass Definitionen von kennungsspezifisch angebotenen Kommandos/Anweisungen nicht in unzulässiger Weise geändert werden. Für diese Überwachung benötigt SDF-A eine Systemsyntaxdatei, in der die Definitionen dieser Kommandos/Anweisungen in ihrer Originalfassung abgelegt sind.

SYSTEM-CONTROL = *NO
Die Überwachung soll nicht stattfinden.

SYSTEM-CONTROL = <filename 1..54>
Name der Systemsyntaxdatei, die die zur Überwachung benötigten Kommando-/Anweisungsdefinitionen enthält.

TYPE = *SYSTEM(...)
Eine Systemsyntaxdatei ist zu öffnen.

SYSTEM-CONTROL =
Bestimmt, ob SDF-A überwachen soll, dass Definitionen von systemweit angebotenen Kommandos/Anweisungen nicht in unzulässiger Weise geändert werden. Für diese Überwachung benötigt SDF-A als Referenzsyntaxdatei eine Systemsyntaxdatei, in der die Definitionen dieser Kommandos/Anweisungen in ihrer Originalfassung abgelegt sind.

SYSTEM-CONTROL = *NO
Die Überwachung soll nicht stattfinden.

SYSTEM-CONTROL = <filename 1..54>
Name der Systemsyntaxdatei, die die zur Überwachung benötigten Kommando-/Anweisungsdefinitionen enthält.

MODE =
Legt fest, wie die geöffnete Syntaxdatei bearbeitet werden darf und ob sie neu anzulegen ist.

MODE = *UPDATE(...)
Der Inhalt der Syntaxdatei kann sowohl ausgegeben als auch verändert werden. Die Syntaxdatei existiert bereits. Sie darf nicht aktiviert sein.

COMPONENT-VERSION = *UNCHANGED / <integer 0..999>
Der Operand ist reserviert für die Systemsoftwareentwicklung von Fujitsu Siemens Computers.

SOFTWARE-UNIT-NAME = *NOCHECK(...) / <structured-name 1..15>(...)
Der Operand ist reserviert für die Systemsoftwareentwicklung von Fujitsu Siemens Computers.

MODE = *CREATE
Der Inhalt der Syntaxdatei kann sowohl ausgegeben als auch verändert werden. SDF-A legt die Syntaxdatei neu an. Dazu gehört auch, dass SDF-A die Globalinformation erstellt. Falls bereits eine Datei gleichen Namens existiert, so lehnt SDF-A das Öffnen der Datei ab und gibt eine Fehlermeldung aus.

MODE = *READ
Der Inhalt der Syntaxdatei kann nur ausgegeben, aber nicht verändert werden (nur Lesezugriff). Die Syntaxdatei existiert bereits. Sie darf aktiviert sein.

MODE = *INIT(...)
Der Operandenwert ist reserviert für die Systemsoftwareentwicklung von Fujitsu Siemens Computers.

REMOVE
Objekte der Syntaxdatei löschen

Mit der Anweisung REMOVE löschen Sie Objekte in der geöffneten Syntaxdatei. Objekte in diesem Sinne sind Anwendungsbereiche, Programme, Kommandos, Anweisungen, Operanden oder Operandenwerte. Der im Folgenden benutzte Begriff „löschen“ beinhaltet zweierlei:

1. SDF-A entfernt aus der geöffneten Syntaxdatei die Definition des zu löschenden Objektes. Dies gilt für alle Arten von Syntaxdateien.
2. SDF-A schreibt in die bearbeitete Benutzer- oder Gruppensyntaxdatei die Information, dass das zu löschende Objekt gesperrt ist. Voraussetzung dafür ist, dass Sie beim Öffnen der Syntaxdatei mit dem Operanden GROUP- bzw. SYSTEM-DESCRIPTIONS eine Gruppen- bzw. Systemsyntaxdatei angegeben haben, in der das zu löschende Objekt definiert ist. In diesem Fall kann der Kommandoname nicht für ein anderes Kommando verwendet werden.

Das Entfernen obligatorischer Operanden aus einem Systemkommando ist nicht erlaubt. Das verhindert einen dauernden Syntaxfehler während der Ausführung.

Die SDF-Standardanweisungen können nicht gelöscht werden.

REMOVE OBJECT=*DOMAIN entfernt auch die Kommandos, die nur diesem Bereich zugeordnet sind, außer wenn das Kommando mit REMOVE-POSSIBLE=*NO definiert ist.

Solange ein Objekt mit REMOVE-POSSIBLE=*NO (ADD- oder MODIFY-Anweisungen) definiert ist, kann es nicht gelöscht werden.

(Teil 1 von 4)

REMOVE

OBJECT = ***PRIV**ILEGE(...) / ***DOM**AIN(...) / ***COM**MAND(...) / ***PROG**RAM(...) / ***STATEM**ENT(...) /
***OPER**AND(...) / ***VAL**UE(...) / ***CORR**ECTION-**INFO**RMATION(...)

***PRIV**ILEGE(...)
NAME = <structured-name 1..30>

***DOM**AIN(...)
NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
***ALL**(...)
EXCEPT = **<u>*NONE</u>** / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
,**REM**OVE-**ATT-COMMANDS** = **<u>*YES</u>** / ***NO**

Fortsetzung ➠

(Teil 2 von 4)

***COM**MAND(...)

NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***ALL**(...)

EXCEPT = ***NONE** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***PROG**RAM(...)

NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***ALL**(...)

EXCEPT = ***NONE** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***STATEM**ENT(...)

NAME = ***ALL(...)** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

***ALL**(...)

EXCEPT = ***NONE** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

***OPER**AND(...)

OPERAND-**L1** = ***CUR**RENT / <structured-name 1..20>

,**VAL**UE-**L1** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** / ***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** / ***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** / ***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION / ***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** / ***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** / ***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** / ***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION / ***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

Fortsetzung ➠

(Teil 3 von 4)

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**ORIG**IN = ***CUR**RENT / ***COM**MAND(...) / ***STATEM**ENT(...)

***COM**MAND(...)

NAME = <structured-name 1..30>

***STATEM**ENT(...)

NAME = <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

***VAL**UE(...)

OPERAND-**L1** = ***ABOVE-CUR**RENT / <structured-name 1..20>

,**VAL**UE-**L1** = ***CUR**RENT / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

Fortsetzung ➠

(Teil 4 von 4)

```
,OPERAND-L3 = *NO / <structured-name 1..20>
,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
            *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
            *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
            *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
            *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
,OPERAND-L4 = *NO / <structured-name 1..20>
,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
            *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
            *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
            *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
            *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
,OPERAND-L5 = *NO / <structured-name 1..20>
,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
            *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
            *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
            *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
            *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
  *COMMAND(...)
    NAME = <structured-name 1..30>
  *STATEMENT(...)
    NAME = <structured-name 1..30>
    ,PROGRAM = <structured-name 1..30>
*CORRECTION-INFORMATION(...)
  PM-NUMBER = *ALL / list-poss(20): <alphanum-name 8..8>
```

OBJECT =
Art des Objekts, das gelöscht werden soll.

OBJECT = *PRIVILEGE(...)
Bestimmt, dass ein Privileg gelöscht wird. Dieser Operandenwert ist der Systemsoftware-entwicklung von Fujitsu Siemens Computers vorbehalten.

OBJECT = *DOMAIN(...)
Bestimmt, dass Anwendungsbereiche gelöscht werden. Die diesen Anwendungsbereichen zugeordneten Kommandos werden *nicht* gelöscht, wenn:
– die Kommandos noch mindestens einem anderen Anwendungsbereich zugeordnet sind,
– die Kommandos mit REMOVE-POSSIBLE=*NO definiert sind oder
– Sie REMOVE-ATT-COMMANDS=*NO angeben.

Anwendungsbereiche enthalten keine Anweisungen; daher können beim Löschen von Anwendungsbereichen auch keine Anweisungen gelöscht werden.

NAME = *ALL(...)
Alle Anwendungsbereiche werden gelöscht.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die hier angegebenen Anwendungsbereiche werden nicht gelöscht.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die namentlich genannten Anwendungsbereiche werden gelöscht, bzw. die Anwendungsbereiche, deren Name zum Wildcard-Suchmuster passt.

REMOVE-ATT-COMMANDS = *YES / *NO
Bestimmt, ob die dem Anwendungsbereich zugeordneten Kommandos gelöscht werden.

OBJECT = *COMMAND(...)
Bestimmt, dass Kommandos gelöscht werden. Das schließt das Löschen der zugehörigen Operanden mit ein.

NAME = *ALL(...)
Alle Kommandos werden gelöscht.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die hier angegebenen Kommandos werden nicht gelöscht.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die namentlich genannten Kommandos werden gelöscht, bzw. die Kommandos, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *PROGRAM(...)
Bestimmt, dass Programme gelöscht werden. Das schließt das Löschen der zugehörigen Anweisungen mit ein.

NAME = *ALL(...)
Alle Programme werden gelöscht.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die hier angegebenen Programme werden nicht gelöscht.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die namentlich genannten Programme werden gelöscht, bzw. die Programme, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *STATEMENT(...)
Bestimmt, dass Anweisungen gelöscht werden. Das schließt das Löschen der zugehörigen Operanden mit ein.

NAME = *ALL(...)
Alle Anweisungen werden gelöscht.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die hier angegebenen Anweisungen werden nicht gelöscht.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die namentlich genannten Anweisungen werden gelöscht, bzw. die Anweisungen, deren Name zum Wildcard-Suchmuster passt.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisungen gehören.

OBJECT = *OPERAND(...)
Bestimmt, dass ein Operand gelöscht wird. Das schließt das Löschen der zugehörigen Operandenwerte mit ein. Steht der zu löschende Operand in einer Struktur, so wird er durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig (oder gar nicht) angegeben werden. Ein Operand, der zur Identifizierung des zu löschenden Operanden nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten VALUE-Li = *NO betrachtet SDF-A den durch OPERAND-Li bestimmten Operanden als den, der zu löschen ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu VALUE-Li ein anderer Wert als *NO angegeben, so muss zu OPERAND-Li+1 ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Bestimmt den zu löschenden Operanden (VALUE-L1=*NO) oder einen Operanden, der auf dem Pfad zu dem zu löschenden liegt (VALUE-L1≠*NO). *CURRENT bedeutet, dass OPERAND-L1 aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *POSIX-PATHNAME / *POSIX-FILENAME / *PRODUCT-VERSION / <composed-name 1..30>
Die Angabe *NO bedeutet, dass OPERAND-L1 zu löschen ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet, die den zu löschenden Operanden unmittelbar oder mittelbar enthält. Ist der struktureinleitende Operandenwert vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,..., VALUE=<c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der struktureinleitende Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L2 für die Spezifizierung des zu löschenden Operanden irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der zu löschende Operand (VALUE-L2=*NO) oder ein Operand, der auf dem Pfad zu dem zu löschenden liegt (VALUE-L2≠*NO).

VALUE-L2 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des zu löschenden Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet, die den zu löschenden Operanden unmittelbar oder mittelbar enthält. Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung des zu löschenden Operanden irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der zu löschende Operand (VALUE-L3=*NO) oder ein Operand, der auf dem Pfad zu dem zu löschenden liegt (VALUE-L3≠*NO).

VALUE-L3 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung des zu löschenden Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet, die den zu löschenden Operanden unmittelbar oder mittelbar enthält. Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L2

VALUE-L4= analog VALUE-L1
siehe VALUE-L2

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L2

ORIGIN =
Bestimmt das Kommando oder die Anweisung, zu dem der zu löschende Operand gehört.

ORIGIN = *CURRENT
Der Operand gehört zu einem Kommando/einer Anweisung, das zurzeit entweder selbst aktuelles Objekt ist oder einen Operanden oder Operandenwert enthält, der aktuelles Objekt ist.

ORIGIN = *COMMAND(...)
Der Operand gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Der Operand gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *VALUE(...)
Bestimmt, dass ein Operandenwert gelöscht wird. Das schließt das Löschen der zugehörigen Strukturen mit ein. Der zu löschende Operandenwert wird durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Gehört der zu löschende Operandenwert zu einem Operanden, der außerhalb jeder Struktur steht, so liegt auf dem Pfad nur dieser Operand. Gehört der zu löschende Operandenwert zu einem Operanden, der in einer Struktur steht, so liegen auf dem Pfad außerdem die übergeordneten Operanden und die zu diesen gehörenden struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur oder kommando- bzw. anweisungsglobal, so muss der Pfad nicht vollständig angegeben werden. Ein Operand, der zur Identifizierung des zu löschenden Operandenwerts nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten OPERAND-Li+1=*NO betrachtet SDF-A den durch VALUE-Li bestimmten Operandenwert als den, dessen Definition zu löschen ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu OPERAND-Li ein anderer Wert als *NO angegeben, so muss zu VALUE-Li ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Bestimmt den Operanden, zu dem der zu löschende Operandenwert gehört (OPERAND-L2=*NO), oder einen Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L2≠*NO). *ABOVE-CURRENT bedeutet, dass ein zu OPERAND-L1 gehörender Wert aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1 = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *POSIX-PATHNAME / *POSIX-FILENAME / *PRODUCT-VERSION / <composed-name 1..30>
Bestimmt den zu löschenden Operandenwert (OPERAND-L2=*NO) oder einen struktureinleitenden Operandenwert, der auf dem Pfad zu dem zu löschenden Operandenwert liegt (OPERAND-L2≠*NO). *CURRENT bedeutet, dass VALUE-L1 aktuelles Objekt ist. Ist er nicht aktuelles Objekt und vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD, ..., VALUE= <c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der Operandenwert nicht vom Typ KEYWORD bzw. KEYWORD-NUMBER, so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass VALUE-L1 zu löschen ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der zu löschende Operandenwert gehört (OPERAND-L3=*NO), oder der Name eines Operanden, der auf dem Pfad zu dem zu löschenden Operandenwert liegt (OPERAND-L3≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig sein.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PSOIX-PATHNAME / * POSIX-FILENAME / *PRODUCT-VERSION / <composed-name 1..30>
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des zu löschenden Operandenwerts irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser ist entweder der zu löschende Operandenwert (OPERAND-L3=*NO) oder ein struktureinleitender Operandenwert, der auf dem Pfad zu dem zu löschenden Operandenwert liegt (OPERAND-L3≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung des zu löschenden Operandenwerts irrelevant ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der zu löschende Operandenwert gehört (OPERAND-L4=*NO), oder der Name eines Operanden, der auf dem Pfad zu dem zu löschenden Operandenwert liegt (OPERAND-L4≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig sein.

VALUE-L3 = analog VALUE-L2
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung des zu löschenden Operandenwerts irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser ist entweder der zu löschende Operandenwert (OPERAND-L4=*NO) oder ein struktureinleitender Operandenwert, der auf dem Pfad zu dem zu löschenden Operandenwert liegt (OPERAND-L4≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L4 = analog VALUE-2
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L5 = analog VALUE-2
siehe VALUE-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, zu dem der zu löschende Operandenwert gehört.

ORIGIN = *CURRENT
Der Operandenwert gehört zu einem Kommando (bzw. zu einer Anweisung), das zurzeit entweder selbst aktuelles Objekt ist oder einen Operanden oder Operandenwert enthält, der aktuelles Objekt ist.

ORIGIN = *COMMAND(...)
Der Operandenwert gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Der Operandenwert gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *CORRECTION-INFORMATION(...)
Reserviert für die Systemsoftwareentwicklung von Fujitsu Siemens Computers.

RESTORE
Objekte der Syntaxdatei rekonstruieren

Mit der Anweisung RESTORE heben Sie die Sperre für Objekte in der geöffneten Syntaxdatei auf. Die Sperre wurde mit der Anweisung REMOVE gesetzt. Die Sperre kann für Kommandos, Anweisungen und Programme aufgehoben werden. Deren Beschreibungen müssen in der zu bearbeitenden Syntaxdatei festgelegt sein oder in einer Syntaxdatei, die höher in der Dateihierarchie steht. Physikalisch gelöschte Objekte können nicht wiederhergestellt werden. Die RESTORE-Anweisung kann nicht angewandt werden, wenn die zu bearbeitende Syntaxdatei die Systemsyntaxdatei ist.

```
RESTORE

OBJECT = *COMMAND(...) / *PROGRAM(...) / *STATEMENT(...)
  *COMMAND(...)
    | NAME = *ALL / list-poss(2000): <structured-name 1..30>
  *PROGRAM(...)
    | NAME = *ALL / list-poss(2000): <structured-name 1..30>
  *STATEMENT(...)
    | NAME = *ALL / list-poss(2000): <structured-name 1..30>
    | ,PROGRAM = <structured-name 1..30>
```

OBJECT =
Art des Objektes, für das die Sperre aufgehoben werden soll.

OBJECT = *COMMAND(...)
Bestimmt, dass die Sperre für Kommandos aufgehoben werden soll.

NAME = *ALL / list-poss(2000): <structured-name 1..30>
Für alle bzw. die namentlich genannten Kommandos wird die Sperre aufgehoben.

OBJECT = *STATEMENT(...)
Bestimmt, dass die Sperre für Anweisungen aufgehoben werden soll.

NAME = *ALL / list-poss(2000): <structured-name 1..30>
Für alle bzw. die namentlich genannten Anweisungen wird die Sperre aufgehoben.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisungen gehören.

OBJECT = *PROGRAM(...)
Bestimmt, dass die Sperre für Programme aufgehoben werden soll. Dabei wird nicht nur die Sperre für das Programm selbst, sondern für alle zum Programm gehörenden Anweisungen aufgehoben.

NAME = *ALL / list-poss(2000): <structured-name 1..30>
Für alle bzw. für die namentlich genannten Programme wird die Sperre aufgehoben.

SET-GLOBALS
Globalinformation ändern

Die Globalinformation enthält allgemeine Festlegungen über die Kommando-/Anweisungseingabe und -verarbeitung. Diese Festlegungen werden wirksam, sobald die Syntaxdatei aktiviert wird. Mit dem Kommando bzw. der Anweisung MODIFY-SDF-OPTIONS lassen sich die geltenden Festlegungen taskspezifisch ändern.

Beim erstmaligen Öffnen einer Syntaxdatei erstellt SDF-A standardmäßig die Globalinformation. Mit der Anweisung SET-GLOBALS ändern Sie diese Festlegungen. Wenn Sie im geführten Dialog mit der Anweisung SET-GLOBALS arbeiten, so sind, falls die Globalinformation aktuelles Objekt ist (siehe EDIT), im Fragebogen die Operanden mit den aktuellen Festlegungen vorbesetzt. Andernfalls erfolgt die Vorbesetzung mit dem Default-Wert *UNCHANGED.

(Teil 1 von 11)

SET-GLOBALS

VERSION = ***UNCH**ANGED / <alphanum-name 1..12> / <c-string 1..12>

,**GUID**ANCE = ***UNCH**ANGED / ***STD** / ***MAX**IMUM / ***MED**IUM / ***MIN**IMUM / ***NO** / ***EXP**ERT

,**LOG**GING = ***UNCH**ANGED / ***STD** / ***INP**UT**-FORM** / ***ACCEP**TED**-FORM** / ***INVAR**IANT**-FORM**

,**PROC**EDURE**-DIALOG** = ***UNCH**ANGED / ***STD** / ***YES** / ***NO**

,**UTIL**ITY**-INTERF**ACE = ***UNCH**ANGED / ***STD** / ***OLD** / ***NEW**

,**CONT**INUATION = ***UNCH**ANGED / ***STD** / ***OLD** / ***NEW**

,**FUNC**TION**-KEYS** = ***UNCH**ANGED / ***STD** / ***OLD-MODE** / ***STYL**E**-GUI**DE**-MODE** / ***BY-TERM**INAL**-TYPE**

,**INP**UT**-HIST**ORY = ***UNCH**ANGED / ***STD** / ***ON** / ***OFF**

,**NUMB**ER**-OF-INP**UTS = ***UNCH**ANGED / ***STD** / <integer 1..100>

,**GEN**ERAL**-INFO-VERS**ION = ***UNCH**ANGED / <integer 1..255>

,**MOD**IFY**-LANG**UAGE**-T**EXT = ***UNCH**ANGED / list-poss(2000): <name 1..1>(...)

<name 1..1>(...)

COMMAND**-REST** = ***NAMES**(...)

***NAMES**(...)

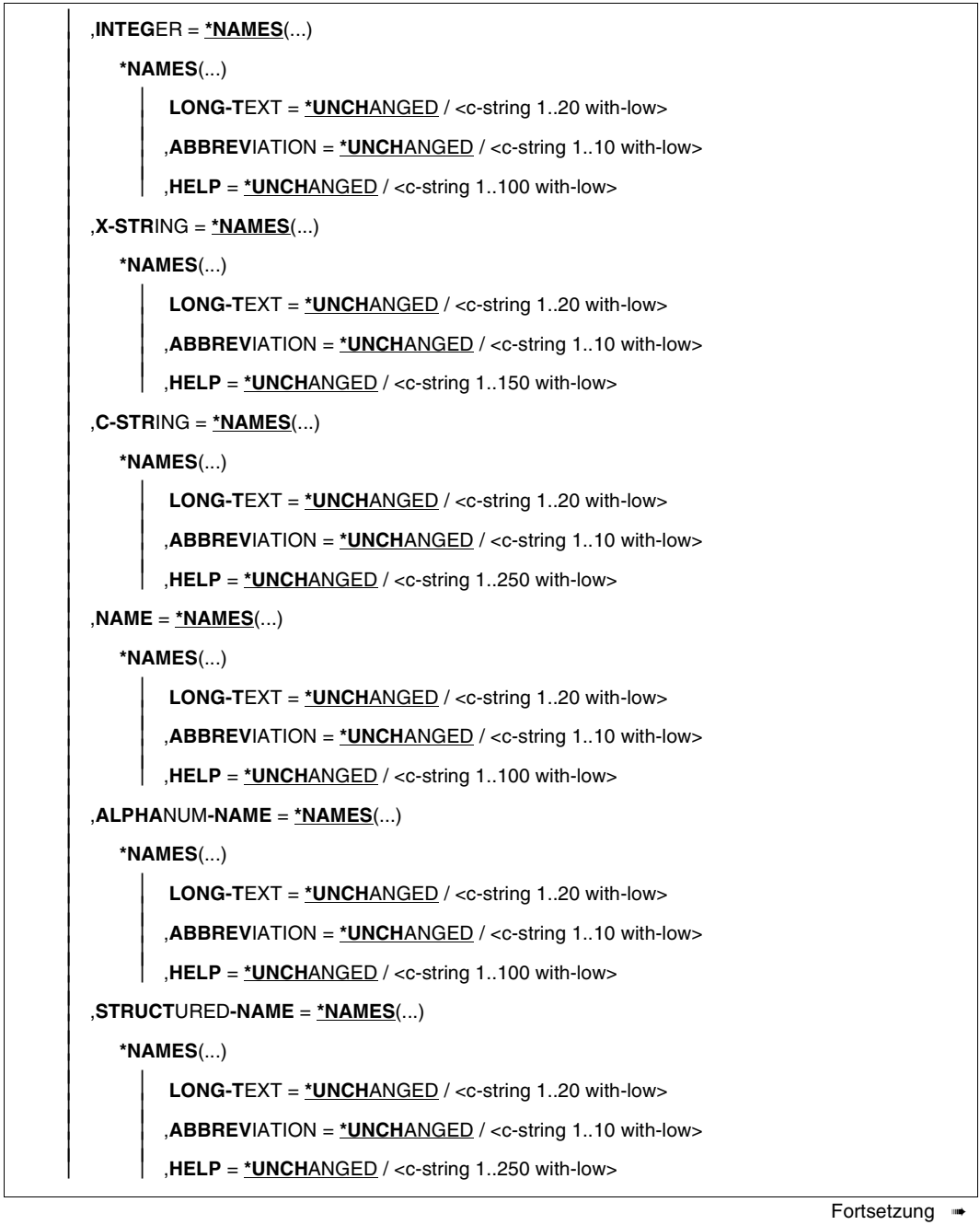
LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..100 with-low>

Fortsetzung ➠

(Teil 2 von 11)



Fortsetzung ➠

(Teil 3 von 11)

,**LABEL** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..150 with-low>

,**VSN** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**FILENAME** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..700 with-low>

,**PAR**TIAL**-FILENAME** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..400 with-low>

,**TIME** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**DATE** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

Fortsetzung ➠

(Teil 4 von 11)

,**COMPOSED-NAME** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..350 with-low>

,**TEXT** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..350 with-low>

,**CAT-ID** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..100 with-low>

,**PROD**UCT-**VERS**ION = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**POS**IX-**PATH**NAME = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..20 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**POS**IX-**FILENAME** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..20 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

Fortsetzung ➠

(Teil 5 von 11)

,**AMBIG**UOUS-**OPERAT**IONS = ***UNCH**ANGED / <c-string 1..50>

,**X-TEXT** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..150 with-low>

,**FIXED** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**WILDCARD** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**LOW**ER-**C**ASE = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**USER-ID** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**GEN**ERATION = ***NAMES**(...)

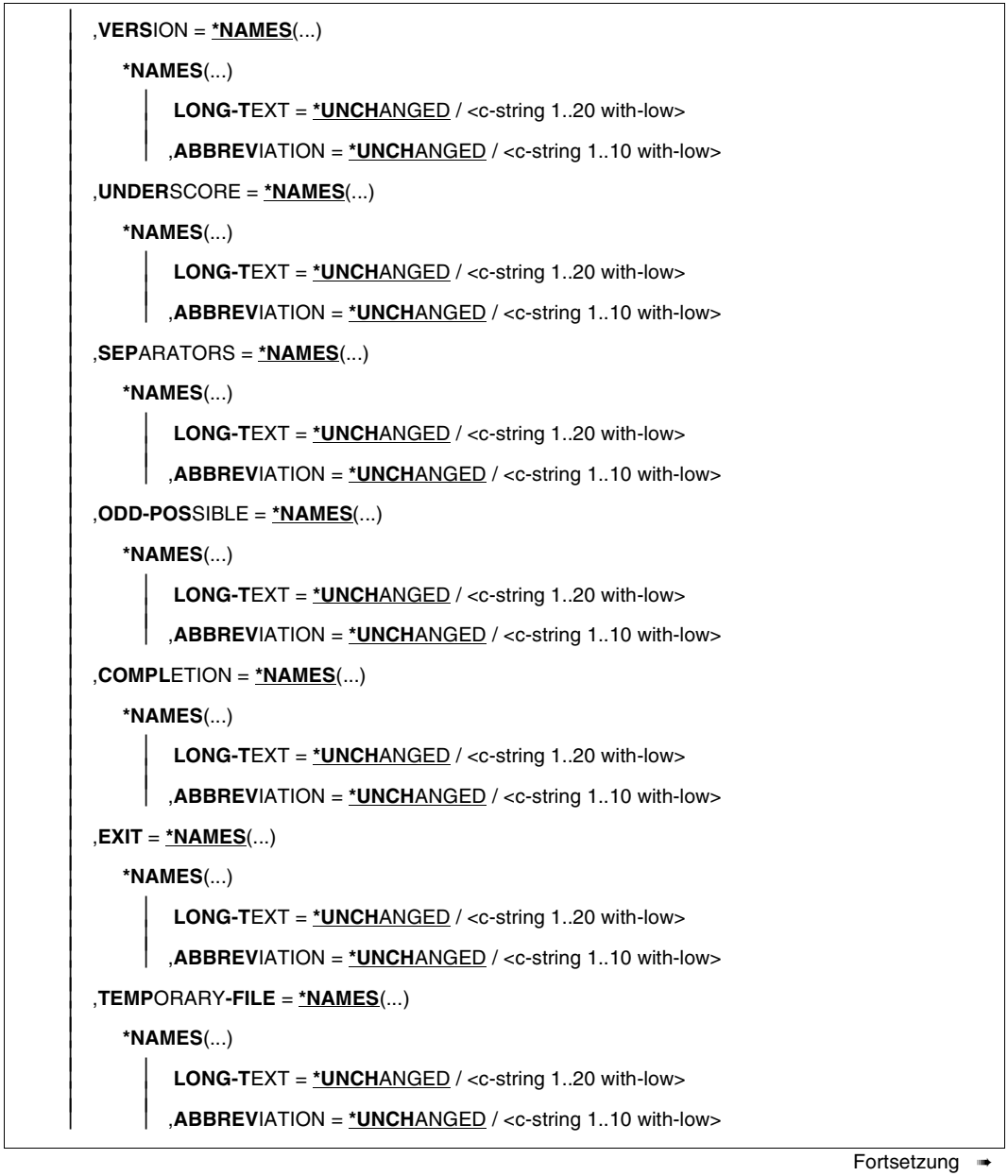
***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

Fortsetzung ➠

(Teil 6 von 11)



Fortsetzung ➠

(Teil 7 von 11)

,**QUOTES-MAND**ATORY = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..11 with-low>

,**LONG**EST-**LOG**ICAL-**LEN** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**USER-INT**ERFACE = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**CORR**ECTION-**STA**TE = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**PATH-COMPL**ETION = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**WILDC**ARD-**CONSTR**UCT = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..20 with-low>

,**REFRESH** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

Fortsetzung ➠

(Teil 8 von 11)

,**REST-SDF-IN** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..11 with-low>

,**EXIT-ALL** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**DOM**AIN = ***UNCH**ANGED / <c-string 1..11>

,**COM**MAND = ***UNCH**ANGED / <c-string 1..11>

,**PROG**RAM = ***UNCH**ANGED / <c-string 1..11>

,**STATEM**ENT = ***UNCH**ANGED / <c-string 1..13>

,**KEYS** = ***UNCH**ANGED / <c-string 1..13>

,**STRUCT**URE = ***UNCH**ANGED / <c-string 1..11>

,**SITUAT**ION = ***UNCH**ANGED / <c-string 1..11>

,**COMMENT-LINE** = ***UNCH**ANGED / <c-string 1..50>

,**PROC**EDURE**-ERROR** = ***UNCH**ANGED / <c-string 1..60>

,**INTERN**AL**-ERROR** = ***UNCH**ANGED / <c-string 1..60>

,**PROC-HELP** = ***UNCH**ANGED / <c-string 1..80>

,**INTERN**AL**-HELP** = ***UNCH**ANGED / <c-string 1..80>

,**CORR**ECT**-COM**MAND = ***UNCH**ANGED / <c-string 1..80>

,**CORR**ECT**-STATEM**ENT = ***UNCH**ANGED / <c-string 1..80>

,**OPER**ANDS = ***UNCH**ANGED / <c-string 1..11>

,**DOM**AIN**-TITLE** = ***UNCH**ANGED / <c-string 1..80>

,**COM**MAND**-TITLE** = ***UNCH**ANGED / <c-string 1..80>

,**STATEM**ENT**-TITLE** = ***UNCH**ANGED / <c-string 1..80>

,**DIR**ECT**-EXEC**UTION = ***UNCH**ANGED / <c-string 1..27>

,**OR** = ***UNCH**ANGED / <c-string 1..10 with-low>

,**DEF**AULT = ***UNCH**ANGED / <c-string 1..10 with-low>

,**DEFAULT-BY-JV** = ***UNCH**ANGED / <c-string 1..20 with-low>

Fortsetzung ➠

(Teil 9 von 11)

,**DEFAULT-BY-VAR** = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ALTER**NATE-**DEF**AULT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**MAND**ATORY = ***UNCH**ANGED / <c-string 1..20 with-low>

,**WITH** = ***UNCH**ANGED / <c-string 1..10 with-low>

,**WITHOUT** = ***UNCH**ANGED / <c-string 1..10 with-low>

,**LIST-POSSIBLE** = ***UNCH**ANGED / <c-string 1..20 with-low>

,**STRUCT**URE-**INCL**UDED = ***UNCH**ANGED / <c-string 1..20>

,**NEXT** = ***UNCH**ANGED / <c-string 1..10>

,**MESS**AGE = ***UNCH**ANGED / <c-string 1..10>

,**ERROR** = ***UNCH**ANGED / <c-string 1..10>

,**NUMB**ER = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**NEXT-CMD** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**NEXT-STMT** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**NEXT-DATA** = ***NAMES**(...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**NEXT-IN**PUT = ***NAMES**(...)

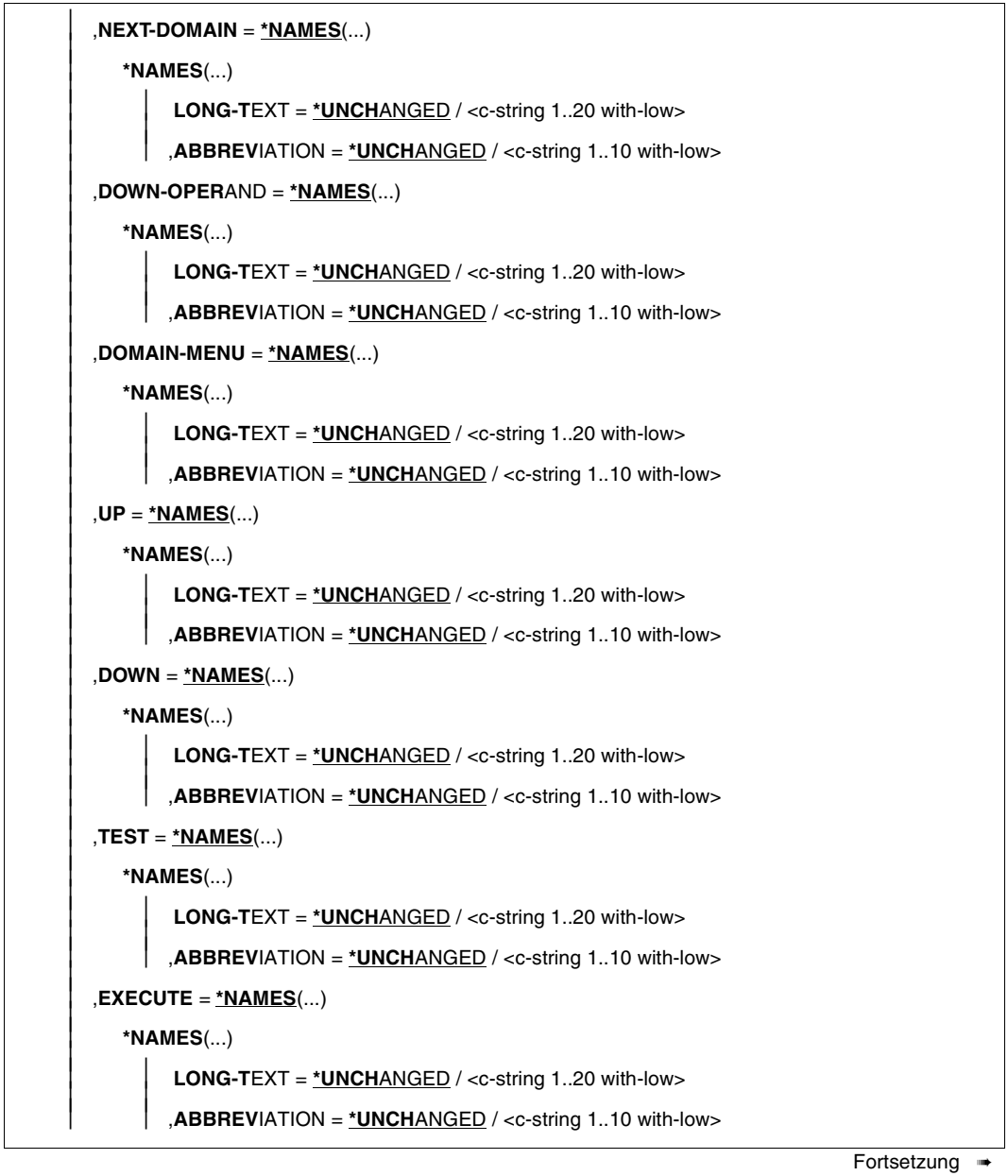
***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

Fortsetzung ➠

(Teil 10 von 11)



Fortsetzung ➠

(Teil 11 von 11)

```
,CONTINUE = *NAMES(...)
    *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
        ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,CANCEL = *NAMES(...)
    *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
        ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,RESTORE = *NAMES(...)
    *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
        ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,ENTER-COMMAND = *UNCHANGED / <c-string 1..80>
,ENTER-STATEMENT = *UNCHANGED / <c-string 1..80>
,ENTER-CONTINUATION = *UNCHANGED / <c-string 1..80>
,ENTER-OPERANDS = *UNCHANGED / <c-string 1..80>
,CORRECT-OPERATION = *UNCHANGED / <c-string 1..80>
,SECRET-OPERAND = *UNCHANGED / <c-string 1..80>
,PROC-INTERRUPT = *UNCHANGED / <c-string 1..80>
,PROC-INT-YES = *UNCHANGED / <c-string 1..10>
,PROC-INT-NO = *UNCHANGED / <c-string 1..10>
,REMOVE-LANGUAGE-TEXT = *NO / list-poss(2000): <name 1..1>
```

VERSION =
Legt Versionsnummer/-namen der Syntaxdatei fest. Beides dient nur zu Dokumentationszwecken.

VERSION = *UNCHANGED
Die Versionsnummer bleibt unverändert.

VERSION = <alphanum-name 1..12> / <c-string 1..12>
Die Syntaxdatei erhält die angegebene Versionsnummer bzw. den angegebenen Namen als Versionsnamen. Leerzeichen am Ende eines c-string werden nicht berücksichtigt.

GUIDANCE =
Bestimmt den Umfang der Benutzerführung. (Diese Festlegung gilt nicht für Aufträge, die über OMNIS gestartet wurden; für diese wird grundsätzlich *EXPERT angenommen.)

GUIDANCE = *UNCHANGED
Die in der Globalinformation stehende Festlegung über die Benutzerführung bleibt unverändert.

GUIDANCE = *STD
Die Benutzerführung bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei hat *STD die gleiche Wirkung wie *NO.

GUIDANCE = *MAXIMUM
Zur Auswahl von Anwendungsbereichen, Kommandos und Anweisungen werden Menüs mit Erläuterungstexten ausgegeben. Zur Operandenbesetzung werden Fragebögen ausgegeben. Für alle Strukturen gibt es eigene Fragebögen. Die Fragebögen enthalten Hilfetexte zu den Operanden, die Default-Werte, alle erlaubten Operandenwerte und Zusatzinformationen zu diesen.

GUIDANCE = *MEDIUM
Zur Auswahl von Anwendungsbereichen, Kommandos und Anweisungen werden Menüs mit Erläuterungstexten ausgegeben. Zur Operandenbesetzung werden Fragebögen ausgegeben. Für Strukturen, deren einleitender Wert mit SIZE=*LARGE (siehe ADD-VALUE) definiert ist, gibt es eigene Fragebögen. Die Fragebögen enthalten die Default-Werte und alle erlaubten Operandenwerte.

GUIDANCE = *MINIMUM
Zur Auswahl von Anwendungsbereichen, Kommandos und Anweisungen werden Menüs ausgegeben. Zur Operandenbesetzung werden Fragebögen ausgegeben. Für Strukturen, deren einleitender Wert mit SIZE=*LARGE (siehe ADD-VALUE) definiert ist, gibt es eigene Fragebögen. Die Fragebögen enthalten nur die Default-Werte.

GUIDANCE = *NO
Mit dem Text %CMD: oder %STMT: wird die Eingabe angefordert. Mehrere Kommandos können, getrennt durch ein „logisches Zeilenende“, in einem Block hintereinander eingegeben werden. Bei fehlerhafter Eingabe werden Korrekturmöglichkeiten geboten.

GUIDANCE = *EXPERT
Mit / oder // wird die Eingabe angefordert. Mehrere Kommandos können, getrennt durch ein „logisches Zeilenende“, in einem Block hintereinander eingegeben werden. Korrekturmöglichkeiten werden bei fehlerhafter Eingabe nicht angeboten.

LOGGING =
Bestimmt, wie die Eingaben protokolliert werden.

LOGGING = *UNCHANGED
Die in der Globalinformation stehende Festlegung über die Protokollierung bleibt unverändert.

LOGGING = *STD
Die Protokollierung bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei hat *STD die gleiche Wirkung wie *INPUT-FORM.

LOGGING = *INPUT-FORM
Im ungeführten Dialog werden die Eingabezeichenfolgen exakt protokolliert. Kennwörter werden ausgeblendet. Im geführten Dialog oder beim Fehlerdialog wird wie bei *ACCEPTED-FORM protokolliert.

LOGGING = *ACCEPTED-FORM
Protokolliert werden:
– alle Namen in ihrer Langform
– jeder in der Eingabe vorkommende Operand mit seinem Namen und dem angegebenen Wert
– die ggf. durch Korrekturen erzeugten Enddarstellungen.

Kennwörter sind ausgeblendet. Eingaben im geführten Dialog sind zu einem String verkettet.

LOGGING = *INVARIANT-FORM
Protokolliert werden:
– alle Namen in der Fassung, die in der Syntaxdatei als STANDARD-NAME definiert ist (d.h. die in den Handbüchern angegebenen Namen)
– jeder in der Eingabe vorkommende Operand mit seinem Namen und dem angegebenen Wert
– alle in der Eingabe implizit enthaltenen optionalen Operanden mit ihren Default-Werten
– die ggf. durch Korrekturen erzeugten Enddarstellungen

Kennwörter sind ausgeblendet. Eingaben im geführten Dialog sind zu einem String verkettet.

PROCEDURE-DIALOG =
Bestimmt, ob der Benutzer zur interaktiven Korrektur aufgefordert werden soll, wenn bei einer im Dialogbetrieb ablaufenden Prozedur oder SYSSTMT-Datei Syntax- oder Semantikfehler auftreten.

PROCEDURE-DIALOG = *UNCHANGED
Die in der Globalinformation stehende Festlegung über die interaktive Korrektur bleibt unverändert.

PROCEDURE-DIALOG = *STD
Die Regelung über die interaktive Korrektur bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei hat *STD die gleiche Wirkung wie *NO.

PROCEDURE-DIALOG = *YES
Der Benutzer wird zur interaktiven Korrektur aufgefordert.

PROCEDURE-DIALOG = *NO
Der Benutzer wird zur interaktiven Korrektur nicht aufgefordert.

UTILITY-INTERFACE =
setzt einen Schalter, der von einem Programm über den Makroaufruf CMDSTA abgefragt werden kann. Über diesen Schalter lässt sich bei Programmen, die ihre Anweisungen sowohl mit RDATA als auch über SDF mit CMDRST lesen können, die Art der Anweisungseingabe steuern.

UTILITY-INTERFACE = *UNCHANGED
Die in der Globalinformation stehende Festlegung über den Schalter bleibt unverändert.

UTILITY-INTERFACE = *STD
Der Schalter bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei hat *STD die gleiche Wirkung wie *NEW.

UTILITY-INTERFACE = *OLD
Programme sollen ihre Anweisungen mit RDATA lesen.

UTILITY-INTERFACE = *NEW
Programme sollen ihre Anweisungen über SDF mit CMDRST lesen.

CONTINUATION =
Bestimmt, in welcher Spalte bei der Kommandoeingabe (SYSCMD) gegebenenfalls das Fortsetzungszeichen „-“ anzugeben ist. (Bei Anweisungseingabe (SYSSTMT) kann das Fortsetzungszeichen in einer beliebigen Spalte angegeben werden.)

CONTINUATION = *UNCHANGED
Die in der Globalinformation stehende Festlegung über das Fortsetzungszeichen bleibt unverändert.

CONTINUATION = *STD
Die Regelung über das Fortsetzungszeichen bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei bewirkt *STD, dass die bei der Systemgenerierung getroffene Festlegung gilt.

CONTINUATION = *OLD
Das Fortsetzungszeichen muss in Spalte 72 angegeben werden.

CONTINUATION = *NEW
Das Fortsetzungszeichen kann in den Spalten 2 bis 72 angegeben werden.

FUNCTION-KEYS =
Bestimmt die Funktionstastenbelegung. Die verschiedenen Modi sind ausführlich im Benutzerhandbuch „Einführung in die Dialogschnittstelle SDF“ [1] beschrieben. Nicht unterstützte Funktionstasten wirken nicht wie DUE, sondern bewirken die leere Leistung.

FUNCTION-KEYS = *UNCHANGED
Die in der Globalinformation stehende Festlegung über die Funktionstastenbelegung bleibt unverändert. Nicht unterstützte Funktionstasten wirken nicht wie DUE, sondern bewirken die leere Leistung.

FUNCTION-KEYS = *STD
Der Schalter bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei wirkt *STD wie *BY-TERMINAL-TYPE.

FUNCTION-KEYS = *OLD-MODE
Die Belegung der Funktionstasten erfolgt im bisherigen Modus, der von allen Terminaltypen unterstützt wird. In diesem Modus gelten folgende Tastenbelegungen:

- K1 Exit-Funktion
- K2 Unterbrechungs-Funktion
- K3 Refresh-Funktion (nur im geführten Dialog)
- F1 Exit-all-Funktion
- F2 Test-Funktion (nur im geführten Dialog)
- F3 Execute-Funktion (nur im geführten Dialog)

FUNCTION-KEYS = *STYLE-GUIDE-MODE
Die Belegung der Funktionstasten entspricht dem Fujitsu-Siemens-Styleguide. In diesem Modus gelten folgende Tastenbelegungen:

- K2 Unterbrechungs-Funktion
- F1 Hilfe-Funktion
- F3 Exit-Funktion
- F5 Refresh-Funktion (nur im geführten Dialog)
- F6 Exit-all-Funktion
- F7 Rückwärts blättern (nur im geführten Dialog)
- F8 Vorwärts blättern (nur im geführten Dialog)
- F9 RESTORE-SDF-INPUT INPUT=*LAST ausführen
- F11 Execute-Funktion (nur im geführten Dialog)
- F12 Cancel-Funktion

FUNCTION-KEYS = *BY-TERMINAL-TYPE
Die Belegung der Funktionstasten ist abhängig von dem Terminaltyp. Unterstützt der Terminaltyp die umfangreichere Funktionalität des Fujitsu-Siemens-Styleguide, wählt SDF die Einstellung *STYLE-GUIDE-MODE. Anderenfalls wählt SDF die Einstellung *OLD-MODE.

Bei der Einstellung wird ausgewertet, mit welchem Typ das Terminal im System generiert wurde. Weicht der generierte Terminaltyp vom tatsächlichen Terminaltyp ab, ist nicht gewährleistet, dass die Einstellung dem tatsächlich unterstützten Funktionsumfang entspricht. Bei einer Terminalemulation kann der erkannte Terminaltyp sowohl von der Generierung als auch von Umgebungsvariablen abhängen. Genaueres ist der Beschreibung des Emulationsprogrammes zu entnehmen.

INPUT-HISTORY =
Bestimmt, ob der Eingabepuffer eingeschaltet, ausgeschaltet oder zurückgesetzt wird.

INPUT-HISTORY = *UNCHANGED
Die in der Globalinformation stehende Festlegung über die Speicherung der Eingaben bleibt unverändert.

INPUT-HISTORY = *STD
Der Schalter bleibt unverändert, wenn die bearbeitete Benutzer- oder Gruppensyntaxdatei aktiviert wird. Für eine Systemsyntaxdatei hat *STD die gleiche Wirkung wie *ON.

INPUT-HISTORY = *ON
Der Eingabepuffer wird eingeschaltet. SDF speichert syntaktisch richtige Eingaben im Eingabepuffer. Nicht gespeichert werden SET-LOGON-PARAMETERS, RESTORE-SDF-INPUT, SHOW-INPUT-HISTORY und ISP-Kommandos.
Mit der Anweisung SHOW-INPUT-HISTORY kann sich der Benutzer die gespeicherten Eingaben ausgeben lassen. Mit der Anweisung RESTORE-SDF-INPUT kann der Benutzer eine bestimmte Eingabe ausgeben lassen, um sie dann unverändert oder auch modifiziert erneut abzuschicken.

Wertangaben für „geheime“ Operanden, die weder dem Default-Wert noch einem mit SECRET=*NO definierten Wert entsprechen, werden im Eingabepuffer mit ^ gespeichert. Wertangaben zu nicht „geheimen“ Operanden werden im Eingabepuffer im Klartext gespeichert. Im Einzelfall können diese Eingaben aus Benutzersicht auch schützenswerte Informationen darstellen (z.B. Prozedurparameter). Soll verhindert werden, dass diese Eingaben mit SHOW-INPUT-HISTORY bzw. RESTORE-SDF-INPUT erneut am Bildschirm sichtbar gemacht werden können, so kann der Benutzer wie folgt vorgehen: Vor sicherheitsrelevanten Eingaben den Eingabepuffer ausschalten und danach wieder einschalten. Wurden die Eingaben bereits gespeichert, kann der Eingabepuffer mit *RESET zurückgesetzt werden. In diesem Fall werden jedoch alle gespeicherten Eingaben gelöscht.

INPUT-HISTORY = *OFF
Der Eingabepuffer wird ausgeschaltet. Die nachfolgenden Eingaben werden nicht gespeichert. Alle bisher gespeicherten Eingaben sind jedoch weiterhin zugreifbar.

NUMBER-OF-INPUTS = *UNCHANGED / *STD / <integer 1..100>
gehört zum Operanden INPUT-HISTORY und legt fest, wie viele Eingaben in dem Eingabepuffer gespeichert werden können. Die obere Grenze liegt bei 100. Bei Erreichen der maximalen Anzahl gespeicherter Eingaben wird die jeweils älteste Eingabe gelöscht.

GENERAL-INFO-VERSION = *UNCHANGED / <integer 1..255>
Dieser Operand ist für die Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert und wird hier nicht beschrieben.

MODIFY-LANGUAGE-TEXT =
Die Globalinformation enthält die von SDF bei der Dialogführung benutzten Texte. Diese Texte können für mehrere Sprachen definiert sein. Welche Sprache SDF bevorzugt benutzt, wird bei der Systemgenerierung zusammen mit der Sprache für die Meldungsausgabe festgelegt.

MODIFY-LANGUAGE-TEXT = *UNCHANGED
Die Texte bleiben unverändert.

MODIFY-LANGUAGE-TEXT = list-poss(2000): <name 1..1>(...)
Die Texte für die angegebenen Sprachschlüssel <name> (E=Englisch, D=Deutsch) werden geändert. Existieren für einen angegebenen Sprachschlüssel noch keine Texte, so wird für ihn durch Kopieren der ersten vorhandenen Sprache ein Satz Texte neu angelegt. Diese Texte werden dann anschließend geändert. Für bestimmte Texte gibt es eine Langform (LONG-TEXT) und eine Kurzform (ABBREVIATION). Die Langform benutzt SDF in der MAXIMUM-Stufe des geführten Dialogs, die Kurzform in der MEDIUM- und MINIMUM-Stufe.
Ab SDF/SDF-A Version 2.0 kann ein Hilfetext auch für die vordefinierten SDF-Datentypen erstellt werden. Diese Hilfetexte werden im geführten Dialog bei Eingabe von ’??’ ausgegeben. Sie erhalten im geführten Dialog den Operandenfragebogen für SET-GLOBALS mit den vorbelegten Texten, wenn Sie vor dem Aufruf der Anweisung SET-GLOBALS mit der Anweisung EDIT auf GLOBAL-INFORMATION positionieren. Ein Beispiel (siehe Seite 337 bis 347) demonstriert die Vorbelegung der Texte und die Funktionalität der Anweisung SET-GLOBALS beim Ändern von Texten.

REMOVE-LANGUAGE-TEXT =
Bestimmt, ob die Texte für die angegebenen Sprachen gelöscht werden.

REMOVE-LANGUAGE-TEXT = *NO
Es werden keine Texte gelöscht.

REMOVE-LANGUAGE-TEXT = list-poss(2000): <name 1..1>
Die Texte für die angegebenen Sprachschlüssel werden gelöscht.

Vorbelegung der sprachabhängigen Texte

Jede Syntaxdatei, die neu angelegt wird, bekommt eine vordefinierte Globalinformation, die mit SET-GLOBALS verändert werden kann. Die sprachabhängigen Texte, die beim Operanden MODIFY-LANGUAGE-TEXT angegeben werden, sind mit Initialwerten vorbelegt. Die sprachabhängigen Texte lassen sich in folgende Kategorien einteilen:

- Datentypen: Sie werden in den Hilfetexten und Fehlermeldungen zu Operandenwerten verwendet und sind wie folgt vorbelegt:

Datentyp	**MODIFY-LANGUAGE-TEXT=E(...**	
	***NAMES(LONG-TEXT=...**	**,ABBREVIATION=...)**
COMMAND-REST	'command-rest'	'cmdrest'
INTEGER	'integer'	'integer'
X-STRING	'x-string'	'x-string'
C-STRING	'c-string'	'c-string'
NAME	'name'	'name'
ALPHANUM-NAME	'alphanum-name'	'aln-name'
STRUCTURED-NAME	'structured-name'	'struc-name'
LABEL	'label'	'label'
VSN	'vsn'	'vsn'
FILENAME	'filename'	'filename'
PARTIAL-FILENAME	'partial-name'	'p-filename'
TIME	'time'	'time'
DATE	'date'	'date'
COMPOSED-NAME	'composed-name'	'comp-name'
TEXT	'text'	'text'
CAT-ID	'cat-id'	'cat-id'
RODUCT-VERSION	'product-version'	'prod-ver'
POSIX-PATHNAME	'posix-pathame'	'posix-pathname'
POSIX-FILENAME	'posix-filename'	'posix-filename'
X-TEXT	'x-text'	'x-text'
FIXED	'fixed'	'fixed'

- Zusätze zu Datentypen (Datentyp-Attribute): Sie werden in den Hilfetexten und Fehlermeldungen zu Operandenwerten verwendet und sind wie folgt vorbelegt:

Datentyp-Attribut	**MODIFY-LANGUAGE-TEXT=E(...**	
	***NAMES(LONG-TEXT=...**	**,ABBREVIATION=...)**
WILDCARDS	'wildcards'	'wildcards'
LOWER-CASE	'lower-case'	'lower-case'
USER-ID	'user-id'	'user-id'
GENERATION	'generation'	'gen'
VERSION	'version'	'version'
UNDERSCORE	'underscore'	'under'
SEPARATORS	'separators'	'separ'
ODD-POSSIBLE	'odd-possible'	'odd-poss'
COMPLETION	'completion'	'completion'
LONGEST-LOGICAL-LEN	'longest-logical-len'	'long-log'
TEMPORARY-FILE	'temporary-file'	'temp-file'
QUOTES-MANDATORY	'quotes'	'quotes'
USER-INTERFACE	'manual-release'	'man-rel'
CORRECTION-STATE	'correction-state'	'corr-state'
PATH-COMPLETION	'path-completion'	'path-compl'
WILDCARD-CONSTRUCT	'wildcard-constr'	'w-constr'

- Bezeichner im Kopffeld der Kommando- und Anweisungsmenüs sowie in den Operandenfragebögen sind wie folgt vorbelegt:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
DOMAIN	'DOMAIN'
COMMAND	'COMMAND'
PROGRAM	'PROGRAM'
STATEMENT	'STATEMENT'
KEYS	'KEYS'
STRUCTURE	'STRUCTURE'
SITUATION	'SITUATION'
COMMENT-LINE	'COMMENT'
AMBIGUOUS-OPERATIONS	'OPERATIONS CORRESPONDING TO'

- Bezeichner für verfügbare Objekte im unteren Teil der Menüs sind wie folgt vorbelegt:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
DOMAIN-TITLE	'AVAILABLE APPLICATION DOMAINS'
COMMAND-TITLE	'AVAILABLE COMMANDS'
STATEMENT-TITLE	'AVAILABLE STATEMENTS'

- Aufforderung zur sofortigen Ausführung: Diese Aufforderung wird nur im geführten Dialog (MAXIMUM-Stufe) und nur für Kommandos/Anweisungen angezeigt, die keine Operanden haben.

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
DIRECT-EXECUTION	'EXECUTED IMMEDIATLY'

- Operanden-Attribute: Sie werden bei Hilfetexten und Fehlermeldungen in den Operandenfragebögen ausgegeben und sind wie folgt vorbelegt:

Operanden-Attribut	**MODIFY-LANGUAGE-TEXT=E(...**
DEFAULT	'default'
DEFAULT-BY-JV	'default-by-JV'
DEFAULT-BY-VAR	'default-by-variable'
ALTERNATE-DEFAULT	'alternate-default'
MANDATORY	'mandatory'
OR	'or'
WITH	'with'
WITHOUT	'without'
LIST-POSSIBLE	'list-possible'

- Struktureinleiter: Für Struktureinleiter wird ausgegeben, dass noch ein Unterfragebogen dazu existiert. Der Bezeichner dafür ist wie folgt vorbelegt:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
STRUCTURE-INCLUDED	'INPUT TRANSFERRED'

- Bezeichner für Menüsteuerung und Fehler: Sie werden im unteren Bereich der Menüs ausgegeben und sind wie folgt vorbelegt:

  Folgeaktion/Meldungen/Fehler

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
NEXT MESSAGE ERROR	'NEXT' 'MESSAGE' 'ERROR'

  Syntaxerklärung für das NEXT-Feld

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**	
	***NAMES(LONG-TEXT=...**	**,ABBREVIATION=...)**
NUMBER	'Number'	'Number'
NEXT-CMD	'Next-command'	'Next-cmd'
NEXT-STMT	'Next-statement'	'Next-stmt'
NEXT-DATA	'Next-data'	'Next-data'
NEXT-INPUT	'Next-input'	'Next-input'
NEXT-DOMAIN	'Next-domain'	'Next-dom'
DOWN-OPERAND	'DOWN'	'DOWN'
DOMAIN-MENU	'DOMAIN-MENU'	'DOM-MENU'
UP	'UP'	'UP'
DOWN	'DOWN'	'DOWN'
TEST	'TEST'	'TEST'
EXECUTE	'EXECUTE'	'EXECUTE'
CONTINUE	'CONTINUE'	'CONTINUE'
CANCEL	'CANCEL'	'CANCEL'
RESTORE	'RESTORE'	'RESTORE'
EXIT	'EXIT'	'EXIT'
REFRESH	'REFRESH'	'REFRESH'
REST-SDF-IN	'REST-SDF-IN'	'REST-SDF-IN'
EXIT-ALL	'EXIT-ALL'	'EXIT-ALL'

- Texte für den ungeführten Dialog sind wie folgt vorbelegt:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
SECRET-OPERAND	'ENTER SECRET OPERAND'

  Die Eingabezeile für den geheimen Operanden wird dunkelgesteuert.

Im Dialogmodus GUIDANCE=*NO werden verschiedene Eingabeaufforderungen ausgegeben, die wie folgt vorbelegt sind:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
ENTER-COMMAND	'CMD'
ENTER-STATEMENT	'STMT'
ENTER-CONTINUATION	'ENTER CONTINUATION'
ENTER-OPERANDS	'ENTER OPERANDS'
CORRECT-OPERATION	'CORRECT OPERATIONNAME'
CORRECT-COMMAND	'CORRECT CMD'
CORRECT-STATEMENT	'CORRECT STMT'
OPERANDS	'OPERANDS'

- Prozedur-Fehlermeldungen im geführten Dialog. Sie erscheinen im Kopffeld der Menüs und geben die Ursache für den Dialog an:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
PROCEDURE-ERROR	'ERROR IN PROCEDURE-CMD/STMT'
PROC-HELP	'DIALOG IN PROCEDURE'

- Programm-Fehlermeldungen im geführten Dialog. Sie erscheinen im Kopffeld der Menüs und geben die Ursache für den Korrekturdialog an:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
INTERNAL-ERROR	'ERROR IN PROG/S-PROC'
INTERNAL-HELP	'DIALOG IN PROG/S-PROC'

- Eingabeaufforderungen bei Prozedurunterbrechung sind wie folgt vorbelegt:

Bezeichner	**MODIFY-LANGUAGE-TEXT=E(...**
PROC-INTERRUPT	'NEXT-DATA'
PROC-INT-YES	'YES'
PROC-INT-NO	'NO'

Beispiel

Im folgenden Beispiel wird eine Benutzersyntaxdatei erzeugt und einige sprachabhängige Texte in der Globalinformation werden geändert. In den Kommandomenüs und im Operandenfragebogen sind die zu ändernden bzw. nachher geänderten Texte halbfett dargestellt.

```
/start-sdf-a
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file file=syssdf.user.globals,type=*user,mode=*create ——— (1)
//set-globals mod-lang-text=e(generation=names('gen','g'),-
                              domain-title='list of domains',-
                              command-title='list of commands',- ——— (2)
                              without='no',-
                              next='select',-
                              number=names('#','#'))
//end
/modify-sdf-options guid=*max,funct-keys=*style-guide-mode ——————— (3)
```

```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

  1  ACCOUNTING                    : Output of informations about the user
                                     identification and introduction of data
                                     into the accounting record
  2  ALL-COMMANDS                  : Output of all command names in alphabetic
                                     order
  3  CONSOLE-MANAGEMENT            : Control of operator console/terminal
  4  DATABASE                      : Management and administration of databases
  5  DCAM                          : Control of transaction-driven system (DCAM)
  6  DCE                           : Management of DCE (Distributed Computing
                                     Environment)
  7  DEVICE                        : Information about devices and volumes
  8  FILE                          : Management of files
  9  FILE-GENERATION-GROUP         : Management of file generation groups
--------------------------------------------------------------------------------
NEXT = +
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN
```

——— (4)

1. Nach dem Aufruf von SDF-A wird eine Benutzersyntaxdatei mit dem Namen SYSSDF.USER.GLOBALS erzeugt und geöffnet.
2. Einige englische Texte, die im geführten Dialog erscheinen, werden geändert.

3. Nach dem Beenden von SDF-A und dem damit verbundenen Abspeichern der Benutzersyntaxdatei wird zunächst in den geführten Dialog gewechselt. Die Syntaxdatei USER.GLOBALS ist noch nicht aktiviert. Deshalb erscheinen in den Menüs die von SDF vorbelegten Texte.
4. Das Menü der Anwendungsbereiche erscheint. Darin sind die Texte halbfett dargestellt, die geändert werden sollen. Mit der Taste DUE (Vorbelegung '+' in der NEXT-Zeile) bzw. F8 wird weitergeblättert.

```
------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 10  FILE-TRANSFER                 : Transfer of files between computers
                                     through the network
 11  IDIAS
 12  IDOM
 13  JOB                           : Job control
 14  JOB-VARIABLES                 : Management of Job variables
 15  MESSAGE-PROCESSING            : Management of message files
 16  MULTI-CATALOG-AND-PUBSET-MGMT : Control of file accesses to local area
                                     network
 17  NETWORK-MANAGEMENT            : Control of DCM applications and connections
 18  PROCEDURE                     : Control of the command procedures
 19  PROGRAM                       : Control of program flow
 20  PROGRAMMING-SUPPORT           : Start of compilers and programming tools
------------------------------------------------------------------------------
NEXT = 13
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F8=+   F9=REST-SDF-IN
```

(5)

5. Durch Eingabe der Ziffer 13 und DUE wird in den Anwendungsbereich JOB gewechselt.

```
DOMAIN   : JOB
-------------------------------------------------------------------------------
AVAILABLE COMMANDS:

 11  ENTER-JOB                     : Initiates a command sequence, stored in an
                                     ENTER file, as a batch job
 12  EOF                           : Generates an end of file  for the
                                     currently active system input file SYSDTA
     (EXECUTED IMMEDIATELY!)
 13  EXIT-JOB                      : Terminates the currently executing task
 14  HELP-MSG-INFORMATION          : Displays a system message
 15  INFORM-OPERATOR               : Sends a message to a console
 16  INFORM-PROGRAM                : Sends a message to an program (STXIT
                                     routine)
 17  LOGOFF                        : Terminates the currently executing task
 18  MODIFY-JOB                    : Modifies job attributes which were defined
                                     for a batch job in the ENTER-JOB command
-------------------------------------------------------------------------------
NEXT = 11
       Number / Next-command / (Next-domain) / *DOMAIN-MENU
KEYS : F3=*EXIT F5=*REFRESH F6=*EXIT-ALL F7=- F8=+ F9=REST-SDF-IN F12=*CANCEL
```

(6)

6. Das Kommandomenü für den Anwendungsbereich JOB erscheint. Mit der Taste DUE (Vorbelegung '+' in der NEXT-Zeile) bzw. F8 wurde bereits einmal weitergeblättert (um das Kommando ENTER-JOB zu sehen). Im Kommandomenü sind wieder die Texte halbfett dargestellt, die geändert werden sollen. Durch Eingabe der Ziffer 11 wird in den Operandenfragebogen des Kommandos ENTER-JOB gewechselt.

```
DOMAIN   : JOB                              COMMAND: ENTER-JOB


-------------------------------------------------------------------------------
FROM-FILE            =
                       *LIBRARY-ELEMENT() or filename_1..54_without-generation
                       Specifies the name of the ENTER file
PROCESSING-ADMISSION = *STD
                       *STD or *PARAMETERS()
                       Specifies the user ID under which the batch job is to be
                       executed
FILE-PASSWORD        =
                       *NONE or c-string_1..4 or x-string_1..8
                       Specifies the write or execute password protecting the
                       ENTER file


-------------------------------------------------------------------------------
NEXT = F6
       Next-command / (Next-domain) / *CONTINUE / *DOMAIN-MENU / *TEST
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

(7)

7. Der Operandenfragebogen des Kommandos ENTER-JOB erscheint.
   Beim Operanden FROM-FILE ist das Datentyp-Attribut **without-generation** hervorgehoben, das ebenfalls geändert werden soll. Um vergleichen zu können, soll nun die

Benutzersyntaxdatei SYSSDF.USER.GLOBALS aktiviert werden. Dazu wird zunächst durch Drücken der Taste F6 (alternativ Eingabe von *EXIT-ALL in der NEXT-Zeile und DUE) zum globalen Menü der Anwendungsbereiche zurückgegangen.

```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

  1  ACCOUNTING                   : Output of informations about the user
                                    identification and introduction of data
                                    into the accounting record
  2  ALL-COMMANDS                 : Output of all command names in alphabetic
                                    order
  3  CONSOLE-MANAGEMENT           : Control of operator console/terminal
  4  DATABASE                     : Management and administration of databases
  5  DCAM                         : Control of transaction-driven system (DCAM)
  6  DCE                          : Management of DCE (Distributed Computing
                                    Environment)
  7  DEVICE                       : Information about devices and volumes
  8  FILE                         : Management of files
  9  FILE-GENERATION-GROUP        : Management of file generation groups
--------------------------------------------------------------------------------
NEXT = mod-sdf-opt syntax-file=*add(syssdf.user.globals)
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN
```

(8)

8. Das globale Menü der Anwendungsbereiche erscheint. Die halbfett dargestellten Texte sind noch nicht geändert. Mit dem Kommando MODIFY-SDF-OPTIONS wird die Benutzersyntaxdatei SYSSDF.USER.GLOBALS aktiviert, woraufhin die Änderung der Texte sofort vollzogen wird.

```
------------------------------------------------------------------------------
LIST OF DOMAINS:

  1  ACCOUNTING                    : Output of informations about the user
                                     identification and introduction of data
                                     into the accounting record
  2  ALL-COMMANDS                  : Output of all command names in alphabetic
                                     order
  3  CONSOLE-MANAGEMENT            : Control of operator console/terminal
  4  DATABASE                      : Management and administration of databases
  5  DCAM                          : Control of transaction-driven system (DCAM)
  6  DCE                           : Management of DCE (Distributed Computing
                                     Environment)
  7  DEVICE                        : Information about devices and volumes
  8  FILE                          : Management of files
  9  FILE-GENERATION-GROUP         : Management of file generation groups
------------------------------------------------------------------------------
SELECT = 13
         # / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN
```

(9)

9. Sie befinden sich noch immer im Menü der Anwendungsbereiche, doch jetzt erscheinen die geänderten Texte (halbfett dargestellt). Es wird in den Anwendungsbereich JOB (Nummer 13) gewechselt.

```
DOMAIN   : JOB
------------------------------------------------------------------------------
LIST OF COMMANDS:

  1  ASSIGN-SYSDTA                 : Assigns SYSDTA to an  input source
  2  ASSIGN-SYSEVENT               : Assigns an event stream to the own user
                                     task
  3  ASSIGN-SYSIPT                 : Assigns SYSIPT to an  input source
  4  ASSIGN-SYSLST                 : Assigns  SYSLST to an output destination
  5  ASSIGN-SYSOPT                 : Assigns SYSOPT to an  output destination
  6  ASSIGN-SYSOUT                 : Assigns SYSOUT to an  output destination
  7  CANCEL-JOB                    : Cancel a dialog, batch or print job
  8  CHANGE-TASK-PRIORITY          : Changes the priority of a dialog, batch or
                                     print job
  9  COPY-SYSTEM-FILE              : Copies current system file  to an user file
 10  DELETE-SYSTEM-FILE            : Deletes a system file

------------------------------------------------------------------------------
SELECT = 11
         # / Next-command / (Next-domain) / *DOMAIN-MENU
KEYS : F3=*EXIT  F5=*REFRESH  F6=*EXIT-ALL  F8=+  F9=REST-SDF-IN  F12=*CANCEL
```

(10)

10. Das Menü des Anwendungsbereiches JOB erscheint. Sie erkennen die geänderten Texte an der halbfetten Darstellung. Durch Eingabe der Ziffer 11 wird in den Operandenfragebogen des Kommandos ENTER-JOB gewechselt.

```
DOMAIN   : JOB                              COMMAND: ENTER-JOB


------------------------------------------------------------------------------
FROM-FILE             =
                        *LIBRARY-ELEMENT() or filename_1..54_no-gen
                        Specifies the name of the ENTER file
PROCESSING-ADMISSION = *STD
                        *STD or *PARAMETERS()
                        Specifies the user ID under which the batch job is to be
                        executed
FILE-PASSWORD         =
                        *NONE or c-string_1..4 or x-string_1..8
                        Specifies the write or execute password protecting the
                        ENTER file


------------------------------------------------------------------------------
SELECT = F6
         Next-command / (Next-domain) / *CONTINUE / *DOMAIN-MENU / *TEST
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

(11)

11. Der Operandenfragebogen des Kommandos ENTER-JOB erscheint. Beim Operanden FROM-FILE sehen Sie, dass das Datentyp-Attribut **without-generation** durch **no-gen** ersetzt ist. Nach Drücken der Taste F6 (alternativ Eingabe von *EXIT-ALL in der NEXT-Zeile und DUE) erscheint wieder das globale Menü der Anwendungsbereiche.

SHOW
Objekte der Syntaxdatei ausgeben

Mit der Anweisung SHOW geben Sie Inhalte einer Syntaxdatei auf SYSOUT oder SYSLST aus. Die Ausgabe kann unterbrochen bzw. wieder gestartet oder mit K2 abgebrochen werden.

Die Ausgabe kann erneut als Eingabe dienen, um eine Syntaxdatei oder ein Syntaxdateiobjekt wiederherzustellen (siehe OBJECT=*ALL, IMPLEMENTATION=*YES, LINES-PER-PAGE=*UNLIMITED(OUTPUT=*FOR-INPUT)).

(Teil 1 von 4)

SHOW

```
OBJECT = *ALL / *GLOBAL-INFORMATION / *DOMAIN(...) / *COMMAND(...) / *PROGRAM(...) /
         *STATEMENT(...) / *PRIVILEGE(...) / *OPERAND(...) / *VALUE(...) /
         *CORRECTION-INFORMATION(...)

  *DOMAIN(...)
    |
    |  NAME = *ALL(...) / *NONE / <structured-name 1..30 with-wild> /
    |         list-poss(2000): <structured-name 1..30>
    |
    |    *ALL(...)
    |      |
    |      |  EXCEPT = *NONE / <structured-name 1..30 with-wild> /
    |      |           list-poss(2000): <structured-name 1..30>

  *COMMAND(...)
    |
    |  NAME = *ALL(...) / *REMOVED / *CURRENT / <structured-name 1..30 with-wild> /
    |         list-poss(2000): <structured-name 1..30>
    |
    |    *ALL(...)
    |      |
    |      |  EXCEPT = *NONE / <structured-name 1..30 with-wild> /
    |      |           list-poss(2000): <structured-name 1..30>

  *PROGRAM(...)
    |
    |  NAME = *ALL(...) / *REMOVED / *CURRENT / <structured-name 1..30 with-wild> /
    |         list-poss(2000): <structured-name 1..30>
    |
    |    *ALL(...)
    |      |
    |      |  EXCEPT = *NONE / <structured-name 1..30 with-wild> /
    |      |           list-poss(2000): <structured-name 1..30>
```

Fortsetzung ➠

(Teil 2 von 4)

***STATEM**ENT(...)

NAME = ***ALL**(...) / ***REM**OVED / ***CUR**RENT / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>

***ALL**(...)

EXCEPT = ***NONE** / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

***PRIV**ILEGE(...)

NAME = ***ALL** / <structured-name 1..30>

***OPER**AND(...)

OPERAND-**L1** = ***CUR**RENT / <structured-name 1..20>

,**VAL**UE-**L1** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***PAR**TIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** /
***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** /
***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** /
***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION /
***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

Fortsetzung

(Teil 3 von 4)

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

,**ORIG**IN = ***CUR**RENT / ***COM**MAND(...) / ***STATEM**ENT(...)

***COM**MAND(...)

NAME = <structured-name 1..30>

***STATEM**ENT(...)

NAME = <structured-name 1..30>

,**PROG**RAM = <structured-name 1..30>

***VAL**UE(...)

OPERAND-**L1** = ***ABOVE-CUR**RENT / <structured-name 1..20>

,**VAL**UE-**L1** = ***CUR**RENT / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** / ***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** / ***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** / ***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION / ***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L2** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L2** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** / ***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** / ***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** / ***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION / ***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L3** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L3** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** / ***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** / ***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** / ***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION / ***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L4** = ***NO** / <structured-name 1..20>

,**VAL**UE-**L4** = ***NO** / ***COM**MAND-**REST** / ***INTEG**ER / ***X-STR**ING / ***C-STR**ING / ***NAME** / ***ALPHA**NUMERIC-**NAME** / ***STRUCT**URED-**NAME** / ***FILENAME** / ***P**ARTIAL-**FILENAME** / ***TIME** / ***DATE** / ***TEXT** / ***CAT-ID** / ***LABEL** / ***VSN** / ***COMPOSED-NAME** / ***X-TEXT** / ***FIXED** / ***DEV**ICE / ***PROD**UCT-**VERS**ION / ***POS**IX-**PATH**NAME / ***POS**IX-**FILENAME** / <composed-name 1..30>

,**OPER**AND-**L5** = ***NO** / <structured-name 1..20>

Fortsetzung ➠

(Teil 4 von 4)

```
        ,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                    *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                    *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                    *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                    *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
        ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
            *COMMAND(...)
              | NAME = <structured-name 1..30>
            *STATEMENT(...)
              | NAME = <structured-name 1..30>
              | ,PROGRAM = <structured-name 1..30>
    *CORRECTION-INFORMATION(...)
      | PM-NUMBER = *ALL / list-poss(20): <alphanum-name 8..8>
,ATTACHED-INFORMATION = *YES / *NO / *IMMEDIATE
,SIZE = *MINIMUM / *MAXIMUM / *MEDIUM
,IMPLEMENTATION-INFO = *NO(...) / *YES
    *NO(...)
      | FORM = *UNGUIDED / *GUIDED
      | ,LANGUAGE = E / <name 1..1>
,LINE-LENGTH = *STD / <integer 72..132>
,LINES-PER-PAGE = *STD / *UNLIMITED(...) / <integer 1..200>
    *UNLIMITED(...)
      | OUTPUT-FORM = *STD / *FOR-INPUT
,OUTPUT = *SYSOUT / *SYSLST(...)
    *SYSLST(...)
      | SYSLST-NUMBER = *STD / <integer 1..99>
,PRIVILEGE = *ANY / list-poss(64): <structured-name 1..30>
```

OBJECT =
Art des Objekts, dessen Definition auszugeben ist.

OBJECT = *ALL
Bestimmt, dass der gesamte Inhalt einer Syntaxdatei ausgegeben wird.

OBJECT = *GLOBAL-INFORMATION
Bestimmt, dass die Globalinformation einer Syntaxdatei ausgegeben wird.

OBJECT = *DOMAIN(...)
Bestimmt, dass die Definitionen von Anwendungsbereichen ausgegeben werden.

NAME = *ALL(...)
Die Definitionen von allen Anwendungsbereichen werden ausgegeben.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Anwendungsbereiche werden nicht ausgegeben.

NAME = *NONE
Die Definitionen aller Kommandos, die keinem Anwendungsbereich zugeordnet sind, werden ausgegeben.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Anwendungsbereiche werden ausgegeben, bzw. die Definitionen der Anwendungsbereiche, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *COMMAND(...)
Bestimmt, dass die Definitionen von Kommandos ausgegeben werden.

NAME = *ALL(...)
Die Definitionen von allen Kommandos werden ausgegeben.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Kommandos werden nicht ausgegeben.

NAME = *REMOVED
Alle Kommandos, die entfernt sind (und die wiederhergestellt werden können, weil ihre Beschreibung auf einer höheren Stufe der Hierarchie vorhanden ist) werden ausgegeben.

NAME = *CURRENT
Die aktuellen Kommandos werden ausgegeben (sofern vorhanden).

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Kommandos werden ausgegeben, bzw. die Definitionen der Kommandos, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *PROGRAM(...)
Bestimmt, dass die Definitionen von Programmen ausgegeben werden.

NAME = *ALL(...)
Die Definitionen von allen Programmen werden ausgegeben.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Programme werden nicht ausgegeben.

NAME = *REMOVED
Alle Programme, die entfernt sind (und die wiederhergestellt werden können, weil ihre Beschreibung auf einer höheren Stufe der Hierarchie vorhanden ist) werden ausgegeben.

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Programme werden ausgegeben, bzw. die Definitionen der Programme, deren Name zum Wildcard-Suchmuster passt.

OBJECT = *STATEMENT(...)
Bestimmt, dass die Definitionen von Anweisungen ausgegeben werden.

NAME = *ALL(...)
Die Definitionen von allen Anweisungen werden ausgegeben.

EXCEPT = *NONE / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der hier angegebenen Anweisungen werden nicht ausgegeben.

NAME = *REMOVED
Alle Anweisungen, die entfernt sind (und die wiederhergestellt werden können, weil ihre Beschreibung auf einer höheren Stufe der Hierarchie vorhanden ist) werden ausgegeben.

NAME = *CURRENT
Die aktuellen Anweisungen werden ausgegeben (sofern vorhanden).

NAME = <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Die Definitionen der namentlich genannten Anweisungen werden ausgegeben, bzw. die Definitionen der Anweisungen, deren Name zum Wildcard-Suchmuster passt.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisungen gehören.

OBJECT = *PRIVILEGE(...)
Bestimmt, dass die Definitionen von Privilegien ausgegeben werden.

NAME = *ALL / <structured-name 1..30>
Die Definitionen von allen bzw. den namentlich genannten Privilegien werden ausgegeben.

OBJECT = *OPERAND(...)
Bestimmt, dass die Definition eines Operanden ausgegeben wird. Steht dieser Operand in einer Struktur, so wird er durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig (oder gar nicht) angegeben werden. Ein Operand, der zur Identifizierung der auszugebenden Operandendefinition nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten VALUE-Li=*NO betrachtet SDF-A den durch OPERAND-Li bestimmten Operanden als den, dessen Definition auszugeben ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu VALUE-Li ein anderer Wert als *NO angegeben, so muss zu OPERAND-Li+1 ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Bestimmt den Operanden, dessen Definition auszugeben ist (VALUE-L1=*NO), oder einen Operanden, der auf dem Pfad zu diesem liegt (VALUE-L1≠*NO). *CURRENT bedeutet, dass OPERAND-L1 aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Die Angabe *NO bedeutet, dass die Definition von OPERAND-L1 auszugeben ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition auszugeben ist. Ist der struktureinleitende Operandenwert vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,..., VALUE=<c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der struktureinleitendene Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L2 für die Spezifizierung des Operanden, dessen Definition auszugeben ist, irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der, dessen Definition auszugeben ist (VALUE-L2=*NO), oder einer, der auf dem Pfad zu diesem liegt (VALUE-L2≠*NO).

VALUE-L2 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition auszugeben ist. Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung des Operanden, dessen Definition auszugeben ist, irrelevant ist. Andernfalls ist der Name eines Operanden anzugeben, der innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig ist. Dieser Operand ist entweder der, dessen Definition auszugeben ist (VALUE-L3=*NO), oder einer, der auf dem Pfad zu diesem liegt (VALUE-L3≠*NO).

VALUE-L3 = analog VALUE-L1
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung des Operanden irrelevant ist. Andernfalls ist ein Operandenwert anzugeben, der eine Struktur einleitet. Diese muss den Operanden unmittelbar oder mittelbar enthalten, dessen Definition auszugeben ist. Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

VALUE-L4 = analog VALUE-L1
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, zu dem die auszugebende Operandendefinition gehört.

ORIGIN = *CURRENT
Die auszugebende Operandendefinition gehört zu einem Kommando (bzw. zu einer Anweisung), das zurzeit entweder selbst aktuelles Objekt ist oder einen Operanden oder Operandenwert enthält, der aktuelles Objekt ist.

ORIGIN = *COMMAND(...)
Die Operandendefinition gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Die Operandendefinition gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *VALUE(...)
Bestimmt, dass die Definition eines Operandenwerts ausgegeben wird. Dieser Operandenwert wird durch Angabe des zu ihm führenden Pfades spezifiziert, d.h. durch Angabe der auf diesem Pfad liegenden Operanden und struktureinleitenden Operandenwerte. Gehört der Operandenwert zu einem Operanden, der außerhalb jeder Struktur steht, so liegt auf dem Pfad nur dieser Operand. Gehört der Operandenwert zu einem Operanden, der in einer Struktur steht, so liegen auf dem Pfad außerdem die übergeordneten Operanden und die zu diesen gehörenden struktureinleitenden Operandenwerte. Ist der Name eines auf diesem Pfad liegenden Operanden nicht nur innerhalb seiner Struktur eindeutig, sondern auch in Bezug auf die übergeordnete Struktur (oder kommando- bzw. anweisungsglobal), so muss der Pfad nicht vollständig angegeben werden. Ein Operand, der zur Identifizierung der auszugebenden Operandenwertdefinition nicht unbedingt gebraucht wird, sowie der zu ihm gehörende Operandenwert muss nicht angegeben werden. Ein zu VALUE-Li (i=1,..,5) angegebener Operandenwert muss zu dem Operanden gehören, der durch OPERAND-Li bestimmt ist. Nach dem ersten OPERAND-Li+1=*NO betrachtet SDF-A den durch VALUE-Li bestimmten Operandenwert als den, dessen Definition auszugeben ist. SDF-A wertet dann die Angaben zu eventuellen restlichen OPERAND-Lj, VALUE-Lj nicht mehr aus. Wird zu OPERAND-Li ein anderer Wert als *NO angegeben, so muss zu VALUE-Li ebenfalls ein von *NO verschiedener Wert angegeben werden.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Bestimmt den Operanden, zu dem der Operandenwert, dessen Definition auszugeben ist, gehört (OPERAND-L2=*NO), oder einen Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L2≠*NO). *ABOVE-CURRENT bedeutet, dass ein zu OPERAND-L1 gehörender Wert aktuelles Objekt ist. <structured-name> muss ein kommando- bzw. anweisungsglobal eindeutiger Operandenname sein.

VALUE-L = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Bestimmt den Operandenwert, dessen Definition auszugeben ist (OPERAND-L2=*NO), oder einen struktureinleitenden Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L2≠*NO). *CURRENT bedeutet, dass VALUE-L1 aktuelles Objekt ist. Ist er nicht aktuelles Objekt und vom Datentyp KEYWORD(-NUMBER), so ist der für ihn definierte Einzelwert anzugeben (siehe ADD-VALUE TYPE=*KEYWORD,...,

VALUE=<c-string>). Dabei ist zu beachten, dass dieser Einzelwert in jedem Fall ohne vorangestellten Stern anzugeben ist. Ist der Operandenwert nicht vom Typ KEYWORD(-NUMBER), so ist der für ihn definierte Datentyp anzugeben.

OPERAND-L2 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass die Definition von VALUE-L1 auszugeben ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der Operandenwert, dessen Definition auszugeben ist, gehört (OPERAND-L3=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L3≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L1 bestimmten Struktur eindeutig sein.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Die Angabe *NO bedeutet, dass VALUE-L2 für die Spezifizierung des auszugebenden Operandenwerts irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser ist entweder der Operandenwert, dessen Definition auszugeben ist (OPERAND-L3=*NO), oder ein struktureinleitender Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L3≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
Die Angabe *NO bedeutet, dass OPERAND-L3 für die Spezifizierung der auszugebenden Operandenwertdefinition irrelevant ist. Andernfalls ist der Name des Operanden anzugeben, zu dem der Operandenwert, dessen Definition auszugeben ist, gehört (OPERAND-L4=*NO), oder der Name eines Operanden, der auf dem Pfad zu diesem Operandenwert liegt (OPERAND-L4≠*NO). Wird ein Operandenname angegeben, so muss dieser innerhalb der durch VALUE-L2 bestimmten Struktur eindeutig sein.

VALUE-L3 = analog VALUE-L2
Die Angabe *NO bedeutet, dass VALUE-L3 für die Spezifizierung der auszugebenden Operandenwertdefinition irrelevant ist. Andernfalls ist ein Operandenwert anzugeben. Dieser ist entweder der Operandenwert, dessen Definition auszugeben ist (OPERAND-L4=*NO), oder ein struktureinleitender Operandenwert, der auf dem Pfad zu diesem liegt (OPERAND-L4≠*NO). Weitere Informationen siehe VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L4 = analog VALUE-2
siehe VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
siehe OPERAND-L3.

VALUE-L5 = analog VALUE-2
siehe VALUE-L2.

ORIGIN =
Bestimmt das Kommando oder die Anweisung, zu dem die auszugebende Operandenwertdefinition gehört.

ORIGIN = *CURRENT
Die Operandenwertdefinition gehört zu einem Kommando (bzw. zu einer Anweisung), das zurzeit entweder selbst aktuelles Objekt ist oder einen Operanden oder Operandenwert enthält, der aktuelles Objekt ist.

ORIGIN = *COMMAND(...)
Die Operandenwertdefinition gehört zu einem Kommando.

NAME = <structured-name 1..30>
Name des Kommandos.

ORIGIN = *STATEMENT(...)
Die Operandenwertdefinition gehört zu einer Anweisung.

NAME = <structured-name 1..30>
Name der Anweisung.

PROGRAM = <structured-name 1..30>
Name des Programms, zu dem die Anweisung gehört.

OBJECT = *CORRECTION-INFORMATION(...)
Der Operandenwert ist für spezielle Zwecke der Systemsoftwareentwicklung von Fujitsu Siemens Computers reserviert und wird deshalb nicht beschrieben.

ATTACHED-INFORMATION =
Bestimmt, welche der Definitionen, die zu dem angegebenen Objekt gehören, ausgegeben werden.

ATTACHED-INFORMATION = *YES
Die Definition des angegebenen Objekts wird einschließlich der Definitionen aller Objekte ausgegeben, die dem angegebenen Objekt zugeordnet sind. (Anwendungsbereich mit zugehörigen Kommandos, Programm mit zugehörigen Anweisungen, Kommando oder Anweisung mit zugehörigen Operanden, Operand mit zugehörigen Operandenwerten, Operandenwert mit zugehöriger Struktur, Globalinformation mit sprachabhängigen Texten.)

ATTACHED-INFORMATION = *NO
Die Definition des angegebenen Objekts wird ohne die Definitionen der Objekte ausgegeben, die dem angegebenen Objekt zugeordnet sind. (Anwendungsbereich ohne zugehörige Kommandos, Programm ohne zugehörige Anweisungen, Kommando oder Anweisung ohne zugehörige Operanden, Operand ohne zugehörige Operandenwerte, Operandenwert ohne zugehörige Struktur, Globalinformation ohne sprachabhängige Texte.)

ATTACHED-INFORMATION = *IMMEDIATE
Die Definition des angegebenen Objekts wird mit den Definitionen der Objekte ausgegeben, die dem angegebenen Objekt unmittelbar zugeordnet sind. Anwendungsbereiche bzw. Programme werden mit den zugehörigen Kommandos bzw. Anweisungen ausgegeben, jedoch ohne die zugehörigen Operanden und Operandenwerte. Für alle anderen Objekte wirkt *IMMEDIATE wie *YES.

SIZE =
Bestimmt den Umfang der Ausgabe. Was der Operand SIZE konkret bewirkt, hängt mit vom Operanden OBJECT ab. Bei IMPLEMENTATION-INFO=*YES bewirkt MEDIUM das Gleiche wie MAXIMUM. Der Operand hat keine Auswirkung auf die Ausgabe der Globalinformation. Für OBJECT=*COMMAND oder *STATEMENT sowie analog für *OPERAND und *VALUE werden folgende Informationen ausgegeben:

SIZE = *MINIMUM
Bei IMPLEMENTATION-INFO=*NO gilt:
der Kommando-/Anweisungsname, alle zugehörigen Operandennamen, alle zugehörigen Default-Werte und die struktureinleitenden Operandenwerte werden ausgegeben. Zusatzinformationen zu den Operandenwerten und Hilfetexte werden nicht ausgegeben.
Bei IMPLEMENTATION-INFO=*YES gilt:
Teile der Definition, die mit den Default-Werten der zugehörigen ADD-Anweisungen definiert sind, werden nicht ausgegeben.

SIZE = *MAXIMUM
Bei IMPLEMENTATION-INFO=*NO gilt:
Der Kommando-/Anweisungsname, alle zugehörigen Operandennamen, alle zugehörigen Default-Werte und alle sonstigen zugehörigen Operandenwerte werden ausgegeben. Außerdem werden die Zusatzinformationen zu den Operandenwerten und die Hilfetexte ausgegeben.
Bei IMPLEMENTATION-INFO=*YES gilt:
Alle Teile der Definition werden ausgegeben.

SIZE = *MEDIUM
Bei IMPLEMENTATION-INFO=*NO gilt:
Der Kommando-/Anweisungsname, alle zugehörigen Operandennamen, alle zugehörigen Default-Werte und alle sonstigen zugehörigen Operandenwerte werden ausgegeben. Zusatzinformationen zu den Operandenwerten und Hilfetexte werden nicht ausgegeben.
Bei IMPLEMENTATION-INFO=*YES gilt:
Teile der Definition, die mit den Default-Werten der zugehörigen ADD-Anweisungen definiert sind, werden nicht ausgegeben.

IMPLEMENTATION-INFO =
Bestimmt, wie die Ausgabe aufbereitet wird. Für die Globalinformation ist dieser Operand wirkungslos.

IMPLEMENTATION-INFO = *NO(...)
Die Definitionen der angegebenen Objekte werden in einer handbuchähnlichen Darstellung ausgegeben. Die Operanden, die mit PRESENCE=*INTERNAL-ONLY belegt sind, werden hier nicht aufgelistet.

FORM =
Bestimmt, ob die Definitionen von Objekten, die für den geführten Dialog nicht zugelassen sind, ausgegeben werden.

FORM = *UNGUIDED
Die Definitionen werden ausgegeben.

FORM = *GUIDED
Die Definitionen werden nicht ausgegeben.

LANGUAGE = E / <name 1..1>
Bestimmt, in welcher Sprache die Hilfetexte ausgegeben werden (E=Englisch, D=Deutsch). Für die Globalinformation ist dieser Operand wirkungslos.

IMPLEMENTATION-INFO = *YES
Es werden die SDF-A-Anweisungen aufgelistet, mit denen sich die spezifizierten Objekte in der Syntaxdatei definieren lassen.

LINE-LENGTH = *STD / <integer 72..132>
Bestimmt die Zeilenlänge für die Ausgabe.

LINE-LENGTH = *STD
Der Default-Wert beträgt 74 Zeichen bei Ausgabe auf den Bildschirm und 72 Zeichen bei Ausgabe in eine Datei.

LINES-PER-PAGE = *STD / *UNLIMITED(...) / <integer 1..200>
Bestimmt die Anzahl der Zeilen pro Seite. Diese Zahl enthält nicht die beiden Zeilen für den Header, den SDF-A für jede neue Seite erzeugt. Der Header wird nur bei OUTPUT= *SYSLST erzeugt.

LINES-PER-PAGE = *STD
Der Default-Wert beträgt 24 Zeilen bei Ausgabe auf den Bildschirm und 55 Zeilen bei Ausgabe in eine Datei.

LINES-PER-PAGE = *UNLIMITED(...)
Keine Kontrolle durch SDF-A (es wird kein Header erzeugt)

OUTPUT-FORM =
Legt fest, welche Zeichen am Zeilenanfang ausgegeben werden.

OUTPUT-FORM = *STD
Das erste Zeichen jeder Zeile ist ein Leerzeichen.

OUTPUT-FORM = *FOR-INPUT
Am Beginn jeder Zeile werden zwei Schrägstriche (//) ausgegeben.
Diese Angabe kann in Kombination mit IMPLEMENTATION=*YES dazu verwendet werden, SDF-A-Anweisungen zu erzeugen, mit deren Hilfe eine Syntaxdatei oder ein Syntaxdatei-Objekt wiederhergestellt wird.

Beispiel:

Mit der folgenden SDF-A-Anweisung wird eine Datei erstellt, die SDF-A-Anweisungen enthält, mit denen eine Syntaxdatei wiederhergestellt werden kann:

```
//SHOW OBJECT=*ALL,IMPLEMENTATION-INFO=*YES, -
//    LINES-PER-PAGE=*UNLIMITED(OUTPUT-FORM=*FOR-INPUT),-
/     OUTPUT=*SYSLST(07),LINE-LENGTH=72
```

OUTPUT =
Bestimmt das Ausgabemedium für die gewünschten Informationen.

OUTPUT = *SYSOUT
die Ausgabe erfolgt in die logische Systemdatei SYSOUT, d.h. im Dialog in der Regel auf den Bildschirm.

OUTPUT = *SYSLST(...)
die Ausgabe erfolgt in die logische Systemdatei SYSLST.

SYSLST-NUMBER = *STD / <integer 1..99>
Bestimmt die Nummer der logischen Systemdatei SYSLST. Bei Angabe von *STD erhält die logische Systemdatei SYSLST keine Nummer.

PRIVILEGE = *ANY / list-poss(64): <structured-name 1..30>
Es werden nur die Objekte ausgegeben, denen mindestens eines der in der Liste genannten Privilegien zugeordnet ist. Bei Angabe von *ANY werden die Objekte ohne Beachtung ihrer Privilegien ausgegeben.

SHOW-CORRECTION-INFORMATION
Korrekturinformation der Syntaxdatei ausgeben

Die Anweisung SHOW-CORRECTION-INFORMATION gibt Informationen über in der Syntaxdatei enthaltene Korrekturen aus.

Die Anweisung ist nur für Diagnosezwecke vorgesehen.

```
SHOW-CORRECTION-INFORMATION

CORRECTION-ID = *SOURCE(...) / *OBJECT(...)
    *SOURCE(...)
        | PM-NUMBER = *ALL / list-poss(100): <alphanum-name 8..8>
    *OBJECT(...)
        | PM-NUMBER = *ALL / <alphanum-name 8..8>
        | ,JULIAN-DATE = *ANY / <integer 0..999>
,PRODUCT-NAME = *ANY / <structured-name 1..15>(...)
    <structured-name 1..15>(...)
        | VERSION = *ANY / <product-version>
```

Die Operandenbeschreibung entfällt an dieser Stelle, da die Anweisung nur für Diagnosezwecke vorgesehen ist.

SHOW-STATUS
Status der geöffneten Syntaxdatei anzeigen

Mit der Anweisung SHOW-STATUS veranlassen Sie, dass SDF-A den Namen der gerade geöffneten Syntaxdatei und Informationen über eine oder mehrere zugeordnete Referenzsyntaxdateien ausgibt.

SHOW-STATUS

Diese Anweisung hat keine Operanden.

STEP
Wiederaufsetzpunkt definieren

Mit der Anweisung STEP definieren Sie in einer Folge von SDF-A-Anweisungen einen Wiederaufsetzpunkt für die Fehlerbehandlung. Die Eingabe der Anweisung STEP ist nur in Prozeduren und Stapelaufträgen zulässig.

STEP

Diese Anweisung hat keine Operanden.

Trifft SDF-A auf einen syntaktischen oder schwerwiegenden logischen Fehler, dann leitet es folgende Schritte ein:

– es wird eine Fehlermeldung ausgegeben,

– die laufende SDF-A-Anweisung wird abgebrochen,

– die nachfolgenden Anweisungen werden bis zum Erreichen von STEP oder END überlesen.

Erreicht SDF-A als Nächstes ein END, so erzeugt es ein abnormales Programmende (TERM UNIT=STEP,MODE=ABNORMAL) und aktiviert den Spin-Off-Mechanismus. Erreicht SDF-A als Nächstes ein STEP, so fährt es mit der auf STEP folgenden Anweisung fort. Falls ein Fehler keinen Einfluss auf den ordnungsgemäßen Ablauf des laufenden Jobs hat, so muss der Benutzer von SDF-A den abnormalen Programmabbruch durch Einfügen eines STEP abfangen.

Geringfügige logische Fehler korrigiert SDF-A selbstständig unter Ausgabe einer Warnmeldung. Es wird kein Spin-Off-Mechanismus aktiviert.

6 SDF-Programmschnittstelle

SDF bietet auch für benutzereigene Programme die Möglichkeit, Anweisungen über die komfortable SDF-Benutzeroberfläche einzulesen, zu analysieren und zu korrigieren. Benutzereigene Programme können damit über die gleiche Benutzeroberfläche bedient werden wie das Betriebssystem selbst.

Damit ein Benutzerprogramm mit SDF arbeitet, müssen folgende Voraussetzungen geschaffen werden:

– Es muss mit SDF-A eine Syntaxbeschreibung in einer Syntaxdatei erstellt werden. Dazu stehen Anweisungen wie ADD-PROGRAM und ADD-STMT zur Verfügung.

– Im Programm selbst muss SDF zur Anforderung der Anweisungen aufgerufen werden. Dies geschieht mit Hilfe der in diesem Kapitel beschriebenen SDF-Makros bzw. mit den Funktionsaufrufen der Schnittstelle zwischen SDF und höheren Programmiersprachen.

6.1 Makroaufrufe zur Implementierung von Anweisungen

Mit Hilfe der in diesem Kapitel beschriebenen SDF-Makros wird SDF zur Anforderung der Anweisungen aufgerufen.

Die Makros stehen in der Bibliothek SYSLIB.SDF.045 zur Verfügung, die mit SDF ausgeliefert wird. Die Beschreibung der Makros befindet sich in diesem Handbuch, weil sie nur genutzt werden können, wenn mit SDF-A eine passende Syntax definiert werden kann.

Die Verarbeitung der Makros unterscheidet sich je nach der im System laufenden SDF-Version.

Änderungen für Programme ab SDF V4.1A

Mit SDF V4.1 wurde das Layout des normierten Übergabebereiches geändert. Aus diesem Grund wurde der Makro CMDSTRUC zur Generierung des Übergabebereiches durch den Makro CMDTA ersetzt. Außerdem wurden die Makros RDSTMT, TRSTMT und CORSTMT zur Behandlung von Anweisungen durch die neuen Makros CMDRST, CMDTST und CMDCST ersetzt.

Das bis SDF V4.0 verwendete Format des normierten Übergabebereiches sowie die Makros CMDSTRUC, RDSTMT, TRSTMT und CORSTMT sollten in neuen Programmen nicht mehr verwendet werden; sie sind jedoch im Anhang noch beschrieben (siehe Abschnitt „Änderungen der SDF-Programmschnittstelle“ auf Seite 605ff).

Die SDF-Schnittstelle zu höheren Programmiersprachen unterstützt nur den normierten Übergabebereich im alten Format. Die Nutzung des neuen Formats ist nur über die Assembler-Schnittstelle möglich!

6.2 Makroaufrufe in gleichzeitig eröffneten Syntaxdateihierarchien

Mit dem Makro OPNCALL eröffnen Benutzerprogramme eine Syntaxdateihierarchie parallel zu der Syntaxdateihierarchie, die bei der LOGON-Verarbeitung für den Benutzerauftrag eröffnet wurde. Parallel zum Arbeitskontext, der von SDF für den Benutzerauftrag erstellt wurde, wird ein neuer Arbeitskontext erstellt.

Der neue Kontext wird vom Programm eröffnet und dazu benutzt, weitere Eingaben zu analysieren. Mit dem Makro CLSCALL oder bei Programmbeendigung wird der Kontext wieder geschlossen.

Die Kontexte, die vom Makro OPNCALL eröffnet werden, heißen daher „Programmkontexte“. Das Benutzerprogramm bestimmt die Syntaxdateihierarchie. Der Kontext, der von SDF bei der Einleitung des Benutzerauftrags erstellt wird, heißt „Systemkontext“. Der Systemkontext benutzt die Voreinstellungen der Systembetreuung in der SDF-Parameterdatei.

- Der Systemkontext wird vom Kommandoprozessor dazu verwendet, Kommandos zu lesen und zu analysieren, bevor sie vom Kommandoprozessor ausgeführt werden. Die Syntaxdateihierarchie und die SDF-Optionen werden wie beschrieben bearbeitet (siehe Seite 25ff). Der Identifikator des Systemkontexts in den verschiedenen Makros ist die Standard NULL-Adresse.
- Der Programmkontext kann von einem Programm dazu verwendet werden, seine Anweisungen vor der Ausführung zu lesen und zu analysieren. Die Syntaxdateihierarchie wird mit OPNCALL vom Benutzerprogramm für die Ebene der System- und Gruppensyntaxdatei(en) festgelegt. Die SDF-Optionen können lokal im Programmkontext mit MODIFY-SDF-OPTIONS verändert werden. Der Identifikator des Programmkontexts wird vom Makro OPNCALL als Rückgabewert übergeben und von den folgenden Makroaufrufen in diesem Kontext verwendet.

Der Kommandoprozessor kann in keinem Fall ein Kommando ausführen, das in einem Programmkontext verarbeitet wird. Mit dem Makro CMD kann kein Programmkontext erstellt werden.

Mit dem Makro OPNCALL kann ein Benutzerprogramm in bzw. aus den Programmkontexten

– Kommandos übersetzen (d.h. diese analysieren, ohne sie auszuführen)
– Anweisungen lesen, übersetzen oder korrigieren
– Information erhalten.

Der Programmkontext hat folgende Eigenschaften:

– Die Systemsyntaxdateien können entweder die aktuellen Systemsyntaxdateien des Systemkontexts des Benutzerauftrags sein oder es kann explizit mit dem Makro OPNCALL bestimmt werden, dass eine andere Basis-Systemsyntaxdatei aktiviert wird. Wenn die Systembetreuung die aktuellen Systemsyntaxdateien des Systemkontexts mit MODIFY-SDF-PARAMETERS dynamisch wechselt, dann gilt diese Veränderung ab der nächsten Transaktion auch für den Programmkontext, der diese Systemsyntaxdateien benutzt hat.

– Die Gruppensyntaxdatei kann entweder die aktuelle Gruppensyntaxdatei des Systemkontexts des Benutzerauftrags sein oder explizit mit dem Makro OPNCALL bestimmt werden. Sie kann im erstellten Programmkontext nicht gewechselt werden.

– Die Benutzersyntaxdatei ist die aktuelle Benutzersyntaxdatei des Benutzerauftrags. Sie gilt sowohl für den System- als auch für den Programmkontext und kann mit dem Kommando bzw. der Anweisung MODIFY-SDF-OPTIONS dynamisch gewechselt werden. MODIFY-SDF-OPTIONS wird von CMDRST oder im Puffer von CMDTST (EXECUTE= *YES) gelesen.

– Wenn im Programmkontext eigene Systemsyntaxdateien geöffnet sind, werden die SDF-Standardanweisungen aus dem Programmkontext verwendet. Standardanweisungen, die im Programmkontext nicht verfügbar sind, holt SDF aus dem in der Basis-Systemsyntaxdatei definierten Programm $CMDPGM im Systemkontext.

Veränderungen der Benutzersyntaxdatei und der SDF-Optionen im Programmkontext haben keine Auswirkung auf den Systemkontext. Von einem Benutzerprogramm können gleichzeitig bis zu 255 Kontexte, einschließlich des Systemkontexts, eröffnet werden.

Folgende SDF-Makros können in einem Programmkontext verwendet werden oder diesen bearbeiten:

– OPNCALL:
erstellt den neuen Kontext und bestimmt die Gruppen- und die Systemsyntaxdateien, die aktiviert werden sollen. Der Identifikator des Programmkontexts wird an das Benutzerprogramm als Rückgabewert übergeben. Dieser Identifikator wird bei jedem Aufruf bestimmt, der in einem Programmkontext verarbeitet wird.

– CMDTST, CMDRST, CMDCST und CMDSTA:
Der Identifikator des Programmkontexts wird bestimmt, wenn in dem entsprechenden Kontext gearbeitet wird.

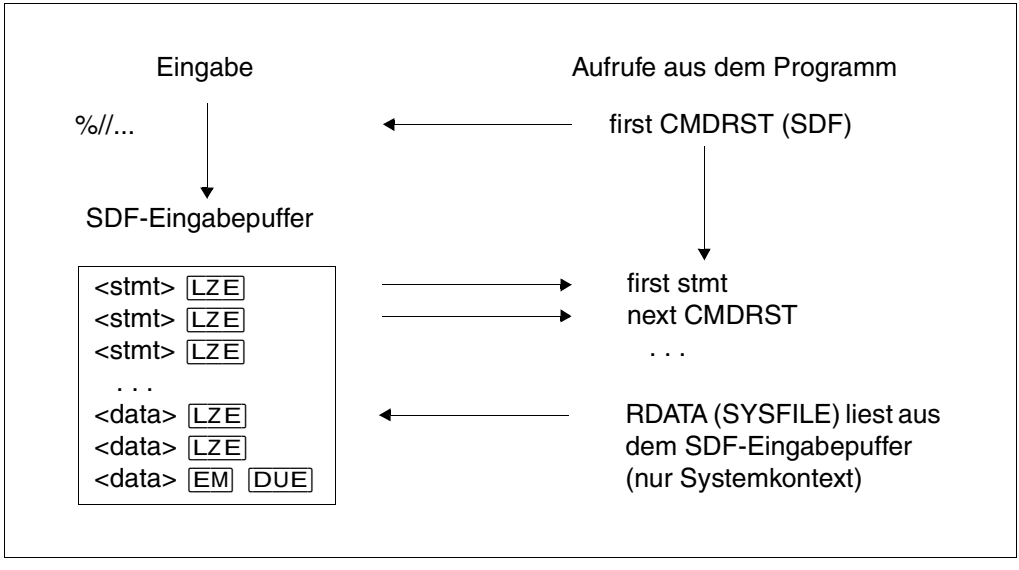
– CLSCALL:
beendet den Programmkontext. Wird kein CLSCALL abgesetzt, werden mit Programmbeendigung alle Programmkontexte freigegeben.

Hinweise zum Programmkontext

Mit dem Programmkontext wird eine neue SDF-Arbeitsumgebung angeboten, die eigene Eingabe- und Syntaxspeicher hat. Deshalb müssen folgende Punkte beachtet werden:

– Wird für eine Liste von Anweisungen geblockte Eingabe verwendet, wirkt die Eingabe nur lokal auf den aktuellen Kontext. Geblockte Eingabe für CMDRST in einem Programmkontext kann keinen anderen Kontext aufrufen.

– In der geblockten Eingabe von Anweisungen können auch Datensätze eingegeben werden. Diese können von einem auf CMDRST folgenden RDATA (SYSFILE) nur im *System*kontext gelesen werden:



RDATA (SYSFILE) kann nur Datensätze aus dem Terminalpuffer des Systemkontexts lesen. RDATA sollte daher nicht nach CMDRST im Programmkontext aufgerufen werden.

– Die EOF-Bedingung für die Anweisungseingabe und den Spin-off-Mechanismus wirkt nur in dem Kontext, in dem die Anweisungen gelesen werden.

– Die Anweisungen werden nicht in der Umgebung des aktuellen Benutzerauftrags übersetzt (CMDTST). Der Benutzer muss den simulierten Benutzerauftragstyp (Stapel- oder Dialogauftrag) im Makro OPNCALL festlegen. Den simulierten Prozedurmodus (Prozedur oder Primary) muss er im Makro CMDTST festlegen.
Voreinstellung ist, dass CMDTST bei der Prüfung der Eingabe keine Prüfung des Benutzerauftragstyps oder des Prozedurmodus ausführen muss. CMDRST liest die Anweisungen in der Umgebung des aktuellen Benutzerauftrags.

CHKPT und RESTART-PROGRAM für Programmkontexte

Beim Checkpoint eines Programms im Programmkontext muss nach RESTART-PROGRAM mit folgendem Verhalten für die wiedereröffneten Hierarchien gerechnet werden:

– War zum Zeitpunkt des CHKPT keine besondere Systemsyntaxdatei für den Programmkontext festgelegt (siehe OPNCALL SFSYSTM=*STD), dann gilt nach dem Restart die Systemsyntaxdatei des aktuellen Systemkontexts.
War eine besondere Systemsyntaxdatei festgelegt, dann wird genau diese wieder eröffnet.

– War zum Zeitpunkt des CHKPT keine besondere Gruppensyntaxdatei für den Programmkontext festgelegt (siehe OPNCALL SFGROUP=*STD), dann gilt nach dem Restart die Gruppensyntaxdatei des aktuellen Systemkontexts, die der aktuellen PROFILE-ID zugeordnet ist.
War eine besondere Gruppensyntaxdatei festgelegt, dann wird genau diese wieder eröffnet.

– Die Benutzersyntaxdateien, die zum Zeitpunkt des Checkpoint geöffnet waren, werden beim Restart erneut geöffnet.

Nach RESTART-PROGRAM werden die Kontexte so wiederhergestellt, als ob die Systemsyntaxdatei von der Systembetreuung dynamisch verändert worden wäre.

6.3 Aufbau des normierten Übergabebereichs

Normierte Übergabebereiche werden für drei Zwecke benötigt.

1. SDF übergibt an das Programm eine analysierte Anweisung. Im Programm ist Speicherplatz für mindestens einen solchen Bereich zu reservieren. Der Bereich ist so groß anzulegen, dass er jede mögliche Anweisungseingabe, die SDF für das Programm analysiert hat, aufnehmen kann. Die Adresse dieses Bereichs ist bei den Makros CMDRST und CMDTST im Operanden OUTPUT anzugeben, beim Makro CMDCST im Operanden INOUT.
2. Das Programm gibt an SDF eine semantisch fehlerhafte Anweisung zurück. Im Programm ist dafür kein besonderer Speicherplatz zu reservieren. Das Programm nimmt denselben Bereich, in den SDF zuvor die analysierte Anweisung geschrieben hat. Nachdem das Programm in dieser Anweisung Semantikfehler erkannt hat, ergänzt es die Anweisung um Informationen für den Semantikfehlerdialog und gibt sie an SDF zurück. Die Adresse dieses Bereichs ist beim Makro CMDCST im Operanden INOUT anzugeben.
3. Das Programm übergibt an SDF Werte, die eingegebene Operandenwerte oder den Default-Wert ersetzen sollen. Für jede Anweisung, in der derartige Operandenwerte vorkommen können (siehe ADD-VALUE...,VALUE=<string>(OVERWRITE-POSSIBLE=*YES),... bzw. ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES) ist im Programm genau ein solcher Bereich anzulegen und vom Programm zu versorgen. Die Adressen dieser Bereiche sind bei den Makros CMDRST und CMDTST im Operanden DEFAULT anzugeben.

Mit SDF V4.1 hat sich das Format des normierten Übergabebereiches geändert. Das neue Layout wird mit dem Makro CMDTA erzeugt und muss mit den neuen Makros CMDRST, CMDTST und CMDCST verwendet werden.

Für alle drei Fälle hat der Übergabebereich den gleichen formalen Aufbau (siehe Bild 7).

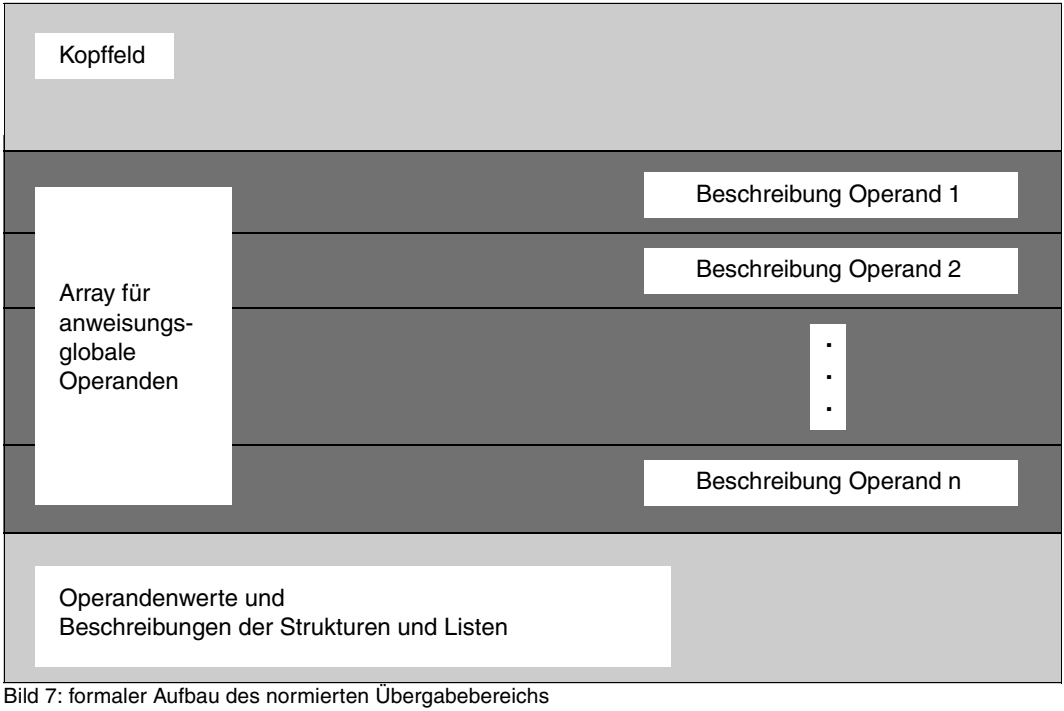


Bild 7: formaler Aufbau des normierten Übergabebereichs

Der normierte Übergabebereich beginnt auf Wortgrenze. In Byte 0 bis 39 liegt das Kopffeld. Es enthält u.a. den internen Anweisungsnamen und die Version der Anweisung (siehe ADD-STMT ...,INTERNAL-NAME=...,STMT-VERSION=...).
Ab Byte 40 beginnt das Array für anweisungsglobale Operanden, d.h. für Operanden, die mit RESULT-OPERAND-LEVEL=1 definiert sind (siehe ADD-OPERAND). In ihm steht für jeden dieser Operanden eine 8 Byte lange Beschreibung. Die Operandenbeschreibungen sind in der Reihenfolge angeordnet, die sich aus den in der Anweisungsdefinition festgelegten Operandenpositionen ergibt (siehe ADD-OPERAND ...,RESULT-OPERAND-NAME=*POSITION(POSITION=<integer>)). Jede Operandenbeschreibung enthält u.a. die Absolutadresse, unter der der zugehörige Operandenwert bzw. die Beschreibung der zugehörigen Liste oder nichtlinearisierten Struktur im Übergabebereich abgelegt ist. In der Beschreibung einer nichtlinearisierten Struktur steht ein Operanden-Array, das die in der Struktur enthaltenen Operanden beschreibt. Es ist genauso aufgebaut, wie das Array für die anweisungsglobalen Operanden.

Im einfachsten Fall hat ein Operand lediglich einen einfachen Wert (siehe Bild 8). Eine Operandenbeschreibung mit einfachem Wert enthält auch die Syntax-Attribute für diesen Wert.

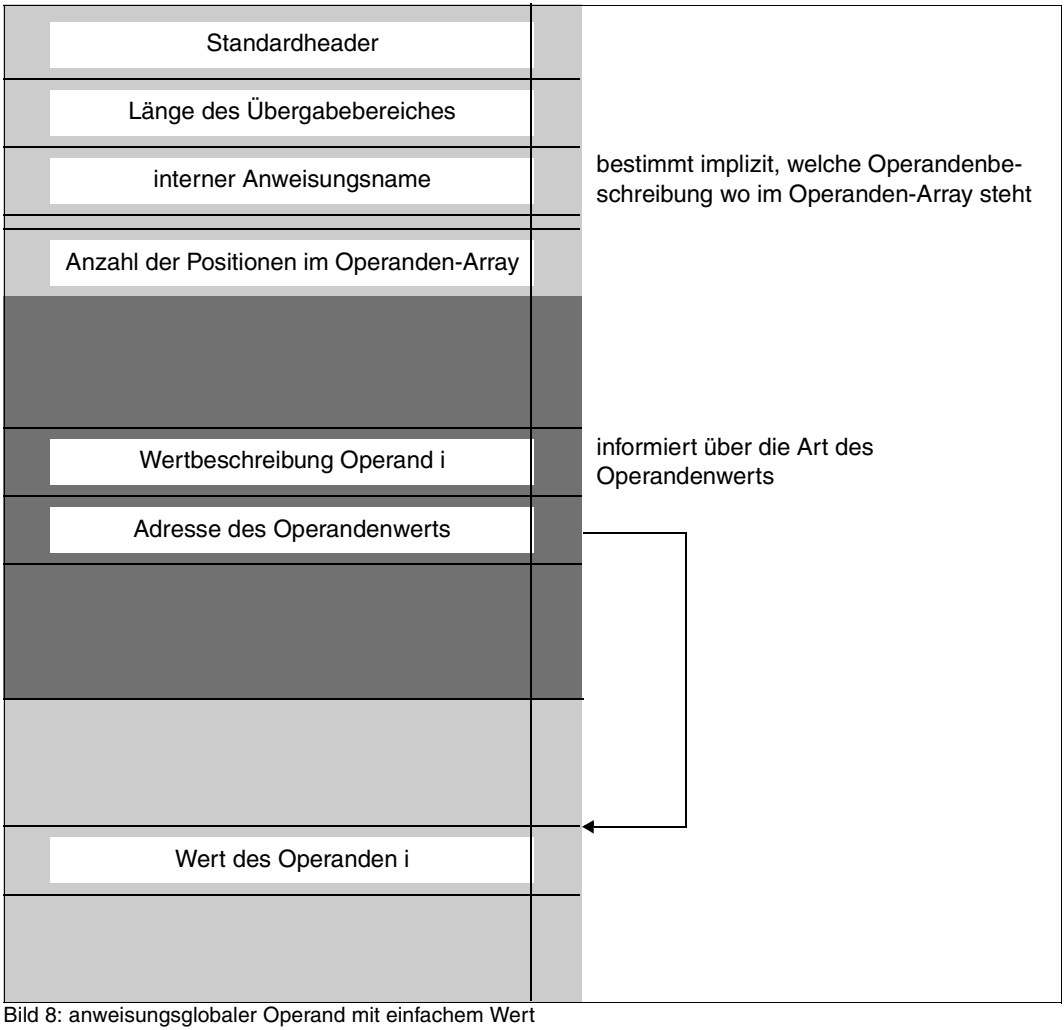


Bild 8: anweisungsglobaler Operand mit einfachem Wert

Leitet der Operandenwert eine Struktur ein (Bild 9), so gibt es für ihn eine Strukturbeschreibung. Diese enthält ein Operanden-Array mit Beschreibungen für alle Operanden der Struktur sowie für die Operanden aus linearisierten Unterstrukturen. Die Operandenbeschreibungen sind in der Reihenfolge angeordnet, die sich aus den in der Anweisungsdefinition festgelegten strukturrelativen Operandenpositionen ergibt (siehe ADD-OPERAND ...,RESULT-OPERAND-NAME=*POSITION(POSITION=<integer>)). Die zu den Operanden der Struktur gehörenden Operandenwerte können eine weitere Struktur einleiten und/oder aus einer Liste von Werten bestehen.

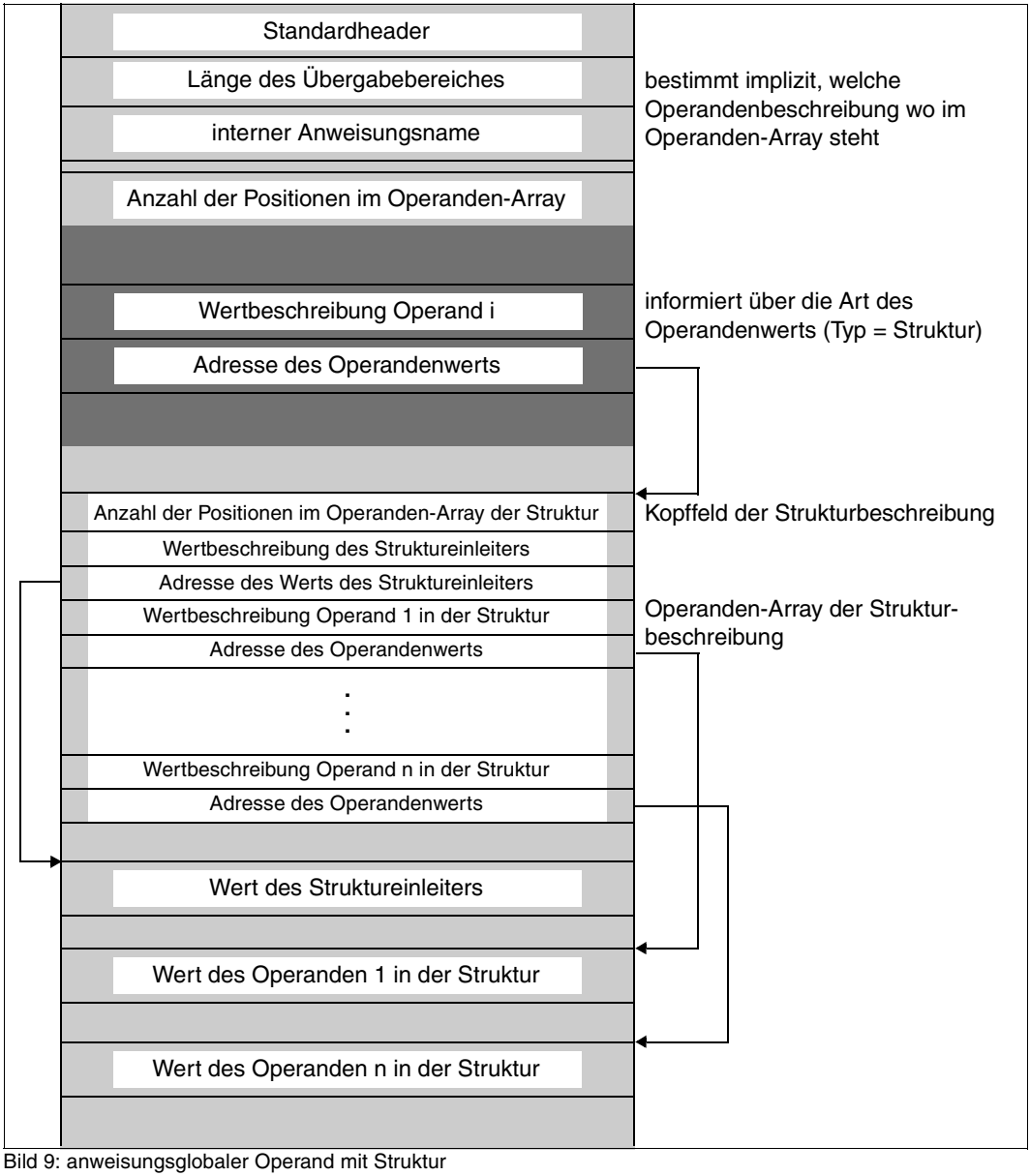


Bild 9: anweisungsglobaler Operand mit Struktur

Von Operanden, die mit ADD-OPERAND...,LIST-POSSIBLE=*YES(...,FORM=*NORMAL) definiert sind, werden die Werte in der in Bild 10 gezeigten Form übergeben. An einem Listenelement kann eine Struktur hängen. In diesem Fall ist „Wert des Listenelements“ eine Strukturbeschreibung.

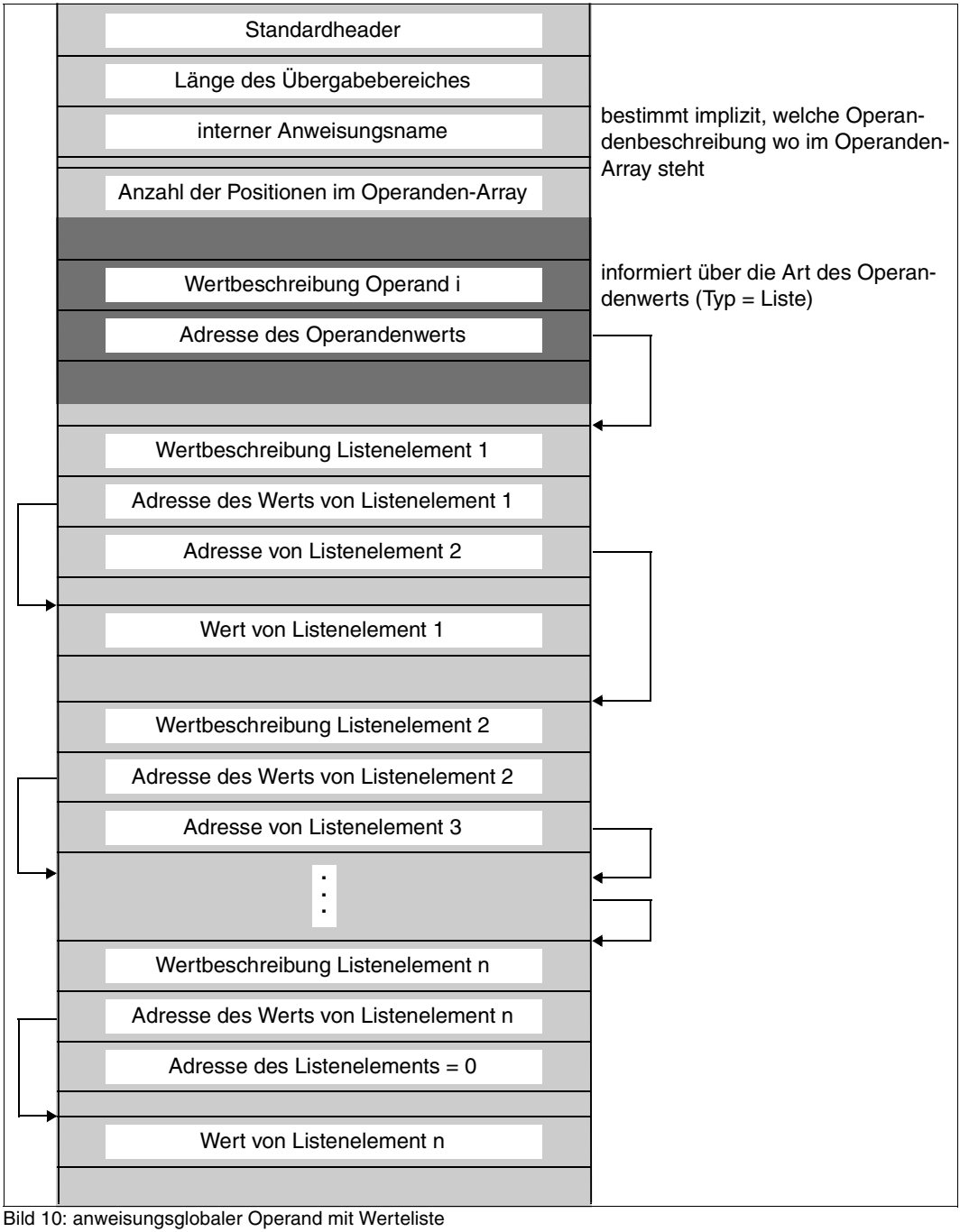


Bild 10: anweisungsglobaler Operand mit Werteliste

Kopffeld des normierten Übergabebereichs

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0 bis 7	Standardheader	Programm	unverändert	Programm
8 bis 11	Länge des Übergabebereichs	Programm	unverändert	Programm
12 bis 19	interner Anweisungsname	SDF	unverändert	Programm
20 bis 23	reserviert	–	–	–
24 bis 26	Version der Anweisung	SDF	unverändert	Programm
27 bis 35	reserviert	–	–	–
36 bis 37	Anzahl der Positionen im Operanden-Array	SDF	unverändert	Programm
38 bis 39	reserviert	–	–	–

Der interne Anweisungsname, die Version der Anweisung und die Anzahl der Positionen in dem Operanden-Array sind in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT...,INTERNAL-NAME=...,MAX-STRUC-OPERAND=...,STMT-VERSION=...). Wenn die tatsächliche Anzahl von Operanden größer ist als der bei MAX-STRUC-OPERAND angegebene Wert, dann wird die tatsächliche Anzahl im Übergabebereich vermerkt.

Übergibt das Programm Default-Werte für eine Anweisung, so muss die Version der Anweisung im Übergabebereich mit der Version der Anweisung in der Syntaxdatei übereinstimmen. Wenn in der Syntaxdatei keine Version vermerkt ist, muss im Übergabebereich '000' (oder binäre Nullen) eingetragen werden, ansonsten wird der Default-Wert zurückgewiesen.

Unterstützt das Programm mehrere Versionen einer Anweisung, dann muss für jede Version der Anweisung ein Default-Wert übergeben werden.

Operandenbeschreibung

Das Operanden-Array und die in ihm stehenden Beschreibungen für die Operanden einer Struktur sind genau so aufgebaut wie bei den anweisungsglobalen Operanden.

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0 bis 3	Wertbeschreibung			
0	Zusatzinformation	siehe unten	siehe unten	siehe unten
1	Typbeschreibung	SDF	unverändert	Programm [1]
2	Globale Syntaxattribute	SDF	–	–
3	Typspezifische Syntaxattribute	SDF	–	–
4 bis 7	Absolutadresse (aligned abgelegt) des zu dem Operanden gehörenden Werts bzw. bei Strukturen oder Listen der weiteren Beschreibung	SDF	unverändert	Programm [1]

[1] Angabe nur für Operanden, zu denen umzusetzende Operandenwerte gehören, d.h. wenn Bit 0 der Zusatzinformation gesetzt ist.

Zusatzinformation

Die Zusatzinformation steht im ersten Byte der Wertbeschreibung. Die folgenden Angaben über die Zusatzinformation gelten unabhängig davon, ob die Zusatzinformation in einer Operandenbeschreibung, im Kopffeld einer Strukturbeschreibung, in der Beschreibung eines Listenelements oder in einer OR-Listenbeschreibung vorkommt.

Bit	Wert	Inhalt	Feldinhalt kommt im Fall ... von		
			analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0	0	Wert ist nicht vorhanden	SDF [1]	unverändert	Programm [2]
0	1	Wert ist vorhanden	SDF	unverändert	Programm [2]
1	0	Wert ist änderbar	–	Programm [3]	–
1	1	Wert ist nicht änderbar	–	Programm [3]	–
2	0	Wert ist nicht fehlerhaft	–	Programm	–
2	1	Wert ist fehlerhaft	–	Programm	–

Fortsetzung ➠

Bit	Wert	Inhalt	Feldinhalt kommt im Fall ... von		
			analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
3	0	Wert soll nicht als Default eingesetzt werden	–	–	Programm
3	1	Wert soll als Default eingesetzt werden	–	–	Programm
4 bis 7	–	reserviert	–	–	Programm

[1] z.B. Werte zu Operanden, die mit ADD-OPERAND...,PRESENCE=*EXTERNAL-ONLY definiert sind, oder Werte in nicht angesprochenen Strukturen.

[2] 0 für Operanden, zu denen weder umzusetzende Operandenwerte gehören noch Strukturen, die Operanden mit umzusetzenden Werten enthalten.
1 für Operanden, zu denen entweder umzusetzende Operandenwerte gehören oder Strukturen, die Operanden mit umzusetzenden Werten enthalten.

[3] Alle Listenwerte, die nach dem ersten änderbaren Listenwert kommen, betrachtet SDF unabhängig davon, wie Bit 1 gesetzt ist, als änderbar. Dadurch werden schon bearbeitete Listenelemente gegen Überschreiben geschützt.

Typbeschreibung

Die Typbeschreibung steht im zweiten Byte der Wertbeschreibung. Die Angaben über die Typbeschreibung gelten unabhängig davon, ob die Typbeschreibung in einer Operandenbeschreibung, im Kopffeld einer Strukturbeschreibung, in der Beschreibung eines Listenelements oder in einer OR-Listenbeschreibung vorkommt.

Strukturbeschreibungen können durch ein Programm eingegeben werden, um eigene Default-Werte anzugeben. Die Eingabe der Default-Werte ist möglich:

– im internen Format (wie beim Operanden OUTPUT der SDF-A-Anweisung ADD-VALUE) oder
– im externen Format als Zeichenfolge analog der Operandenbeschreibung. Dabei muss der Hilfsdatentyp <input-text> benutzt werden. Der Wert wird dann so analysiert, als ob er über die Benutzerschnittstelle eingegeben worden wäre.

Wert (dezimal)	Bedeutung
1	Kommandorest (Command-Rest)
2	Ganzzahl (Integer)
4	X-Zeichenkette (X-String)
5	C-Zeichenkette (C-String)
6	Name (Name)
7	alphanumerischer Name (Alphanumeric-Name)

Fortsetzung ➠

Wert (dezimal)	Bedeutung
8	strukturierter Name (Structured-Name)
9	Marke (Label)
11	Dateiname (Filename)
12	teilqualifizierter Dateiname (Partial-Filename)
13	Uhrzeit (Time)
14	Datum (Date)
15	zusammengesetzter Name (Composed-Name)
16	Text (Text)
17	Katalog-Kennung (Cat-Id)
18	Eingabe-Text (Input-Text)
19	Struktur
20	Liste
21	OR-Liste
22	Schlüsselwort (Keyword)
23	reserviert für internen Gebrauch
24	VSN
25	X-Text (X-Text)
26	Festpunktzahl (Fixed)
27	Gerät (Device)
28	Produkt-Version (Product-Version)
29	POSIX-Pfadname (Posix-Pathname)
35	POSIX-Dateiname (Posix-Filename)

Globale Syntaxattribute

Die Beschreibung der globalen Syntaxattribute eines Operandenwerts steht im dritten Byte der Wertbeschreibung. Es handelt sich dabei um Attribute, die für mehrere Datentypen festgelegt werden können (siehe ADD-VALUE, Seite 168ff).

Globale Attribute sind immer Ausgabe-Attribute. Sie werden ignoriert, falls das Programm Default-Werte liefert, oder in Korrekturdialogen.

Bit	Bedeutung, wenn das Bit gesetzt ist
0	Wert ist ein Wildcard-Suchmuster
1	Wert ist ein Wildcard-Konstruktor
2 bis 7	reserviert

Datentypspezifische Syntaxattribute

Die Beschreibung der datentypspezifischen Syntaxattribute steht im vierten Byte der Wertbeschreibung. Es handelt sich dabei um Attribute, die nur für einen Datentyp oder wenige Datentypen festgelegt werden können.

Datentypspezifische Attribute sind ebenfalls Ausgabe-Attribute. Sie werden ignoriert, falls das Programm Default-Werte liefert, oder in Korrekturdialogen.

Datentyp FILENAME / PARTIAL-FILENAME:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	Dateiname enthält catid
1	Dateiname enthält userid
2	Dateiname enthält Dateigeneration oder Dateigenerationsgruppe
3	Dateiname enthält Version
4	Dateiname ist temporär
5 bis 7	reserviert

Datentyp NAME:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	Name enthält Unterstrich (_)
1 bis 7	reserviert

Datentyp COMPOSED-NAME:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	zusammengesetzter Name enthält Unterstrich (_)
1	zusammengesetzter Name enthält catid
2 bis 7	reserviert

Datentyp TEXT:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	Text enthält Trennzeichen
1 bis 7	reserviert

Datentyp X-TEXT:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	X-Text hat eine ungerade Anzahl von Bytes
1 bis 7	reserviert

Datentyp POSIX-PATHNAME / POSIX-FILENAME:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	POSIX-Pfad- oder Dateiname ist absolut
1	POSIX-Pfad- oder Dateiname ist relativ
2	POSIX-Pfad- oder Dateiname wurde in Hochkommas eingegeben
3 bis 7	reserviert

Datentyp C-STRING:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	C-String enthält ein Hochkomma
1 bis 7	reserviert

Datentyp PRODUCT-VERSION:

Bit	Bedeutung, wenn das Bit gesetzt ist
0	Produktversion enthält Korrekturstand
1	Produktversion enthält Freigabestand
2 bis 7	reserviert

Kopffeld einer Strukturbeschreibung

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	**fehlerhafte Anweisung zurück an SDF**	**Default-Werte an SDF**
0 bis 1	Anzahl der Positionen im Operanden-Array	SDF	unverändert	Programm
2 bis 3	reserviert	–	–	–
4 bis 7	Wertbeschreibung für den strukturein-leitenden Operandenwert			
4	Zusatzinformation	–	siehe Seite 377	siehe Seite 377
5	Typbeschreibung	SDF	unverändert	Programm
6	Globale Syntaxattribute	SDF	–	–
7	Typspezifische Syntaxattribute	SDF	–	–
8 bis 11	Absolutadresse (aligned abgelegt) des struktureinleitenden Operandenwerts	SDF	unverändert	Programm

Die Anzahl der Positionen in dem Operanden-Array ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-VALUE...,STRUCTURE=*YES(...,MAX-STRUC-OPERAND=...).

Das zu der Struktur gehörende Operanden-Array beginnt unmittelbar hinter dem Kopffeld (siehe Bild 9 auf Seite 374). Es ist genau so aufgebaut wie das für anweisungsglobale Operanden.

Listenelement

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	**fehlerhafte Anweisung zurück an SDF**	**Default-Werte an SDF**
0 bis 3	Wertbeschreibung			
0	Zusatzinformation	siehe Seite 377	siehe Seite 377	siehe Seite 377
1	Typbeschreibung	SDF	unverändert	Programm [1)]
2	Globale Syntaxattribute	SDF	–	–
3	Typspezifische Syntaxattribute	SDF	–	–
4 bis 7	Absolutadresse (unaligned abgelegt) des zu dem Listenelement gehörenden Werts bzw. bei Strukturen der weiteren Beschreibung	SDF	unverändert	Programm [1)]
8 bis 11	Absolutadresse (aligned abgelegt) des nächsten Listenelements	SDF	unverändert	Programm [1)]

[1)] Angabe nur für Operanden, zu denen umzusetzende Operandenwerte gehören, d.h. wenn Bit 0 der Zusatzinformation gesetzt ist.

Im letzten Element der Liste hat die „Absolutadresse des nächsten Listenelements“ den Wert 0 (siehe Bild 10 auf Seite 375).

Eine OR-Liste besteht nur aus einem Element. Bei diesem Element entfällt die „Absolutadresse des nächsten Listenelements“.

Für einen Operanden, der mit LIST-POSSIBLE=*YES(FORM=*NORMAL) definiert ist, muss die Anzahl der Listenelemente begrenzt werden (LIMIT=...), damit es nicht zum Überlauf des normierten Übergabebereiches kommt. Die Größe einer Liste im normierten Übergabebereich kann nach folgender Formel berechnet werden:

$$n * (10 + 2 + l)$$

wobei: n die Anzahl der Listenelemente und

l die Länge eines einzelnen Listenelementes (aufgerundet auf ein Vielfaches von 2) ist.

Beispiel:

```
ADD-OPERAND ... LIST-POSSIBLE=*YES(LIMIT=100,FORM=*NORMAL)
  ADD-VALUE *NAME(1,8)
```

Eine so definierte Liste kann im normierten Übergabebereich bis zu 2000 Byte belegen (100 * (10 + 2 + 8) = 2000).

Ablage der Werte

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	**fehlerhafte Anweisung zurück an SDF**	**Default-Werte an SDF**
0 bis 1	Längenangabe	SDF	unverändert	Programm
2 bis 3	reserviert	–	–	–
4 bis ...	Wert	SDF	unverändert	Programm

Wie die Werte übergeben werden, hängt ab von der Definition in der Syntaxdatei (siehe ADD-VALUE...,OUTPUT=*NORMAL(...). Dabei gelten folgende Besonderheiten:

– Ein Wert, der mit ADD-VALUE TYPE=*INTEGER(...,OUT-FORM=*BINARY) definiert ist, wird als 4-Byte-String mit Vorzeichen abgelegt.

– Ein Wert, der mit ADD-VALUE TYPE=*TIME definiert ist, wird als 4-Byte-String mit 2 Bytes (binär) für Stunden und je 1 Byte für Minuten und Sekunden abgelegt.

6.4 SDF-Makros

6.4.1 Typen von Makroaufrufen

Der Eindeutigkeit wegen sei hier erwähnt, dass mit der Bezeichnung „Makroaufruftyp“ nicht die Begriffe „Aktionsmakro“ bzw. „Definitionsmakro“ gemeint sind. Diese Begriffe nehmen Bezug auf die Funktion eines Makros:

Ein **Aktionsmakro** ist ein Makro, von dem die Ausführung bestimmter Handlungen erwartet wird, wie z.B. ein Druckvorgang durch den **PRNT**-Makro gesteuert wird.

Ein **Definitionsmakro** ist ein Makro, von dem keine Handlungen, sondern Definitionen (Adressierungshilfen, DSECTS) erwartet werden. Der Makro **CUPAB** ist ein Beispiel für die Generierung von symbolischen Namen zur Adressierung von Operandentabellen.

In **Typen** werden die Makroaufrufe abhängig von der Art ihrer **Operandenübergabe** eingeteilt. Es gibt den R-Typ (Übergabe in Registern), den S-Typ (Übergabe im Speicher) und den sog. O-Typ (Makros ohne Typzuordnung).
Makros vom S-Typ können sowohl Aktionsmakros (MF=E) als auch Definitionsmakros (MF=D) sein.

R-Typ-Makroaufrufe

Operation	Operanden
makro	{ operand1 | (r1) } , { operand2 | (r2) }

Ein Makroaufruf ist von Typ R, wenn alle erforderlichen Operandenwerte in die zwei für diesen Zweck verwendeten Register R0 und R1 geladen werden können. Durch einen R-Typ-Makroaufruf wird also kein Parameterbereich generiert.

S-Typ-Makroaufrufe

Beim S-Typ werden die im Makroaufruf angegebenen Operandenwerte in Form eines Datenbereichs an den Funktionsbaustein übergeben. Der Datenbereich ist Teil der Makroauflösung. Er erhält die für die Übergabe der Operandenwerte notwendigen Daten- und Speicherdefinitionen (DC- und DS-Anweisungen).

Name	Operation	Operanden
[opadr]	makro	{ operand$_1$,..,operand$_n$[,MF=(S / L / C[,PREFIX=p],[MACID=mac] / (C,p) / D[,PREFIX=p] / (D,p) / M[,PREFIX=p],[MACID=mac]) / MF=((E,opadr) / (E,(r)) / E[,PARAM=adr]) }

Der S-Typ unterstützt die Angabe des Operanden MF. In Abhängigkeit von der Funktionalität der verschiedenen Makros sind mehrere Arten der Darstellung von MF möglich:

– Standardform (MF=S):

  MF=S ist Voreinstellung. Es werden zuerst der Befehlsteil und anschließend der Datenbereich generiert, unter Beachtung der im Makroaufruf angegebenen Operandenwerte. Der Datenbereich enthält keine Feldnamen und keine erläuternden Equates. Der Standardheader ist initialisiert.

– L-Form (MF=L):

  Es wird nur der Datenbereich generiert, unter Beachtung der im Makroaufruf angegebenen Operandenwerte. Der Datenbereich enthält keine Feldnamen und keine erläuternden Equates. Der Standardheader ist initialisiert. Der Makroaufruf wird gewöhnlich im Definitionsteil des Programms abgesetzt. Bei Shared-Code-Programmierung darf dieser Aufruf nicht im invarianten Programmteil liegen, wenn er variable Daten enthält. Der Datenbereich wird im invarianten Programmteil mit konstanten Werten initialisiert, vor dem E-Form-Aufruf in einen ablauflokalen Datenbereich kopiert und dort ggf. modifiziert. Die Modifizierung wird z.B. mit der M-Form realisiert, falls sie für die betreffende Schnittstelle angeboten wird.

– C-Form (MF=C / MF=(C,p)):

  Es wird nur der Datenbereich generiert; im Makroaufruf angegebene Operandenwerte werden nicht ausgewertet. Jedes Feld hat einen Feldnamen und erläuternde Equates, falls erforderlich. Die ersten Zeichen der Feldnamen können bestimmt werden (p = Präfix zur Ersetzung der ersten Zeichen). Der Datenbereich wird durch ein Längenequate beendet. Der Standardheader muss i.d.R. vom Anwender initialisiert werden.

– D-Form (MF=D / MF=(D,p)):

  Es wird eine DSECT generiert. Jedes Feld hat einen Feldnamen und erläuternde Equates, falls erforderlich. Die ersten Zeichen der Feldnamen können bestimmt werden (p = Präfix zur Ersetzung der ersten Zeichen). Die DSECT beschreibt die Struktur eines Speicherbereiches, ohne selbst Speicherplatz zu belegen. Sie wird durch ein Längenequate beendet. Es wird nicht auf den anfänglichen Adresspegel umgeschaltet. Der bei DSECT angegebene symbolische Name wird in einen ESD-Satz eingetragen. Der Adresspegel wird auf null gesetzt.

– E-Form:

  Es werden nur die zum Aufruf des Funktionsbausteins notwendigen Befehle generiert. Der Befehlsteil endet i.d.R. mit einem SVC. Im Makroaufruf muss die Adresse des Datenbereichs mit den Operandenwerten angegeben sein. Folgende Formen sind gebräuchlich:

MF=(E,adr)	adr = Adresse des Datenbereichs
MF=(E,(r))	r = Register mit dem Wert der Adresse des Datenbereichs
MF=E,PARAM=adr	adr = Adresse des Datenbereichs

  Weitere Operanden werden in der E-Form nicht ausgewertet.

– M-Form (MF=M)

  Es werden Befehle (z.B. MVCs) generiert, die während des Programmlaufs in einem mit MF=M bereits initialisierten Datenbereich bzw. bei Shared-Code-Programmierung in einer ablauflokalen Kopie des mit MF=L initialisierten Datenbereichs Felder mit den Operandenwerten überschreiben, die im Makroaufruf angegeben werden. Damit bietet die M-Form eine komfortable Möglichkeit, die Operandenwerte, mit denen ein Makro aufgerufen wird, dynamisch dem Programmlauf anzupassen.
  Da die dafür generierten Befehle die symbolischen Adressen und Equates der C-Form oder D-Form benutzen, ist bei der Verwendung der M-Form sicherzustellen, dass diese Namen für die Adressierung der zu modifizierenden Operandenliste zur Verfügung stehen. Insbesondere ist darauf zu achten, dass bei einem Makroaufruf mit MF=M die ggf. Operanden PREFIX und MACID mit den gleichen Werten angegeben werden wie im zugehörigen MF=C- bzw. MF=D-Aufruf.

Eine ausführliche Beschreibung der MF-Formen finden Sie im Handbuch „Makroaufrufe an den Ablaufteil“ [8].

O-Typ-Makroaufrufe

Eine Reihe von Makroaufrufen kann weder dem R-Typ noch dem S-Typ zugeordnet werden. Sie sind Makroaufrufe ohne Typ-Zuordnung.

Vom O-Typ sind z.B. jene Makroaufrufe, die im Operandenfeld die Angabe **eines** Registers vorsehen (häufig nur R1), das die Anfangsadresse einer Operandenliste enthält.

Die Operandenliste wird im Datenteil des Programms definiert (DC-Anweisungen) und enthält die Operandenwerte.

6.4.2 Standardheader

Alle neuen Makros und i.d.R. die Makros, die um die 31-Bit-Schnittstelle erweitert wurden, benutzen zur Identifikation ihrer Schnittstelle den Standardheader.
Der Standardheader ist ein 8 Byte langes Feld am Anfang der Operandenliste (Parameterliste) mit der (normierten) Bezeichnung der Schnittstelle und 4 Byte zur Aufnahme eines Returncodes. Der Standardheader wird vom jeweiligen Makro erzeugt und initialisiert, d.h. mit den gültigen Werten für UNIT, FUNCTION und VERSION versorgt. Bei Makroaufrufen in der E-Form unter Bezug auf die Operandenliste muss unter Umständen der Aufrufer den Standardheader initialisieren. Näheres ist beim jeweiligen Makro angegeben.

Aufbau des Standardheaders:

Byte	Feldinhalt und Bedeutung
0 - 1	Bezeichnung der Funktionseinheit (UNIT) mit der verlangten Funktion
2	Bezeichnung der Funktion (FUNCTION) innerhalb der Funktionseinheit
3	Bezeichnung des Änderungsstandes (VERSION) der Funktion
4	Unterwert 2 des Returncodes (SC2)
5	Unterwert 1 des Returncodes (SC1)
6 - 7	Hauptwert des Returncodes (Maincode)

Folgende Werte des Returncodes sind Konvention:

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	Erfolgreiche Funktionsausführung. Es gibt keine zusätzlichen Informationen zum Maincode.
01	00	0000	Erfolgreiche Funktionsausführung. Es waren keine weiteren Aktionen erforderlich.
00	01	FFFF	Die angeforderte Funktion wird nicht unterstützt (falsche Angabe für UNIT oder FUNCTION im Standardheader). Nicht behebbarer Fehler.
00	02	FFFF	Die angeforderte Funktion ist nicht verfügbar. Nicht behebbarer Fehler.
00	03	FFFF	Die angegebene Version der Schnittstelle wird nicht unterstützt (falsche Versionsangabe im Standardheader). Nicht behebbarer Fehler.
00	04	FFFF	Parameterliste nicht auf Wortgrenze ausgerichtet.
00	41	FFFF	Das Subsystem ist nicht vorhanden; es muss explizit erzeugt werden.
00	42	FFFF	Die aufrufende Task ist mit dieser Schnittstelle nicht konnektiert; sie muss explizit konnektiert werden.
00	81	FFFF	Subsystem zurzeit nicht verfügbar.
00	82	FFFF	Subsystem im DELETE- oder HOLD-Zustand.

Der Maincode kennzeichnet das Ergebnis der Funktionsausführung. SC1 dient der Klassifizierung des Hauptwertes. SC2 dient der weiteren Unterteilung des Fehlers in Fehlerklassen oder enthält zusätzliche Diagnoseinformationen.

Bei allen ab BS2000 V9.0 neu eingeführten Makros sollte der Returncode ausschließlich im Standardheader übergeben werden. In einer Übergangsphase kann der Returncode bei manchen Makroschnittstellen aber auch im Register R15 oder sowohl im Standardheader als auch im Register R15 übergeben werden. Um zu prüfen, ob im Standardheader ein Returncode übergeben wurde, sollte das Returncode-Feld mit X'FFFFFFFF' vorbesetzt werden.

6.4.3 Metasyntax für die Makro-Aufrufformate

Bei der Darstellung der Makroaufrufformate werden bestimmte Metazeichen verwendet und Vereinbarungen getroffen, die in der folgenden Tabelle erläutert sind:

Formale Darstellung	**Erläuterung**	**Beispiel**
GROSS-BUCHSTABEN	Großbuchstaben bezeichnen Schlüsselwörter oder Konstanten, die in dieser Form vom Benutzer angegeben werden müssen. Schlüsselwörter müssen mit * beginnen, falls alternativ sowohl Schlüsselwörter als auch Namen von Konstanten oder Variablen angegeben werden können.	`DATATYPE=CATID` Einzugeben ist: `DATATYPE=CATID`

Kleinbuch-staben	Kleinbuchstaben bezeichnen Variablen, die bei der Eingabe vom Benutzer durch aktuelle Werte ersetzt werden müssen, d.h. ihr Inhalt kann von Fall zu Fall verschieden sein.	`SHORTST=integer`
		Einzugeben ist:
		`SHORTST=10` oder
		`SHORTST=54`
		usw.

{ }	Geschweifte Klammern schließen Alternativen ein, d.h. aus den eingeschlossenen Größen muss eine Angabe ausgewählt werden.	`CCSNAME={ *NO / *EXTEND / name }`
		Einzugeben ist:
		`CCSNAME=*NO` oder
		`CCSNAME=*EXTEND` oder
		`CCSNAME=xxxxx`

/	Ein Schrägstrich trennt alternative Operandenwerte, von denen einer ausgewählt werden muss.	`SYMTYP=CSECT/ENTRY`
		Einzugeben ist:
		`SYMTYP=CSECT` oder
		`SYMTYP=ENTRY`

<u>Unterstreichung</u>	Die Unterstreichung hebt den Default-Wert (Voreinstellung) hervor. Das ist der Wert, den das System einsetzt, wenn der Benutzer keine Angabe macht. Hat ein Operand keinen Default-Wert, so ist die Angabe des Operanden Pflicht.	`PATTERN=`<u>`*NO`</u>` / addr`
. . .	Punkte bedeuten eine Wiederholung. Sie zeigen an, dass die davor stehende Einheit mehrmals hintereinander wiederholt werden kann.	`(modul,...)` Einzugeben ist: `(MODUL1)` oder `(A,B,C)` oder `(A1,A2,A3,A4)` usw.

Tabelle 4: Elemente der Metasyntax (Teil 1 von 2)

Formale Darstellung	**Erläuterung**	**Beispiel**

list-poss(n)	Aus den list-poss(n) folgenden Werten kann eine Liste gebildet werden. Ist (n) angegeben, können maximal n Elemente in der Liste vorkommen. Bei mehr als einem Listen-Element muss die Liste in runde Klammern eingeschlossen sein.	`DEVCLAS=list-poss(2): DISK / TAPE`
		Einzugeben ist:
		`DEVCLAS=DISK` oder
		`DEVCLAS=TAPE` oder
		`DEVCLAS=(DISK,TAPE)`

[ ]	Eckige Klammern schließen Wahlangaben ein, d.h. Angaben, die man weglassen darf. Steht bei Wahlangaben das Komma innerhalb der Klammer, so wird es nur bei Verwendung dieser Wahlangabe verlangt und kann beim ersten Operanden weggelassen werden. Steht es außerhalb der Klammer, so muss es stets geschrieben werden, auch wenn keine Wahlangabe gemacht wird.
	(Runde Klammern müssen eingegeben werden.)
		`MAP=S[YSOUT]`
		
		Einzugeben ist:
		`MAP=SYSOUT`
		oder abgekürzt:
		`MAP=S`

< >	Spitze Klammern kennzeichnen den Datentyp des Operanden.	`INOUT=<var:pointer>`
=	Das Gleichheitszeichen verbindet den Operandennamen mit den dazugehörenden Operandenwerten.	`SPIN=*NO / *YES`

Tabelle 4: Elemente der Metasyntax (Teil 2 von 2)

Datentypen der Operandenwerte

Datentyp	Zeichenvorrat	Besonderheiten
c-string	EBCDIC-Zeichen	ist in Hochkommata einzuschließen und ohne einleitendes "C" anzugeben. Hochkommata in der Zeichenkette sind doppelt anzugeben. Eine inhaltliche Bedeutung der Angabe wird anschließend in SDF-Schreibweise angegeben, abgetrennt durch Doppelpunkt. Der Zusatz n..m beschreibt die Eingabelänge in Bytes. Beispiel: im Syntaxdiagramm: `SELECT = <c-string 1..255>` bei der Eingabe: `SELECT='testfile'`
x-string	Sedezimal 00..FF	Ist in Hochkommata einzuschließen, und der Buchstabe X muss vorangestellt werden: X'xxxx'. Der Zusatz n..m beschreibt die maximale Eingabelänge in Bytes. Beispiel: im Syntaxdiagramm: `PASSWORD = <x-string 1..10>` bei der Eingabe in Assembler: `PASSWORD=X'FF00AA1122'` bei der Eingabe in C/C++: `PASSWORD=0xFF00AA1122`
name	A..Z, 0..9, $, #, @	Bezeichner. Das Format ist der jeweiligen Operandenbeschreibung zu entnehmen. Beispiel: im Syntaxdiagramm: `PARAM = <name 1..8>` bei der Eingabe: `PARAM=MYPARAM`

label	A..Z
	0..9
	$,#,@
		Kennzeichnet eine Marke.
		Beispiel:
		`OUTAREA=structure (2):`
		`(1) address: <label>`

integer	0..9,+,-	"+" bzw. "-" kann nur erstes Zeichen sein. Der Zusatz n..m beschreibt den zulässigen Wertebereich. Beispiel: im Syntaxdiagramm: `MAXLEN = <integer 1..255>` bei der Eingabe: `MAXLEN=200`
var:		Leitet eine variable Angabe ein. Nach dem Doppelpunkt folgt der Datentyp der Variablen (siehe folgende Tabelle). Beispiel: im Syntaxdiagramm: `SELECT = <var: char:255>` bei der Eingabe (Angabe des Variablenbezeichners): `SELECT=VARSELECT`
reg:		Leitet eine Register-Angabe ein (Register 0..15). Nach dem Doppelpunkt folgt der Datentyp des Registerinhalts. Bei der Eingabe kann ein Register oder ein Register-EQUATE verwendet werden.

Tabelle 5: Datentypen der Operandenwerte

Datentypen der Variablen und Registerinhalte

Datentyp	Beschreibung
char:n	Kennzeichnet eine Zeichenkette der Länge n. Die Zeichenkette darf nur dann kürzer sein, wenn eine bestimmte Bedingung eingehalten wird. Die Bedingung ist der jeweiligen Operandenbeschreibung zu entnehmen. Fehlt die Längenangabe, wird n=1 angenommen.
int:n	Kennzeichnet eine Ganzzahl, die n Byte belegt, wobei n≤4 gilt. Fehlt die Längenangabe, wird n=1 angenommen.
bit:n	Kennzeichnet einen Bitstring der Länge n. Fehlt die Längenangabe, wird n=1 angenommen. (in C: Deklaration als 'unsigned')
enum-of E:n	Die Variable ist die Aufzählung E, die n Byte belegt, wobei n≤4 gilt. Fehlt die Längenangabe, wird n=1 angenommen.

pointer	Zeiger (die Adresse wird übergeben).
	Bereichsreferenzierung mit Hilfe von Zeigervariablen:
	– mit MF=L: Adressnotation:
	`...,MF=M,<operand>=A(area)`
	– mit MF=M: Registernotation, Adressnotation oder symbolische Adresse eines Feldes, das die Adresse eines Bereiches enthält:
	`LA 2,area`
	`... MF=M,<operand>=(2)`
	oder
	`... MF=M,<operand>=A(area)`
	oder
	`LA 2,area`
	`ST 2,areaADD`
	`... MF=M,<operand>=areaADD`
	`areaADD DS A`

Tabelle 6: Datentypen der Variablen und Registerinhalte

6.4.4 Funktionelle Übersicht

Datendefinitionen

CMDALLW	Liste zulässiger Operationen für die Makros CMDRST und CMDTST generieren
CMDANALY	EQUATES für Returncodes generieren
CMDMEM	Übergabebereich für Statusinformationen generieren
CMDRETC	DSECT für Kommando-Returncodes erzeugen
CMDTA	Übergabebereich für eine analysierte Anweisung generieren

Anweisungen bearbeiten

CMDCST	Semantikfehlerdialog anstoßen
CMDRST	Anweisung lesen und analysieren
CMDTST	Anweisung analysieren

Die Makros CMDTA, CMDCST, CMDRST und CMDTST erfordern ASSEMB-H.

Aufrufe in gleichzeitig eröffneten Syntaxdateihierarchien

OPNCALL	Programmkontext erstellen
CLSCALL	Programmkontext schließen

Sonstige

CMDRC	Kommando-Returncodes setzen
CMDSEL	Auswahlmaske für den geführten Dialog erzeugen
CMDSTA	Informationen über aktivierte Syntaxdateien ausgeben
CMDVAL	Eingabewert auf Datentyp oder auf Übereinstimmung mit einem Wildcard-Suchmuster überprüfen
CMDWCC	Syntax von Wildcard-Suchmustern prüfen und Mustervergleich durchführen
CMDWCO	Mit Hilfe von Wildcard-Konstruktoren neue Namen erzeugen
TRCMD	Kommando analysieren

6.4.5 Beschreibung der Makros

Die Beschreibung der Makros erfolgt alphabetisch geordnet.

CLSCALL
Programmkontext schließen

CLSCALL schließt den mit OPNCALL erstellten Programmkontext. Der angegebene Kontext-Identifikator ist nicht mehr gültig.

Operation	Operanden
CLSCALL	CALLID = addr / reg [,MF = { L / (E,(1)) / (E,opadr) }]

CALLID = addr / (r)
Adresse eines 4 Byte langen Feldes oder Register, das den Kontext-Identifikator enthält.

MF =
definiert besondere Anforderungen an die Makroauflösung (Einzelheiten siehe Seite 385ff und Handbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Registerverwendung

Register 1: Adresse der Parameterliste

Rückinformation und Fehleranzeigen

Register 15 enthält im rechtsbündigen Byte einen Returncode:

X’00’ normale Ausführung

X’08’ callid nicht vorhanden

X’40’ callid in falscher Umgebung

CMDALLW
Liste der zulässigen Operationen generieren

Der Makro CMDALLW generiert eine Liste der zulässigen Operationen, die beim Aufruf der Makros CMDRST und CMDTST benutzt werden kann, wenn in diesen Makros für den Parameter STMT=A(identifier) angegeben wird.

Operation	Operanden
CMDALLW	LIST = (name, ...) ,SIZE = 1 / i ,PREFIX = C / p ,MACID = MDA / mac ,MF = L / D / C

LIST = (name,...)
gibt die Liste der zulässigen Anweisungen in den Makros CMDRST und CMDTST an. Das Layout wird so generiert, wie es beim Parameter STMT in diesen Makros notwendig ist. Nur die Anweisungen, deren interner Anweisungsname angegeben ist, sind zulässig. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT). Er ist mindestens ein und maximal acht Byte lang. Die SDF-Standardanweisungen sind unabhängig von der hier getroffenen Festlegung immer zulässig.

SIZE = 1 / i
gibt die Anzahl der Namen an, die in die Liste der zulässigen Operationen eingetragen werden können. i muss ganzzahlig und größer als null sein.

Beschreibung der Parameter PREFIX, MACID und MF siehe Seite 385ff (Typen von Makroaufrufen).

Mögliche Kombinationen

```
[label]  CMDALLW MF=D[,SIZE=i][,PREFIX=p][,MACID=mac]
[label]  CMDALLW MF=C[,SIZE=i][,PREFIX=p][,MACID=mac]
[label]  CMDALLW MF=L[,LIST=(name,...)]
```

Rückinformation und Fehleranzeigen

Der Makro liefert keinen Returncode.

Beispiele

– Speicherbereich für maximal 20 Namen anfordern:

```
ALLC  CMDALLW MF=C,SIZE=20
```

– Modell für die Liste der zulässigen Anweisungen in einer DSECT deklarieren:

```
ALLD  CMDALLW MF=D,PREFIX=D
```

– Deklaration einer Liste von beispielsweise 20 Namen im statischen Code. Die Länge des Speicherbereiches wird hier durch die Anzahl der Listenelemente festgelegt.

```
ALL20 CMDALLW LIST=(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T),-
      PREFIX=I
```

– Der Speicherbereich kann durch den statischen Code initialisiert werden. Die Weiterverarbeitung erfolgt mit Verweis auf die DSECT:

```
MVC     ALLC(IMDALL#),ALL20
L       R5,ALLC
USING   ALLD,R5
MVC     DMDALNR,=H'15'
MVC     DMDAL1+10*DMDALNL,=CL8'V'
...
```

(IMDALL#, DMDALNR, DMDAL1 und DMDALNL wurden durch CMDALLW mit PREFIX=I und D erzeugt)

– Aufruf mit statischen Listen:

```
        ...
        LA 2,MYLIST
        LA 1,RST1
        USING RSTD,1
        CMDRST MF=M,PARAM=RST1,STMT=(2)     * Update parameter area RST1
                                            * using register notation
        ...
RST1    CMDRST MF=L,STMT=A(MYLIST)          * direct initialization of
                                            * parameter area RST1
MYLIST  CMDALLW LIST=(A,B,C)

RSTD    CMDRST MF=D
```

CMDANALY
EQUATES für Returncodes generieren

Der Makro CMDANALY generiert eine Folge von EQUATE-Anweisungen. Diese beschreiben die Returncodes für die Makros CMDSTA, CORSTMT, RDSTMT und TRSTMT. Für die Makros CMDRST, CMDTST und CMDCST ist das nicht notwendig (siehe z.B. „Migration von RDSTMT zu CMDRST“ auf Seite 422).

Operation	Operanden
CMDANALY	P = CMD / mac

P = CMD / mac
Präfix des Makro-Identifikators. Standardmäßig beginnt der Makro-Identifikator mit der Zeichenfolge „CMD“.

Hinweise zu einigen Returncodes

1. DIALOGUE_IMPOSSIBLE_:
   Zur Korrektur der aktuellen Eingabe konnte kein Korrekturdialog bereitgestellt werden:
   - die Eingabe war unbekannt
   - Stapelauftrag
   - Spin-off-Zustand
   - Prozedurmodus bei Angabe PROCEDURE-DIALOG=*NO
2. DIALOGUE_REJECTED_:
   Ein Korrekturdialog konnte zwar gestartet werden, aber die verlangte Korrektur wurde vom Aufrufenden im geführten oder ungeführten Modus zurückgewiesen.

CMDCST
Semantikfehlerdialog anstoßen

Der Makro CMDCST bewirkt, dass SDF mit dem Benutzer einen Dialog führt, in dem dieser Semantikfehler in einer Anweisung korrigiert. SDF hat die Anweisung unmittelbar zuvor analysiert und als syntaktisch richtig an das Programm übergeben. Verdeckt eingegebene Operandenwerte muss der Benutzer während des Korrekturprozesses noch einmal eingeben.

Voraussetzungen für den Semantikfehlerdialog sind:

- Das Programm läuft in einer interaktiven Task und bei der Syntaxanalyse war ein Fehlerdialog zugelassen. Das bedeutet im Einzelnen:
  - Temporär oder permanent geführter Dialog musste eingestellt sein.
  - In Prozeduren musste die SDF-Option PROCEDURE-DIALOGUE=*YES eingestellt sein.
  - Wenn CMDCST nach CMDTST aufgerufen wird, musste bei CMDTST DIALOG=*ERROR eingestellt sein.
- Es steht die gleiche Syntaxdatei wie bei der ersten Analyse der Anweisung zur Verfügung (kein zwischenzeitlicher Wechsel der Syntaxdatei).

Sind diese Voraussetzungen nicht erfüllt, lehnt SDF den Dialog ab (Fehlercode X'20').

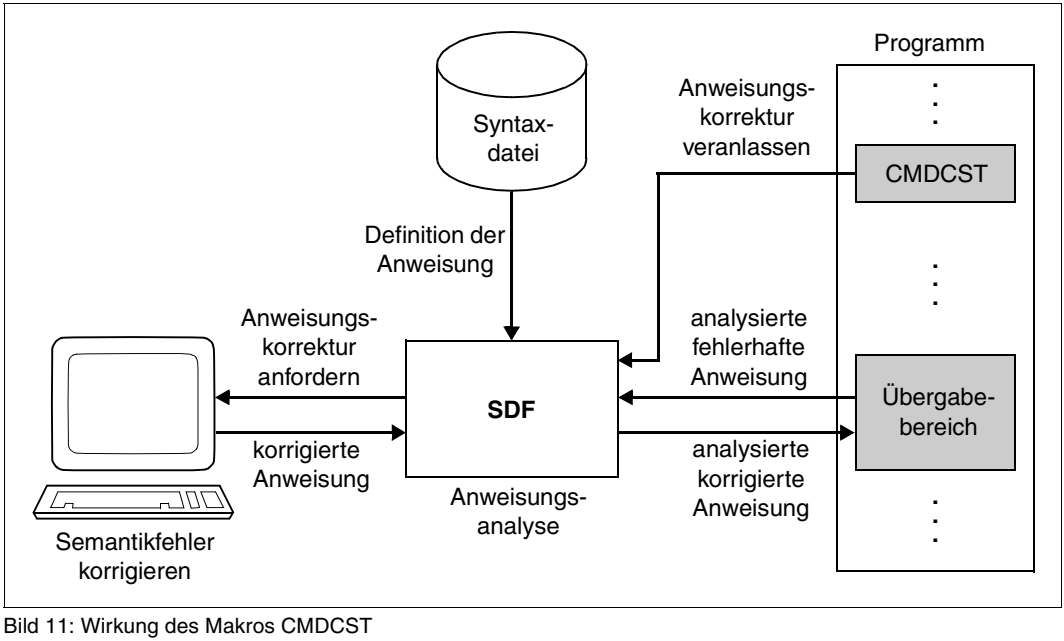


Bild 11: Wirkung des Makros CMDCST

Operation	Operanden
CMDCST	INOUT = <var: pointer> ,MESSAGE = <var: pointer> ,DEFAULT = *NO / <var: pointer> ,INVAR = *NO / <var: pointer> ,CALLID = *NO / <var: pointer> ,CCSNAME = *NO / *EXTEND / <c-string 1..8> / <var: char:8> ,MF = C / D / L / M / E

INOUT = <var: pointer>
Adresse des normierten Übergabebereichs, der auf Wortgrenze beginnen muss. In ihm steht das zuvor von SDF an das Programm übergebene Analyseergebnis der fehlerhaften Anweisung. Das Programm hat die von ihm als fehlerhaft erkannten Operandenwerte gekennzeichnet. Änderungen von Eingabewerten durch das Programm übernimmt SDF nicht. Nach dem Fehlerdialog und der erneuten Analyse legt SDF das geänderte Analyseergebnis wieder in diesem Bereich ab (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff).

MESSAGE = <var: pointer>
Adresse des auszugebenden Textes für den Fehlerdialog. Dieser Text wird im geführten Dialog in das Anweisungsmenü integriert. Der Textbereich muss auf Halbwortgrenze ausgerichtet sein und folgendes Format haben:

2 Byte: absolute Länge des Satzes (n+4)
2 Byte: (reserviert)
n Byte: Meldungstext

Der Text darf maximal 400 Zeichen lang sein. In SDF-formatierten Bildschirmen werden jedoch nur die ersten 280 Zeichen dargestellt. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden.

DEFAULT =
bestimmt, ob SDF folgende Werte durch vom Programm dynamisch erzeugte Werte ersetzt:
– eingegebene Operandenwerte oder
– Default-Werte der Operanden

In der Syntaxdatei müssen die Operanden bzw. Operandenwerte entsprechend definiert sein (siehe ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES,... bzw. ADD-VALUE..., VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). Der vom Programm erzeugte Wert muss gültiger Operandenwert sein.

Im geführten Dialog zeigt SDF die vom Programm erzeugten Werte im Fragebogen.

Beispiel:

SDF-A ersetzt in den eingegebenen MODIFY-Anweisungen den Wert *UNCHANGED durch den aktuellen Wert.

***NO**

Die angegebenen Operandenwerte werden nicht durch vom Programm dynamisch erzeugte Werte ersetzt.

<var: pointer>

Adresse einer auf Wortgrenze ausgerichteten Liste, die Adressen von Umsetzbeschreibungen für Anweisungen enthält. Als Umsetzbeschreibung wird ein formatierter Übergabebereich mit dem Typ 'Struktur' verwendet (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff). Je Anweisung kann nur eine Umsetzbeschreibung angegeben werden. Eine Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche Operanden mit welchen Werten zu belegen sind. Die Liste der Adressen von Umsetzbeschreibungen ist wie folgt aufgebaut:

2 Byte: Anzahl der Umsetzbeschreibungen in der Liste (n)
2 Byte: (reserviert)
4 Byte: Adresse der ersten Umsetzbeschreibung
. . .
4 Byte: Adresse der n-ten Umsetzbeschreibung

Die Bereiche für die Umsetzbeschreibungen, die für die Default-Werte des Programms übergeben werden, müssen auf Wortgrenze ausgerichtet sein. Dies gilt ebenso für den Ausgabebereich der Makros (OUTPUT-Operand).

Stehen die zu defaultierenden Operanden in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten: Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit zu defaultierenden Operanden hängt. Auf der anderen Seite gibt der Anwender ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit zu defaultierenden Operanden hängt. SDF versucht zunächst, die vom Benutzer eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen. Ist über den struktureinleitenden Wert keine eindeutige Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Benutzer den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

INVAR =
Legt fest, ob die INVARIANT-INPUT-Form der Anweisung abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task zu dieser Zeit erlaubt sind. Die INVARIANT-INPUT-Form ist damit die größtmögliche Eingabeform für eine Anweisung, die für einen Benutzer mit bestimmten Privilegien und im gewählten Dialogmodus zulässig ist.

Im Gegensatz zur Protokollierungsform LOGGING=*INVARIANT-FORM (Anweisung MODIFY-SDF-OPTIONS) werden Kennwörter und geheime Operanden jedoch **nicht** ausgeblendet.

***NO**
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

<var: pointer>
Gibt die Adresse eines Puffers an, in den SDF die INVARIANT-INPUT-Form der Anweisung schreibt. Der Puffer muss auf Wortgrenze ausgerichtet sein und das erste Halbwort muss die Länge des Puffers enthalten. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab. Der Puffer hat dann folgenden Inhalt:

2 Byte: maximale Länge des Puffers
2 Byte: Ausgabelänge, die SDF schreibt (n+4)
2 Byte: (reserviert)
n Byte: INVARIANT-INPUT-Form der Anweisung, ab 7. Byte

CALLID =
bezieht sich auf einen Kontext (=Syntaxdateihierarchie), der durch einen OPNCALL-Makro eröffnet wurde. Der Name der Syntaxdateihierarchie (callid) muss den 4 Byte langen Wert haben, der von SDF an das Feld zurückgegeben wird, welches durch den Operanden CALLID im Open-Context-Makro bezeichnet wurde.
Diese Funktion bezieht sich auf den Gebrauch der Makros OPNCALL und CLSCALL.

***NO**
Die aktuelle Syntaxdateihierarchie (Kontext) der Task wird für die Analyse der Anweisung verwendet.

<var: pointer>
Adresse des Aufrufprüfungsfeldes. Der Bereich muss auf Wortgrenze ausgerichtet sein.

CCSNAME =
Gibt den Namen des Zeichensatzes an, der für den Korrekturdialog auf 8-bit-Terminals und für die Konvertierung von Klein- in Großbuchstaben verwendet wird. Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Jeder codierte Zeichensatz wird durch seinen Namen (Coded Character Set Name, CCSN) bestimmt (siehe Handbuch „XHCS“ [11]). Die Ausgabe von Meldungen wird durch diesen Parameter nicht beeinflusst.

***NO**
Der 7-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

***EXTEND**
Der 8-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

<c-string 1..8> / <var: char:8>
Gibt den Namen eines speziellen 8-bit-Code an, der für Ein-/Ausgabeoperationen verwendet wird. Der Name muss 8 Byte lang sein und kann als C-String-Konstante oder als String-Variable übergeben werden.

Beschreibung der Parameter MF, PARAM, MACID und PREFIX siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8].

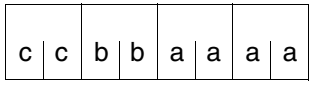
Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 371ff beschrieben. Das bis SDF V4.0 verwendete Format des Übergabebereiches finden Sie im Kapitel „Anhang“ auf Seite 605.

Hinweise zur INVARIANT-INPUT-Form einer Anweisung finden Sie beim Makro CMDRST auf Seite 421.

Der Returncode wird im Standardheader der Parameterliste übergeben.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	erfolgreiche Beendigung
00	20	0004	nicht behebbarer Systemfehler
	01	0008	Parameterfehler:
00			– falsche Parameterliste
01			– INOUT
03			– DEFAULT
04			– MESSAGE
06			– INVAR
07			– CALLID
00	40	000C	Übergabebereich zu klein
00	40	0010	Eingabeende (EOF) oder Anweisung fehlerhaft, danach wurde Eingabeende (EOF) erkannt
00	40	0014	Anweisung fehlerhaft, danach wurde Kommando erkannt
00	40	0018	Anweisung ist zwar in Ordnung, die vom Programm übergebenen Default-Werte sind jedoch fehlerhaft
00	40	001C	Anweisung fehlerhaft, danach wurde //STEP erkannt
00	40	0020	Fehlerdialog nicht möglich
00	40	0024	Fehlerdialog vom Benutzer abgelehnt
00	40	002C	END-Anweisung wurde gelesen
00	40	0034	Anweisung fehlerhaft, danach wurde END erkannt
00	40	0040	angegebene CALLID nicht gefunden
00	40	0044	im DSSM-Katalog eingetragene Syntaxdatei nicht gefunden
00	40	005C	INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten
00	20	0064	XHCS-Fehler bei Anweisungseingabe

Migration von CORSTMT zu CMDCST

Die Migration von CORSTMT nach CMDCST ist nur dann notwendig, wenn Sie die neue Funktionalität von CMDCST, CMDRST und CMDTST nutzen wollen. Dabei sind dieselben Punkte wie bei CMDRST zu beachten (siehe „Migration von RDSTMT zu CMDRST“ auf Seite 422ff).

Der Makro-Returncode wird im Standardheader der Parameterliste übergeben. Der Maincode von CMDCST entspricht den Werten, die bei CORSTMT im rechtsbündigen Byte im Register 15 übergeben wurden. Die Erzeugung von EQUATES für Returncodes mit dem Makro CMDANALY ist nicht mehr notwendig, da diese mit CMDCST MF=D automatisch erzeugt werden. Im Folgenden finden Sie eine Gegenüberstellung der alten (CORSTMT-) und neuen (CMDCST-)Feldnamen:

CORSTMT	CMDCST
&P.NOERR	&PREFIX.MDCSUCCESSFUL
&P.SYERR	&PREFIX.MDCSYSTEM_ERROR
&P.PAERR	&PREFIX.MDCPARAMETER_ERROR
&P.TRUNC	&PREFIX.MDCAREA_TOO_SMALL
&P.EOF	&PREFIX.MDCEOF
&P.SCMD	&PREFIX.MDCSTMTERROR_CMD
&P.DFLT	&PREFIX.MDCWRONG_DEFAULTS
&P.STEP	&PREFIX.MDCSTMTERROR_STEP
&P.DIMP	&PREFIX.MDCDIALOG_IMPOSSIBLE
&P.DREJ	&PREFIX.MDCDIALOG_REJECTED
&P.END	&PREFIX.MDCEND_STMT
&P.EERR	&PREFIX.MDCSTMTERROR_END
&P.NOPRG	&PREFIX.MDCPROGRAM_NOT_IN_SYNTAX_FILE
&P.INCID	&PREFIX.MDCINVALID_CALLID
&P.NOFND	&PREFIX.MDCSYNTAX_FILE_NOT_FOUND
&P.ITRC	&PREFIX.MDCINVARIANT_INPUT_TRUNCATED
&P.XHCS	&PREFIX.MDCXHCS_ERROR

CMDMEM
Übergabebereich für Statusinformationen generieren

Der Makro CMDMEM generiert eine DSECT oder eine CSECT. Diese beschreibt den Übergabebereich, in den nach Aufruf des Makros CMDSTA Informationen über die aktivierten Syntaxdateien und die geltenden Festlegungen für die Kommando-/Anweisungseingabe und -verarbeitung geschrieben werden.

Operation	Operanden
CMDMEM	D / C ,P = CMD / prefix

D / C
bestimmt, ob eine DSECT (D) oder eine CSECT (C) generiert wird.

P = CMD / prefix
bestimmt eine Zeichenfolge, die mit dem Anfang aller Namen in dem Programmabschnitt verkettet wird. Die Zeichenfolge darf maximal drei Zeichen lang sein. Standardmäßig wird die Zeichenfolge „CMD“ mit den Namen des Abschnitts verkettet.

Der Übergabebereich ist ab SDF V2.0A vergrößert. Der von SDF V4.0A gelieferte Makro CMDSTA benutzt dieses neue Layout. Der Übergabebereich, den der Aufrufende verwendet, muss mit dem Aufruf zusammenpassen, d.h. der Aufruf der Funktion und die Definition des Übergabebereichs müssen mit derselben SDF-Umgebung erstellt worden sein. Mißbrauch kann zu einem Parameterfehler führen, der vom Makro CMDSTA zurückgegeben wird. Frühere Aufrufe des Makros CMDSTA mit dem alten Übergabebereich werden unterstützt.

CMDRC
Kommando-Returncodes setzen

Mit dem Makro CMDRC kann das Benutzerprogramm eigene Werte als „Kommando-Returncode“ für das Programm sichern, die der Kommandoprozessor nach Programmbeendigung an Stelle des offiziellen Kommando-Returncodes zur Verfügung stellt. Die Werte können wie die Kommando-Returncodes des Programms interpretiert werden oder wie die der Kommandos, mit denen das Programm gestartet wurde (START- oder RESUME-PROGRAM). Dieser Kommando-Returncode kann dann von anderen Systemfunktionen (z.B. SDF-P) weiterverwendet werden.

Der durch CMDRC gesetzte Returncode ist nur bei Programmbeendigung verfügbar, da er bis zu diesem Zeitpunkt vom Kommandoprozessor zwischengespeichert wird. Es ist immer nur der letzte durch CMDRC gesetzte Returncode von Bedeutung. Während der Programmausführung wird der offizielle Kommando-Returncode im Kommandoprozessor durch einen CMDRC-Aufruf nicht verändert.

Operation	Operanden
CMDRC	RCADDR = addr [,MF = { L | (E,(1)) | (E,opadr) }]

RCADDR=addr
Adresse des Kommando-Returncodes.
Die DSECT für den Kommando-Returncode wird mit dem Makro CMDRETC erzeugt (siehe Seite 410).

MF =
definiert besondere Anforderungen an die Makroauflösung (Einzelheiten siehe Seite 385ff und Handbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Registerverwendung

Register 1: Adresse der Parameterliste

Rückinformation und Fehleranzeigen

Register 15 enthält im rechtsbündigen Byte einen Returncode:

X’00’ normale Ausführung

X’01’ abnormale Ausführung

Wird CMDRC nicht aufgerufen, so generiert SDF selbst die oben genannten gültigen Returncodes entsprechend des UNIT-Parameters des TERM-Makros.

CMDRETC
DSECT für Kommando-Returncodes erzeugen

Dieser Makro erzeugt eine DSECT oder CSECT für den Eingabebereich des Makros CMDRC.

Operation	Operanden
CMDRETC	PREFIX = C / p ,MACID = MDR / mac ,MF = D / C

Beschreibung der Parameter siehe Seite 385ff (Typen von Makroaufrufen).

CMDRST
Anweisung lesen und analysieren

Der Makro CMDRST bewirkt, dass SDF

– eine Programmanweisung von SYSSTMT einliest (Für die Systemdatei SYSSTMT gilt die gleiche Zuweisung, die für die Systemdatei SYSDTA getroffen ist. Bezüglich Folgezeilen, Fortsetzungszeichen und Angabe von Bemerkungen gelten für die Anweisungseingabe von SYSSTMT die gleichen Regeln wie für die Kommandoeingabe von SYSCMD.)

– die eingelesene Anweisung analysiert und

– das Analyseergebnis an das Programm übergibt.

Voraussetzung ist, dass eine aktivierte Syntaxdatei die Definition des Programms und seiner Anweisungen enthält.
Die Eingabelänge einer über CMDRST eingelesenen Anweisung beträgt 16348 Byte.

Neben dem normalen Lesen und Analysieren von Anweisungen kann CMDRST auch Datensätze lesen. Diese Funktionalität ist jedoch nur zu Migrationszwecken vorgesehen. SDF reicht die Datensätze nur durch, d.h. es wird keinerlei Formatierung darauf angewendet.

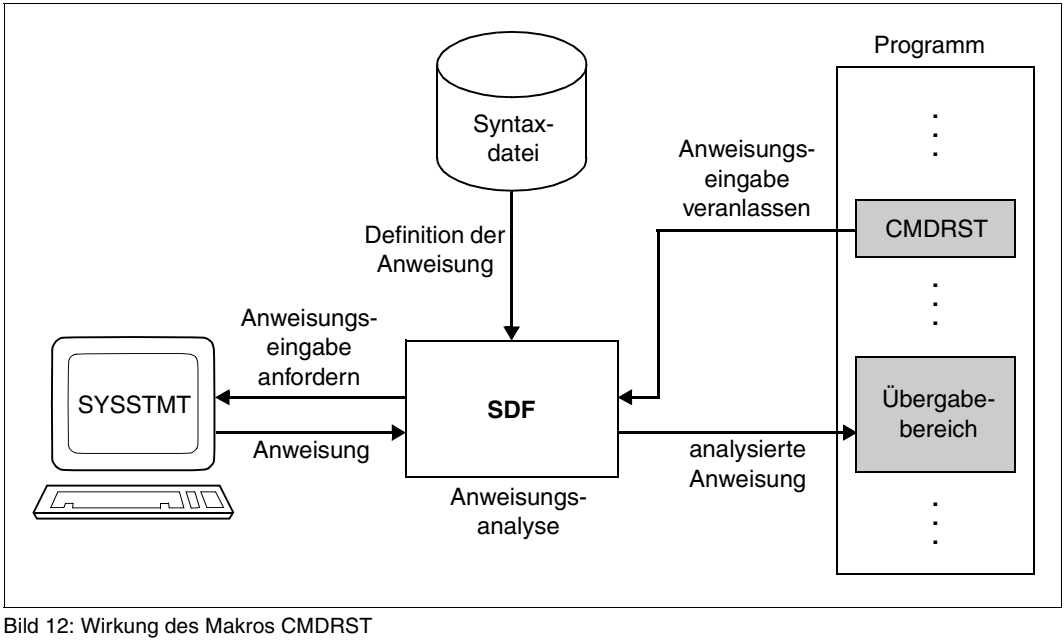


Bild 12: Wirkung des Makros CMDRST

Operation	Operanden
CMDRST	PROGRAM = <c-string 1..8> / <var: char:8>
	,OUTPUT = <var: pointer>
	,STMT = *ALL / <var: pointer>
	,PREFER = *ALL / <c-string 1..8> / <var: char:8>
	,DEFAULT = *NO / <var: pointer>
	,MESSAGE = *NO / <var: pointer>
	,PROT = *YES / *NO / <var: bit:1>
	,BUFFER = *NO / <var: pointer>
	,INVAR = *NO / <var: pointer>
	,SPIN = *NO / *YES / <var: bit:1>
	,ERRSTMT = *STEP / *NEXT / <var: bit:1>
	,CALLID = *NO / <var: pointer>
	,CCSNAME = *NO / *EXTEND / <c-string 1..8> / <var: char:8>
	,DATA_RECORD = *NO / <var: pointer>
	,STMTRC = *NO / <var: pointer>
	,OUTFORM = *NEW / *OLD / <var: bit:1>
	,MF = C / D / L / M / E

PROGRAM = <c-string 1..8> / <var: char:8>
interner Name des Programms, das den Makroaufruf absetzt. In der Syntaxdatei ist dieser Name in der Programmdefinition abgelegt (siehe ADD-PROGRAM). Er ist mindestens 1 und maximal 8 Byte lang und kann als C-String-Konstante oder als String-Variable übergeben werden.

OUTPUT = <var: pointer>
Adresse des normierten Übergabebereichs, der auf Wortgrenze beginnen muss. Der Übergabebereich wird mit dem Makro CMDTA generiert (oder bei OUTFORM=*OLD mit CMDSTRUC, siehe Kapitel „Anhang“ auf Seite 605).

STMT =
bestimmt, welche Anweisungen als Eingabe zulässig sind.
Nur die Anweisungen, deren interner Anweisungsname angegeben ist, sind zulässig. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT). Er ist mindestens ein und maximal acht Byte lang. Die SDF-Standardanweisungen sind unabhängig von der hier getroffenen Festlegung immer zulässig.

***ALL**
Alle Anweisungen sind zulässig.

<var: pointer>
Adresse der Liste der zulässigen Anweisungen. Diese Liste kann mit dem Makro CMDALLW generiert worden sein.
Die Liste muss auf Halbwortgrenze ausgerichtet und wie folgt aufgebaut sein:

2 Byte: Anzahl der internen Namen in der Liste (n)
8 Byte: erster interner Anweisungsname
. . .
8 Byte: n-ter interner Anweisungsname

PREFER =
ist nur für den geführten Dialog relevant und bestimmt, ob als nächste Eingabe eine bestimmte Anweisung erwartet wird.

***NO**
Es wird keine bestimmte Anweisung erwartet. SDF fragt mit dem Anweisungsmenü beim Benutzer ab, welche Anweisung er eingeben will.

<c-string 1..8>
interner Name der Anweisung, deren Eingabe mit hoher Wahrscheinlichkeit zu erwarten ist. SDF gibt kein Anweisungsmenü aus, in dem der Benutzer die einzugebende Anweisung auswählt, sondern direkt den Operandenfragebogen für die erwartete Anweisung. Der Benutzer kann allerdings statt der erwarteten eine andere Anweisung eingeben.

Beispiel:
Nach MODIFY-OPERAND erwartet SDF-A als nächste Anweisung MODIFY-VALUE. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT). Er ist mindestens 1 und maximal 8 Byte lang.

<var: char:8>
Adresse eines 8 Byte langen Bereiches, das den internen Namen der zu erwartenden Anweisung enthält. Der Name muss linksbündig ausgerichtet und mit Leerzeichen (X'40') aufgefüllt sein.

DEFAULT =
bestimmt, ob SDF folgende Werte durch vom Programm dynamisch erzeugte Werte ersetzt:
– eingegebene Operandenwerte oder
– Default-Werte der Operanden

In der Syntaxdatei müssen die Operanden bzw. Operandenwerte entsprechend definiert sein (siehe ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES),... bzw. ADD-VALUE..., VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). Der vom Programm erzeugte Default-Wert muss gültiger Operandenwert sein. Im geführten Dialog zeigt SDF die vom Programm erzeugten Werte im Fragebogen.

Beispiel:
SDF ersetzt in den eingegebenen MODIFY-Anweisungen den Wert *UNCHANGED durch den aktuellen Wert.

***NO**
SDF soll die eingegebenen Operandenwerte nicht durch vom Programm dynamisch erzeugte Werte ersetzen.

<var: pointer>
Adresse einer auf Wortgrenze ausgerichteten Liste, die Adressen von Umsetzbeschreibungen für Anweisungen enthält. Als Umsetzbeschreibung wird ein formatierter Übergabebereich mit dem Typ 'Struktur' verwendet (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff). Je Anweisung kann nur eine Umsetzbeschreibung angegeben werden. Eine Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche Operanden mit welchen Werten zu belegen sind. Die Liste der Adressen von Umsetzbeschreibungen ist wie folgt aufgebaut:

2 Byte: Anzahl der Umsetzbeschreibungen in der Liste (n)
2 Byte: (reserviert)
4 Byte: Adresse der ersten Umsetzbeschreibung
. . .
4 Byte: Adresse der n-ten Umsetzbeschreibung

Die Bereiche für die Umsetzbeschreibungen, die für die Default-Werte des Programms übergeben werden, müssen auf Wortgrenze ausgerichtet sein. Dies gilt ebenso für den Ausgabebereich der Makros (OUTPUT-Operand).

Stehen die zu defaultierenden Operanden in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten: Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit zu defaultierenden Operanden hängt. Auf der anderen Seite gibt der Anwender ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit zu defaultierenden Operanden hängt. SDF versucht zunächst, die vom Benutzer eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen. Ist über den struktureinleitenden Wert keine eindeutige

Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Benutzer den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

MESSAGE =
bestimmt, ob SDF bei der Aufforderung zur Anweisungseingabe eine Meldung ausgeben soll. Diese wird im geführten Dialog in das Anweisungsmenü integriert.

***NO**
SDF soll keine Meldung ausgeben.

<var: pointer>
Adresse des auszugebenden Meldungstextes, auf Halbwortgrenze ausgerichtet. Der Text wird als Satz variabler Länge erwartet:

2 Byte: absolute Länge des Satzes (n+4)
2 Byte: (reserviert)
n Byte: Meldungstext

Der Meldungstext darf maximal 400 Zeichen lang sein. In SDF-formatierten Bildschirmen werden jedoch nur die ersten 280 Zeichen dargestellt. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden.

PROT =
bestimmt, ob SDF zu protokollierende Eingaben und Meldungen nach SYSOUT schreibt. Falls nicht nach SYSOUT geschrieben wird, sollte der Benutzer des Programms in der Programmdokumentation darüber informiert werden. Ein Protokollpuffer kann bereitgestellt werden.

***YES**
SDF soll zu protokollierende Eingaben und Meldungen nach SYSOUT schreiben.

***NO**
SDF soll keine Protokollierung durchführen. Folgendes Verhalten kann erwartet werden:

Ergebnis der Analyse	**PROT-Parameter**	
	***YES**	***NO**
Kein Fehler:	Eingabeanweisung	– / –
Syntaxfehler:	1. Eingabeanweisung 2. Syntaxfehlermeldung 3. Spin-off-Meldung	Spin-off-Meldung

BUFFER =
Das Protokoll der Anweisung und die Fehlermeldungen können in einen vom Benutzer bereitgestellten Bereich geschrieben werden.

***NO**
Es wird kein Puffer bereitgestellt.

<var: pointer>
Adresse eines Bereichs, in dem das Protokoll der angegebenen Anweisung und die Meldungen, unabhängig vom Wert, der bei PROT angegeben wurde, abgelegt wird. Der Bereich muss auf Halbwortgrenze ausgerichtet sein und wird wie folgt belegt:

2 Byte: maximale Länge des Protokollbereiches
2 Byte: tatsächlich genutzte Länge des Protokollbereiches
n Byte: Protokollsätze

Jeder einzelne Protokollsatz hat folgendes Format:

2 Byte: absolute Länge des Protokollsatzes (m+4)
2 Byte: (reserviert)
m Byte: Inhalt des Protokollsatzes

Wenn der Puffer nicht leer ist, ist der erste Datensatz normalerweise das Protokoll des Eingabe-Kommandos. Die weiteren Datensätze enthalten Meldungen. Wenn kein Eingabeprotokoll zur Verfügung steht oder ausgegeben werden kann, wird ein doppelter Schrägstrich („//“) in den Ausgabebereich geschrieben.

INVAR =
Legt fest, ob die INVARIANT-INPUT-Form der Anweisung abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task zu dieser Zeit erlaubt sind. Im Gegensatz zur Protokollierungsform LOGGING=*INVARIANT-FORM (Anweisung MODIFY-SDF-OPTIONS) werden Kennwörter und geheime Operanden jedoch **nicht** ausgeblendet. Mehr dazu finden Sie auf Seite 421.

***NO**
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

<var: pointer>
Gibt die Adresse eines Puffers an, in den SDF die INVARIANT-INPUT-FORM der Anweisung schreibt. Der Puffer muss auf Wortgrenze ausgerichtet sein und das erste Halbwort muss die Länge des Puffers enthalten. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab. Der Puffer hat dann folgenden Inhalt:

2 Byte: maximale Länge des Puffers
2 Byte: Ausgabelänge, die SDF schreibt (n+4)
2 Byte: (reserviert)
n Byte: INVARIANT-INPUT-Form der Anweisung, ab 7. Byte

SPIN =
bestimmt, welche Anweisung SDF im Stapelbetrieb als nächste liest und analysiert.

***NO**
SDF soll die nächste Anweisung der Anweisungsfolge lesen und bearbeiten.

***YES**
SDF soll alle Anweisungen bis zur nächsten STEP-Anweisung bzw. bis zur END-Anweisung überlesen und ggf. die Bearbeitung mit der Anweisung fortsetzen, die der STEP-Anweisung folgt.

<var: bit:1>
Bitvariable: bit = 0: wie *NO
bit = 1: wie *YES

ERRSTMT =
Bestimmt, welche Anweisung den Spin-off-Mechanismus abbricht, wenn SDF einen Syntaxfehler für die Leseanweisung feststellt.

***STEP**
SDF leitet Spin-off ein bis STEP (oder END) erkannt wird.
Der Returncode ist X'1C', X'34', ...

***NEXT**
SDF leitet keinen Spin-off ein: Die nächste Anweisung wird beim nächsten CMDRST-Aufruf gelesen. Returncode ist dann X'50'.

<var: bit:1>
Bitvariable: bit = 0: wie *STEP
bit = 1: wie *NEXT

CALLID =
Diese Funktion bezieht sich auf den Gebrauch der Makros OPNCALL und CLSCALL. CALLID benennt den Programmkontext (=Syntaxdateihierarchie, durch einen OPNCALL-Makro eröffnet), in dem die Anweisung gelesen und analysiert werden muss. Der Name der Syntaxdateihierarchie (CALLID) muss den 4 Byte langen Wert haben, der von SDF an das Feld zurückgegeben wird, das durch den Operanden CALLID im OPNCALL-Makro bezeichnet wurde.

***NO**
Die aktuelle Syntaxdateihierarchie (Kontext) der Task wird für die Analyse der Anweisung verwendet. Das kann z.B. die für die Task beim LOGON-Vorgang eröffnetet Syntaxdateihierarchie sein.

<var: pointer>
Adresse des Aufrufprüfungsfeldes. Der Bereich muss auf Wortgrenze ausgerichtet sein.

CCSNAME =
Gibt den Namen des Zeichensatzes an, der für den Korrekturdialog auf 8-bit-Terminals und für die Konvertierung von Klein- in Großbuchstaben verwendet wird. Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Jeder codierte Zeichensatz wird durch seinen Namen (Coded Character Set Name, CCSN) bestimmt (siehe Handbuch „XHCS“ [11]). Die Ausgabe von Meldungen wird durch diesen Parameter nicht beeinflusst.

***NO**
Der 7-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

***EXTEND**
Der 8-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

<c-string 1..8> / <var: char:8>
Gibt den Namen eines speziellen 8-bit-Code an, der für Ein-/Ausgabeoperationen verwendet wird. Der Name muss 8 Byte lang sein und kann als C-String-Konstante oder als String-Variable übergeben werden.

DATA_RECORD =
bestimmt, ob SDF Daten speichert, die an Stelle von Anweisungen eingegeben wurden. Wenn dieser Fall eintritt, gibt SDF einen speziellen Returncode zurück (siehe „Rückinformation und Fehleranzeigen“ auf Seite 421). Der Parameter darf nur zur Migration von RDATA zu CMDRST verwendet werden (nicht für Programme, die schon RDSTMT- oder CMDRST-Aufrufe enthalten). Damit kann ein Programm, das im Dialog nur die SDF-Schnittstelle anbietet, kompatibel in Prozeduren und Stapelaufträgen aufgerufen werden. Den SDF-Anweisungen des Programms muss in den Prozeduren und Stapelaufträgen „//“ vorangestellt werden.

***NO**
Keine Datenspeicherung. Das Resultat des CMDRST-Aufrufes ist abhängig von der Eingabe:

Eingabe von:	Art der Eingabe	Ausgabe von CMDRST
Terminal	//<stmt>	im OUTPUT-Parameter
Terminal	<data>	nicht möglich
Prozedur/Batch	//<stmt>	im OUTPUT-Parameter
Prozedur/Batch	<data>	im OUTPUT-Parameter [1)]
S-Prozedur	//<stmt>	im OUTPUT-Parameter
S-Prozedur	<data>	Fehler [2)]

[1)] Daten werden wie Anweisung behandelt

[2)] in S-Prozeduren ist „//“ vor Anweisungen Pflicht

<var: pointer>
Adresse eines auf Wortgrenze ausgerichteten Bereiches, in dem SDF die Daten ablegt, die an Stelle von Anweisungen gelesen wurden.
Der Bereich hat folgendes Layout:

2 Byte: maximale Länge des Bereiches
2 Byte: tatsächlich genutzte Ausgabelänge (n+4)
2 Byte: (reserviert)
n Byte: Datensätze

Das Resultat des CMDRST-Aufrufes ist abhängig von der Eingabe:

Eingabe von:	Art der Eingabe	Ausgabe von CMDRST
Terminal	//<stmt>	im OUTPUT-Parameter
Terminal	<data>	nicht möglich
Prozedur/Batch	//<stmt>	im OUTPUT-Parameter
Prozedur/Batch	<data>	im DATA_RECORD-Parameter
S-Prozedur	//<stmt>	im OUTPUT-Parameter
S-Prozedur	<data>	im DATA_RECORD-Parameter

STMTRC =
Legt fest, ob die Anweisung einen Returncode liefert. Dieser Returncode kann mit SDF-P-Funktionen in S-Prozeduren wie ein Kommando-Returncode ausgewertet werden.

***NO**
Die Anweisung liefert keinen Returncode. SDF übergibt einige Standard-Returncodes entsprechend der bei ERRSTMT und SPIN getroffenen Festlegungen:

(SC2)	SC1	Maincode	Bedeutung
0	0	CMD0001	kein Fehler
0	1	CMD0230	SDF ist im Spin-off
1	0	CMD0232	Bei Spin-off wurde //STEP erkannt, Steuerung ging wieder an das Programm

<var: pointer>
Adresse einer Struktur, die den Returncode der Anweisung enthält. Damit der Benutzer sinnvolle Returncodes erhält, prüft SDF die folgenden Returncode-Bedingungen. Wenn diese nicht eingehalten werden, liefert SDF eigene Returncodes.

1. Bei einem Fehler in der gelesenen Anweisung sollte CMDRST als Nächstes mit SPIN=*YES und einem Anweisungs-Returncode mit SC1 ungleich null aufgerufen werden.
2. Bei fehlerfreier Abarbeitung sollte CMDRST als Nächstes mit SPIN=*NO und einem Anweisungs-Returncode mit SC1 gleich null aufgerufen werden.
3. Bei semantischen Fehlern sollte CMDRST als Nächstes mit SPIN=*YES und SC1 ungleich null aufgerufen werden. Wenn kein Returncode übergeben wird oder der SC1 gleich null ist, setzt SDF den Returncode CMD0230 ein.
4. Wenn CMDRST mit ERRSTMT=*NEXT aufgerufen wird und der Makro-Returncode X'50' zurückkommt, kann der nächste CMDRST ohne Spin-off und mit SC1 gleich null aufgerufen werden. Das Programm sollte keinen Spin-off starten, da es selbst den Spin-off von SDF unterdrückt hat.

OUTFORM =
Legt fest, welches Format des normierten Übergabebereiches SDF ausgibt. Bei der Angabe von Default-Werten (siehe Parameter DEFAULT) identifiziert SDF das Format des Übergabebereiches selbst.

***NEW**
Der normierte Übergabebereich hat das neue Format (siehe Seite 371ff).

***OLD**
Der Übergabebereich hat das bis SDF V4.0 verwendete Format (siehe Kapitel „Anhang“ auf Seite 605).

<var: bit:1>
Bitvariable: bit = 0: wie *NEW
bit = 1: wie *OLD

Beschreibung der Parameter MF, PARAM, MACID und PREFIX siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8].

Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 371ff beschrieben. Das bis SDF V4.0 verwendete Format des Übergabebereiches finden Sie im Kapitel „Anhang“ auf Seite 605.

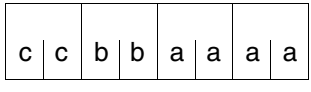
In der INVARIANT-INPUT-Form wird die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt, die für die Task zu dieser Zeit erlaubt sind. Die INVARIANT-INPUT-Form ist damit die größtmögliche Eingabeform für eine Anweisung, die für einen Benutzer mit bestimmten Privilegien und im gewählten Dialogmodus zulässig ist. Kennwörter und geheime Operanden werden nicht ausgeblendet. Die INVARIANT-INPUT-Form ist nicht das Gleiche wie die Protokollierungsform LOGGING=*INVARIANT-FORM (siehe MODIFY-SDF-OPTIONS).

Die INVARIANT-INPUT-Form liefert ein portables Format für SDF-Eingaben. Da nur Standardnamen verwendet werden, kann die INVARIANT-INPUT-Form ohne Probleme zu einem fernen Partner übertragen werden, auch wenn dort andere externe Namen oder andere Default-Werte verwendet werden.

Die INVARIANT-INPUT-Form kann vom fernen Partner abgewiesen werden, wenn Kommandos, Operanden usw. dort entfernt wurden (Anweisung REMOVE) oder auf Grund der Privilegien nicht erlaubt sind.
Default-Werte, die in der lokalen Umgebung nicht erlaubt sind, werden auch nicht zum fernen Partner übertragen. Bei der Ausführung der Anweisung im fernen System werden die Default-Werte eingesetzt, die in dieser Umgebung gelten.

Der Returncode wird im Standardheader der Parameterliste übergeben.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	erfolgreiche Beendigung
00	20	0004	nicht behebbarer Systemfehler
	01	0008	Parameterfehler:
00			– falsche Parameterliste
01			– OUTPUT
02			– STMT-Liste
03			– DEFAULT
04			– MESSAGE
05			– PROT
06			– INVAR
07			– CALLID
08			– DATA_RECORD
10			– STMTRC
00	40	000C	Übergabebereich zu klein

Fortsetzung ➠

(SC2)	SC1	Maincode	Bedeutung
00	40	0010	Eingabeende erkannt
00	40	0014	Anweisung fehlerhaft, Kommando wurde erkannt
00	40	0018	Anweisung ist zwar in Ordnung, die vom Programm übergebenen Default-Werte sind jedoch fehlerhaft
00	40	001C	Anweisung fehlerhaft, //STEP wurde erkannt
00	40	0028	Puffer zu klein, Protokoll abgeschnitten
00	40	002C	END-Anweisung wurde gelesen
00	40	0034	Anweisung fehlerhaft, die nächste zu bearbeitende Anweisung ist END
00	40	003C	Programm nicht in Syntaxdatei bekannt
00	40	0040	angegebene CALLID nicht gefunden
00	40	0044	im DSSM-Katalog eingetragene Syntaxdatei nicht gefunden
00	40	0050	Anweisung fehlerhaft, Spin-off wurde nicht eingeleitet
00	40	005C	INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten
00	20	0064	XHCS-Fehler bei Anweisungseingabe
02	00	0068	Datensatz wurde an Stelle einer Anweisung gelesen und im DATA_RECORD-Puffer gespeichert

CMDRST liefert auch die Herkunft der Anweisung. Bei fehlerfreier Parameterliste wird die SYSSTMT-Zuweisung im Feld <prefix><macid>SYSSTMT_STATE der Parameterliste abgelegt.

Folgende Werte können auftreten:

Wert	SYSSTMT =
X'01'	Datensichtstation
X'02'	SYSCMD in Nicht-S-Prozedur oder Datei
X'03'	Kartenleser
X'04'	Diskette
X'05'	SYSCMD in S-Prozedur
X'06'	S-Variable

Migration von RDSTMT zu CMDRST

Die Migration von RDSTMT nach CMDRST ist nur dann notwendig, wenn Sie die neue Funktionalität von CMDRST nutzen wollen. Folgende Punkte sollten Sie dabei beachten:

- Beim CMDRST-Aufruf muss immer der MF-Parameter angegeben werden. Die Parameterliste von CMDRST muss immer, getrennt vom ausführenden Code, deklariert und mit MF=L initialisiert werden. Die DSECT für die Parameterliste muss mit MF=D generiert werden. Direkte Modifkationen (über ein Offset vom Beginn der Parameterliste) sind nicht erlaubt. Mehr zu den Parametern MF und PARAM finden Sie im Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8].

Gegenüberstellung von RDSTMT und CMDRST

RDSTMT	**CMDRST**
<Programmcode> . . .	<Programmcode> . . .
	LA 1,LABEL USING D,1
<Code zur Manipulieren der Parameterliste mittels LABEL+Offset>	<Code zum Manipulieren der Parameter- liste mit den Namen aus DSECT D>
RDSTMT MF=(E,(LABEL))	CMDRST MF=M,PARAM=(1),<params> CMDRST MF=E,PARAM=LABEL
<Programmcode> . . .	<Programmcode> . . .
<Daten>	<Daten>
LABEL RDSTMT MF=L,<params>	LABEL CMDRST MF=L,<params> <dsect> D CMDRST MF=D

- Der Makro-Returncode wird im Standardheader der Parameterliste übergeben. Der Maincode von CMDRST entspricht den Werten, die bei RDSTMT im rechtsbündigen Byte im Register 15 übergeben wurden. Die Erzeugung von EQUATES für Returncodes mit dem Makro CMDANALY ist nicht mehr notwendig, da diese mit CMDRST MF=D automatisch erzeugt werden. Im Folgenden finden Sie eine Gegenüberstellung der alten (RDSTMT-) und neuen (CMDRST-)Feldnamen:

RDSTMT	**CMDRST**
&P.NOERR	&PREFIX.MDRSUCCESSFUL
&P.SYERR	&PREFIX.MDRSYSTEM_ERROR
&P.PAERR	&PREFIX.MDRPARAMETER_ERROR
&P.TRUNC	&PREFIX.MDRAREA_TOO_SMALL
&P.EOF	&PREFIX.MDREOF
&P.SCMD	&PREFIX.MDRSTMTERROR_CMD
&P.DFLT	&PREFIX.MDRWRONG_DEFAULTS
&P.STEP	&PREFIX.MDRSTMTERROR_STEP
&P.PTRC	&PREFIX.MDRPROTOCOL_TRUNCATION
&P.END	&PREFIX.MDREND_STMT
&P.EERR	&PREFIX.MDRSTMTERROR_END
&P.NOPRG	&PREFIX.MDRPROGRAM_NOT_IN_SYNTAX_FILE
&P.INCID	&PREFIX.MDRINVALID_CALLID
&P.NOFND	&PREFIX.MDRSYNTAX_FILE_NOT_FOUND
&P.NEXT	&PREFIX.MDRSTMTERROR_NEXT
&P.ITRC	&PREFIX.MDRINVARIANT_INPUT_TRUNCATED
&P.XHCS	&PREFIX.MDRXHCS_ERROR

- Die Syntax der Parameter von CMDRST ist nur in folgenden Punkten anders als bei RDSTMT:
  - Schlüsselwörter sind mit führendem Stern einzugeben (z.B. *YES)
  - Zeichenketten müssen in Hochkommas eingeschlossen werden (z.B. 'test')
  - Der Parameterwert *ADDR ist nicht mehr erlaubt ; es können nur Identifikatoren angegeben werden.

Beispiele für unterschiedliche Angaben

RDSTMT	**CMDRST**
PROGRAM = name	PROGRAM = 'name'
OUTPUT= addr/(r)	OUTPUT = identifier [2]
STMT = *ALL/(name,...)/ [1] *ADDR(addr/(r))	STMT = *ALL/identifier
PREFER = *NO/name/ *ADDR(addr/(r))	PREFER = *NO/'name'/identifier
DEFAULT = *NO/(addr,...)	DEFAULT = *NO/identifier
MESSAGE = *NO/addr/(r)	MESSAGE = *NO/identifier
PROT = YES/NO	PROT = *YES/*NO
BUFFER = *NO/addr|(r)	BUFFER = *NO/identifier
INVAR = *NO/addr|(r)	INVAR = *NO/identifier
SPIN = NO/YES	SPIN = *YES/*NO
ERRSTMT = STEP/NEXT	ERRSTMT = *STEP/*NEXT
CALLID = *NO/addr/(r)	CALLID = *NO/id
CCSNAME = *NO/*EXTEND/name	CCSNAME = *NO/*EXTEND/'name'/ identifier
MF = L	MF = L
MF = (E,(1))/(E,param)	MF = E,PARAM=identifier
MF nicht angegeben (Standard-Form)	(nicht unterstützt)

[1] Die Listenangabe bei STMT und DEFAULT ist nicht mehr möglich

[2] identifier kann ein Label im Programm sein (Adresse), oder bei MF=M ein Register

- Gleichzeitig mit der Einführung von CMDRST wurde das Format des normierten Übergabebereiches geändert. Im neuen Übergabebereich können wesentlich mehr Informationen abgelegt werden und die Ausrichtung der Daten ist gewährleistet. Deshalb sollten möglichst nur noch Übergabebereiche im neuen Format bei OUTPUT und auch bei DEFAULT angegeben werden. Aus Kompatibilitätsgründen unterstützt SDF natürlich weiterhin das alte Format des Übergabebereiches (siehe Parameter OUTFORM).

Wenn das Programm eigene Default-Werte an SDF übergibt, identifiziert SDF das Format des Übergabebereiches selbst:

– Bei OUTFORM=*OLD muss der Programmteil zur Analyse des Übergabebereiches nicht geändert werden. Der Übergabebereich muss jedoch weiterhin mit CMDSTRUC generiert werden (siehe Kapitel „Anhang“ auf Seite 605).

– Bei OUTFORM=*NEW ist es in manchen Fällen ausreichend, CMDSTRUC durch CMDTA zu ersetzen und das Programm erneut zu übersetzen. Die Felder in den DSECTs, die von CMDSTRUC generiert werden, sind zum Teil auch in CMDTA-generierten DSECTs enthalten. Sie finden eine Gegenüberstellung der alten und neuen DSECT beim Makro CMDTA („Migration von CMDSTRUC zu CMDTA“ auf Seite 436).

CMDSEL
Auswahlmaske für den geführten Dialog erzeugen

Der Makro CMDSEL erzeugt eine Auswahlmaske in der Art der SDF-Fragebögen, in der Benutzer im geführten Dialog einen oder mehrere Begriffe (Items) auswählen können.

In Bild 13 sehen Sie das prinzipielle Layout einer mit CMDSEL erzeugten Auswahlmaske.

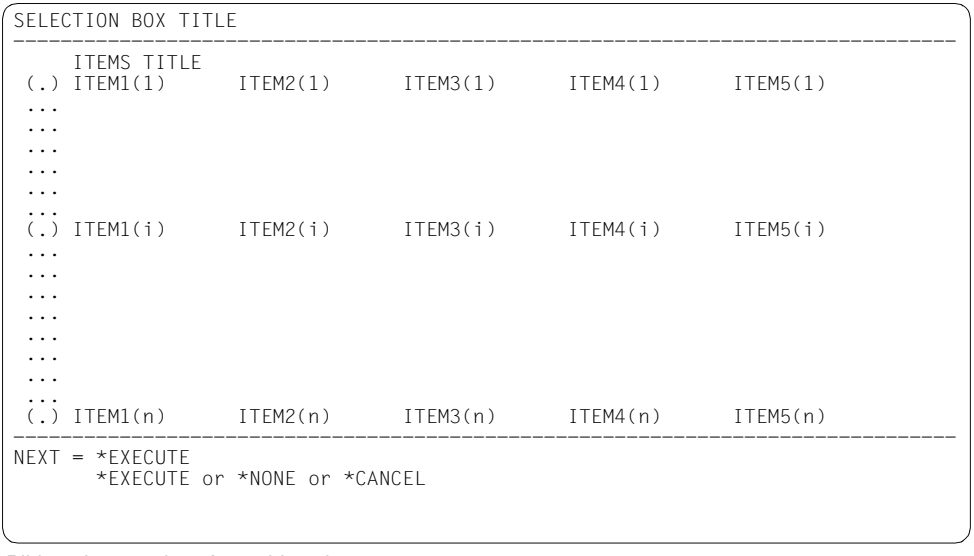


Bild 13: Layout einer Auswahlmaske

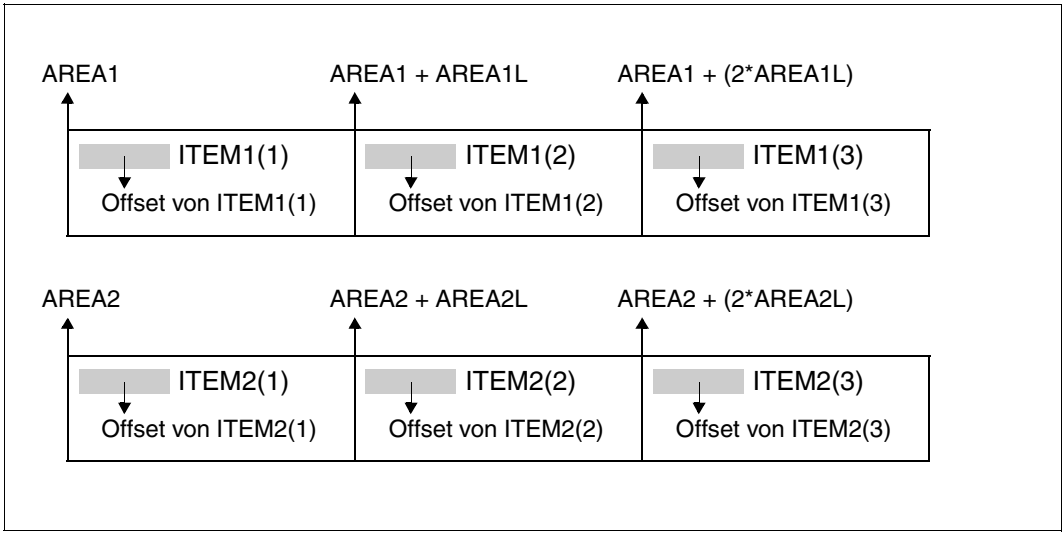
Auf BS2000-Bildschirmen können maximal 24 Zeilen ausgegeben werden. Wieviele Zeilen davon für die Auswahlmaske verwendbar sind, hängt davon ab, ob eine Überschrift ausgegeben wird und wie viele NEXT-Zeilen es gibt. Bei mehr als 20 Zeilen in der Auswahlmaske erfolgt die Anzeige auf mehreren Bildschirmseiten. Die NEXT-Zeile enthält dann zusätzlich + (vorwärts blättern) und – (rückwärts blättern).

Eine Zeile am Bildschirm kann maximal 5 Spalten enthalten. Der Inhalt einer Spalte in einer Zeile muss in einem Datenbereich zur Verfügung gestellt werden, dessen Anfangsadresse und Länge im Programm übergeben wird. Der Platzbedarf aller Zeilen in einer Spalte errechnet sich demzufolge aus der Anzahl der Zeilen multipliziert mit der Länge des Datenbereichs einer Spalte.

Im Datenbereich einer Spalte steht jeweils ein auszugebender Begriff mit einem bestimmten Offset relativ zum Beginn des Datenbereiches und mit einer festgelegten Länge. Standardmäßig hat der Begriff kein Offset zum Beginn des Datenbereiches.

Beispiel

Bei einer Auswahlbox mit 2 Spalten und 3 Zeilen ergibt sich die folgende Datenstruktur:



Operation	**Operanden**
CMDSEL	SELECT = *SINGLE / *MULTIPLE / <var: enum-of SELECTION_S:1>
	,LINENBR = <integer 1..65536> / <var: int:4>
	,OUTPUT = <var: pointer>
	,TITLE@ = NULL / <var: pointer>
	,TITLEL = <integer 1..75> / <var: int:4>
	,TITEMS = NULL / <var: pointer>
	,TITEMSL = 0 / <integer 1..65535> / <var: int:4>
	,AREA1 = NULL / <var: pointer>
	,AREA1L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF1L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM1L = *AREA_LENGTH / <integer 1..75> / <var: int:1>
	,AREA2 = NULL / <var: pointer>
	,AREA2L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF2L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM2L = *AREA_LENGTH / <integer 1..75> / <var: int:1>

Fortsetzung ➠

Operation	Operanden
CMDSEL (Forts.)	,AREA3 = NULL / <var: pointer>
	,AREA3L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF3L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM3L = *AREA_LENGTH / <integer 1..75> / <var: int:1>
	,AREA4 = NULL / <var: pointer>
	,AREA4L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF4L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM4L = *AREA_LENGTH / <integer 1..75> / <var: int:1>
	,AREA5 = NULL / <var: pointer>
	,AREA5L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF5L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM5L = *AREA_LENGTH / <integer 1..75> / <var: int:1>

SELECT =
legt fest, ob in der Maske Einfachauswahl oder Mehrfachauswahl erlaubt ist.

***SINGLE**
Nur ein Begriff ist in der Maske auswählbar.

***MULTIPLE**
Mehrere Begriffe sind in der Maske auswählbar.

<var: enum-of SELECTION_S:1>
Variable, die folgende Werte annehmen kann:
0: wie *SINGLE
1: wie *MULTIPLE

LINENBR = <integer 1..65536> / <var: int:4>
Anzahl der Zeilen, die in der Auswahlmaske angezeigt werden.

OUTPUT = <var: pointer>
Adresse eines Bereiches, indem das Ergebnis der Auswahl eingetragen wird. Für jeden der auswählbaren Begriffe muss ein Byte reserviert werden, d.h. der Bereich muss mindestens die bei LINENBR angegebene Länge (in Byte) haben und mit null oder Leerzeichen initialisiert sein. Wenn der Benutzer den Begriff(i) auswählt, dann erhält Byte(i) einen Wert verschieden von null/Leerzeichen. Wird ein Byte schon vor der Benutzerauswahl mit einem Wert ungleich null oder Leerzeichen belegt, dann ist der zugehörige Begriff der Default-Wert in der Auswahlmaske.

TITLE@ =
Überschrift der Auswahlbox, die in der ersten Zeile auf dem Bildschirm ausgegeben wird.

NULL
Keine Überschrift der Auswahlbox.

<var: pointer>
Adresse der Zeichenkette mit der Überschrift.

TITLEL = <integer 1..75> / <var: int:4>
Länge der Zeichenkette, die als Überschrift ausgegeben wird. Der Parameter wird nur ausgewertet, wenn bei TITLE@ eine Adresse≠0 angegeben wurde.

TITEMS = NULL / <var: pointer>
Adresse einer Begriffs-Überschrift, die in der 3. Bildschirmzeile ab der 6. Postion angezeigt wird (3270-Terminals: 8. Position).

TITEMSL = 0 / <integer 1..65535> / <var: int:4>
Länge der Begriffs-Überschrift (maximal 73 Zeichen).

AREA*x* =
mit *x*=(1..5):
Datenbereich, in dem der Inhalt einer Zeile in einer Spalte abgelegt ist. D.h. es muss für jede Zeile in jeder Spalte genau einen solchen Datenbereich geben.

NULL
In einer bestimmten Zeile in Spalte *x* wird kein Begriff ausgegeben.

<var: pointer>
Adresse des Datenbereiches, in dem der auszugebende Begriff abgelegt ist.

AREA*x*L = 0 / <integer 1..65535> / <var: int:2>
mit *x*=(1..5):
Länge eines Datenbereiches (in Byte) in Spalte *x*.
Die Gesamtlänge aller Datenbereiche der Spalte *x* wird berechnet aus der Anzahl der Zeilen multipliziert mit dem Wert von AREA*x*L.

ITOFF*x*L =
mit *x*=(1..5):
Offset des Begriffs relativ zum Beginn des bei AREA*x* angegebenen Datenbereiches.

***AREA_START**
Kein Offset

<integer 1..65535> / <var: int:2>
Offset in Byte

ITEM*x*L =
mit *x*=(1..5): Länge des Begriffs (maximal 75 Zeichen)

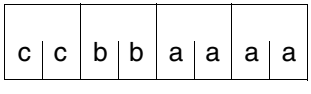
***AREA_LENGTH**
Der Begriff füllt den gesamten Datenbereich AREA*x* aus.

<integer 1..75> / <var: int:1>
Länge des Begriffs (in Byte)

Rückinformation und Fehleranzeigen

Der Returncode wird im Standardheader der Parameterliste übergeben.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	erfolgreiche Beendigung
00	20	0004	nicht behebbarer Systemfehler
	01	0008	Parameterfehler:
00			– falsche Parameterliste
01			– LINENBR
02			– SELECT
03			– OUTPUT
04			– TITLE@
05			– TITLEL
06			– Item1
07			– Item2
08			– Item3
09			– Item4
0A			– Item5
00	40	000C	Benutzer hat nichts ausgewählt

CMDSTA
Informationen über aktivierte Syntaxdateien ausgeben

Der Makro CMDSTA bewirkt, dass das Programm informiert wird über

– die aktivierten Syntaxdateien

– die Festlegungen, die für die Kommando-/Anweisungseingabe und -verarbeitung gelten.

Operation	Operanden
CMDSTA	OUTAREA = addr / (r) ,CALLID = *NO / addr / (r) ,FORM = SHORT / LONG / USER [,MF = { L / (E,(1)) / (E,opadr) }]

OUTAREA = addr / (r)
Adresse eines Übergabebereichs bzw. Register, das diese Adresse enthält. Die Information wird in diesen Bereich geschrieben. Der Bereich kann mit dem Makro CMDMEM generiert werden. Dieses Feld muss auf Halbwortgrenze ausgerichtet sein.

Der Übergabebereich wurde ab SDF V2.0A vergrößert. Näheres siehe Seite 407.

CALLID =
bezieht sich auf einen Kontext (=Syntaxdateihierarchie), der durch einen OPNCALL-Makro eröffnet wurde. Der Name der Syntaxdateihierarchie (callid) muss den 4 Byte langen Wert haben, der von SDF an das Feld zurückgegeben wird, welches durch den Operanden CALLID im Open-Context-Makro bezeichnet wurde. Diese Funktion bezieht sich auf den Gebrauch der Makros OPNCALL und CLSCALL und ist für die BS2000-Entwicklung reserviert.

***NO**
Die aktuelle aktivierte Syntaxdateihierarchie wird verwendet.

addr / (r)
Adresse eines 4 Byte langen Feldes oder Register, das diese Adresse enthält. Der Aufrufende übermittelt die callid dem Kontext (=Syntaxdateihierarchie), den er verwenden will. Dieses Feld muss auf Wortgrenze ausgerichtet sein.

FORM =
gibt an, ob die Ausgabeinformationen in Kurzform, in Langform oder zusätzlich mit allen Benutzersyntaxdateien in den Übergabebereich OUTAREA geschrieben werden.

SHORT
Ausgegeben werden nur Informationen über die aktivierte Basis-Systemsyntaxdatei (nicht die aktivierten Subsystem-Syntaxdateien), über die aktivierte Gruppensyntaxdatei, über die zuletzt aktivierte Benutzersyntaxdatei sowie die aktuellen Optionen für die Eingabe und die Verarbeitung von Kommandos und Anweisungen. In der Kurzform werden 530 Byte ausgegeben.

LONG
wie SHORT; zusätzlich werden alle Informationen über Subsystem-Syntaxdateien ausgegeben. In diesem Fall müssen die ersten zwei Byte des Übergabebereiches seine Länge enthalten. Der Übergabebereich muss für die Langform größer als 530 Byte sein. Die Anzahl der ausgegebenen Subsystem-Informationen hängt davon ab, wieviel Platz im Übergabebereich zur Verfügung steht. Passten nicht alle Subsystem-Informationen in den Übergabebereich, so wird ein entsprechender Return-Code übergeben.

USER
wie LONG; zusätzlich werden Informationen zu allen aktivierten Benutzersyntaxdateien ausgegeben.

MF =
definiert besondere Anforderungen an die Makroauflösung (Einzelheiten siehe Seite 385ff und Handbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Rückinformation und Fehleranzeigen

Register 15 enthält im rechtsbündigen Byte einen Returncode. EQUATE-Anweisungen dafür können mit dem Makro CMDANALY generiert werden.

X’00’ normale Beendigung
X’04’ nicht behebbarer Fehler
X’08’ Operandenfehler im Makroaufruf
X’0C’ Übergabebereich zu klein
X’38’ SDF ist nicht verfügbar
X’4C’ Das Programm ist oberhalb der 16 MByte-Grenze nicht ablauffähig, weil SDF nicht geladen ist. Bitte Systembetreuung verständigen.

Der Übergabebereich hat bei FORM=SHORT folgenden Aufbau:

Byte	Inhalt	
0	Bit 0 = 0	UTILITY-INTERFACE = NEW-MODE
	Bit 0 = 1	UTILITY-INTERFACE = OLD-MODE
	Bit 1 = 0	PROCEDURE-DIALOGUE = NO
	Bit 1 = 1	PROCEDURE-DIALOGUE = YES
	Bit 2 = 0	CONTINUATION = NEW-MODE
	Bit 2 = 1	CONTINUATION = OLD-MODE
	Bit 3 = 0	CMD-STATISTICS = NO
	Bit 3 = 1	CMD-STATISTICS = YES
	Bit 4 = 0	MENU-LOGGING = NO
	Bit 4 = 1	MENU-LOGGING = YES
	Bit 5 = 0	MODE = EXECUTION
	Bit 5 = 1	MODE = TEST
	Bit 6 = 0	kein Spin-off für CMDRST (read statement)
	Bit 6 = 1	Spin-off für CMDRST
	Bit 7 = 0	INPUT-HISTORY = OFF
	Bit 7 = 1	INPUT-HISTORY = ON
1	Umfang der Benutzerführung (siehe SET-GLOBALS ..., GUIDANCE=... und MODIFY-SDF-OPTIONS ...,GUIDANCE=...)	
	X'04'	NO
	X'05'	EXPERT
	X'06'	MAXIMUM
	X'07'	MINIMUM
	X'08'	MEDIUM
2	Art der Eingabeprotokollierung (siehe SET-GLOBALS ..., LOGGING=... und MODIFY-SDF-OPTIONS ...,LOGGING=...)	
	X'04'	INPUT-FORM
	X'05'	ACCEPTED-FORM
	X'06'	INVARIANT-FORM
3 bis 56	Name der aktivierten Basis-Systemsyntaxdatei	

Fortsetzung ➠

Byte	Inhalt
57 bis 68	Versionsnummer der aktivierten Basis-Systemsyntaxdatei (siehe SET-GLOBALS VERSION= ...)
69 bis 122	Name der für die Task aktivierten Gruppensyntaxdatei
123 bis 134	Versionsnummer der für die Task aktivierten Gruppensyntaxdatei (siehe SET-GLOBALS VERSION= ...)
135 bis 188	Name der zuletzt aktivierten Benutzersyntaxdatei
189 bis 200	Versionsnummer der zuletzt aktivierten Benutzersyntaxdatei (siehe SET-GLOBALS VERSION= ...)
201 bis 230	Name des Programms, das der Kommandoprozessor im Testmodus zur Analyse der eingegebenen Anweisungen verwendet
231 bis 234	aktuelle SDF-Version in der Form nn.n
235	Funktionstastenbelegung (siehe SET-GLOBALS...,FUNCTION-KEYS= und MODIFY-SDF-OPTIONS..., FUNCTION-KEYS=....)
	X'04' OLD-MODE X'05' STYLE-GUIDE-MODE
236 bis 237	Anzahl zu speichernder Eingaben (siehe SET-GLOBALS..., NUMBER-OF-INPUTS=... und MODIFY-SDF-OPTIONS... INPUT-HISTORY=*ON (NUMBER-OF-INPUTS=...))
238	sicherheitsrelevante Einstellungen (siehe MODIFY-SDF-OPTIONS ..., MODE=*TEST(...),...,INPUT-HISTORY=*ON(...))
	bit 0 = 0 CHECK-PRIVILEGES = *NO bit 0 = 1 CHECK-PRIVILEGES = *YES bit 1 = 0 PASSWORD-PROTECTION = *NO bit 1 = 1 PASSWORD-PROTECTION = *YES
239 bis 243	reserviert
244 bis 245	Anzahl der aktivierten Benutzersyntaxdateien
246 bis 531	reserviert für künftige Erweiterungen

Das Bit 6 in Byte 0 (Spin-off für CMDRST) kann von einem Benutzerprogramm nicht verwendet werden, da CMDRST nach Spin-off-Ende immer einen Return macht. Deshalb wird diese Anzeige für Benutzerprogramme immer zurückgesetzt. Dieses Merkmal ist für die Systemprogrammierung reserviert.

Wenn bei der Ausführung von MODIFY-SDF-OPTIONS eine Substitution von Aliasnamen durchgeführt wurde, stehen die Syntaxdateinamen mit <cat-id> und <user-id> im CMDSTA-Übergabebereich.

Für FORM=LONG wird der Übergabebereich wie folgt erweitert:

Byte	Inhalt
239	reserviert
240 bis 243	Adresse der Liste aller für die Task aktivierten Benutzersyntaxdateien
244 bis 245	Anzahl der aktivierten Benutzersyntaxdateien
246 bis 531	reserviert für künftige Erweiterungen
532 bis 533	Anzahl der aktivierten Subsystem-Syntaxdateien
534 bis 587	Name der ersten aktivierten Subsystem-Syntaxdatei
588 bis 599	Versionsnummer der ersten aktivierten Subsystem-Syntaxdatei
600 bis 653	Name der zweiten aktivierten Subsystem-Syntaxdatei
654 bis 665	Versionsnummer der zweiten aktivierten Subsystem-Syntaxdatei
. . .	

Für FORM=USER wird der Übergabebereich wie folgt erweitert:

Byte	Inhalt
(im Anschluss an die Informationen über Subsystem-syntaxdateien)	Name der ersten aktivierten Benutzersyntaxdatei (54 Byte)
	Versionsnummer der ersten aktivierten Benutzersyntaxdatei (2 Byte)
	Name der zweiten aktivierten Benutzersyntaxdatei (54 Byte)
	Versionsnummer der zweiten aktivierten Benutzersyntaxdatei (2 Byte)
. . .	

CMDTA
Übergabebereich für eine analysierte Anweisung generieren

Der Makro CMDTA generiert eine DSECT für den normierten Übergabebereich und deklariert und initialisiert den Datenbereich. Der normierte Übergabebereich dient folgenden Zwecken:

1. SDF übergibt an das Programm eine analysierte Anweisung (siehe CMDCST, CMDRST und CMDTST).
2. Das Programm gibt eine semantisch fehlerhafte Anweisung an SDF zurück (siehe CMDCST).
3. Das Programm übergibt an SDF Werte, die eingegebene Operandenwerte ersetzen sollen (siehe CMDRST und CMDTST).

Der normierte Übergabebereich ist im Detail auf den Seiten 371ff beschrieben.

Operation	Operanden
CMDTA	MAXLEN = <integer 0..2147483647> / <var: int 0..2147483647> ,MF = C / D / L / M

MAXLEN = <integer 0..2147483647> / <var: int:8>
Legt die Länge des normierten Übergabebereiches fest (in Byte).

Beschreibung der Parameter MF, MACID und PREFIX siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8].

Migration von CMDSTRUC zu CMDTA

Im neuen Übergabebereich sind alle Daten korrekt ausgerichtet (Adressen auf Wortgrenze, Halbworte auf Halbwortgrenze usw.). Aus diesem Grund können ICM-Befehle durch L- und LH-Befehle ersetzt werden.
Das Längenfeld des neuen Übergabebereiches belegt 4 Byte (nach dem Standardheader). Im alten Übergabebereich waren es 2 Byte direkt am Anfang des Übergabebereiches.

Nachfolgend finden Sie eine Gegenüberstellung der von CMDSTRUC und CMDTA erzeugten DSECT. Die Namen, die sich mit CMDTA ändern, sind `halbfett` dargestellt.

```
*-      ****************************
*-      * STRUCTURED DESCRIPTION    *
*-      ****************************
*-     CMDSTRUC                    CMDTA

CMDSDES  DSECT                     CMDSDES  DSECT
CMDML    DC    XL2'0'              CMDFHDR  FHDR  MF=(C,CMD),EQUATES=NO
CMDINTN  DC    CL8' '              CMDML          DS    F
CMDVER   DC    XL4'0'              CMDINTN        DS    CL8
CMDLAB   DC    AL4(0)              CMDLAB         DS    A
CMDNRMO  DC    XL2'0'              CMDVER         DS    CL3
CMDMAINO EQU   *                   CMDUNU1        DS    XL1
CMDSDESL EQU   *-CMDSDES           CMDUNU2        DS    XL8
                                   CMDNRMO        DS    H
                                   CMDUNU3        DS    XL2
                                   CMDMAIN        EQU   *
                                   CMDSDEL        EQU   *-CMDFHDR

*-       ***************************
*-       * OPERAND DESCRIPTOR      *
*-       ***************************

CMDODES  DSECT                     CMDODES        DSECT
CMDGSTAT DC    X'00'               CMDGSTA        DS    AL1
CMDOCC   EQU   X'80'               CMDOCC         EQU   X'80'
CMDUCH   EQU   X'40'               CMDUCH         EQU   X'40'
CMDERR   EQU   X'20'               CMDERR         EQU   X'20'
CMDRDEF  EQU   X'10'               CMDRDEF        EQU   X'10'
*                                  CMDOTYP        DS    FL1
CMDOTYPE DC    X'00'               CMDGATT        DS    AL1
*                                  CMDWILD        EQU   X'80'
CMDODESL EQU   *-CMDODES           CMDCWIL        EQU   X'40'
                                   CMDSATT        DS    0XL1
                                   CMDFNAT        DS    AL1
                                   CMDFNCA        EQU   X'80'
                                   CMDFNUS        EQU   X'40'
                                   CMDFNGE        EQU   X'20'
                                   CMDFNVE        EQU   X'10'
                                   CMDFNTP        EQU   X'08'
                                            ORG   CMDSATT
                                   CMDNAAT        DS    AL1
                                   CMDNAUN        EQU   X'80'
                                            ORG   CMDSATT
                                   CMDCNAT        DS    AL1
                                   CMDCNUN        EQU   X'80'
                                   CMDCNCA        EQU   X'40'
                                            ORG   CMDSATT
                                   CMDTXAT        DS    AL1
```

```
                                        CMDTXSE         EQU   X'80'
                                                ORG   CMDSATT
                                        CMDXTAT         DS    AL1
                                        CMDXTOD         EQU   X'80'
                                                ORG   CMDSATT
                                        CMDPXAT         DS    AL1
                                        CMDPXAB         EQU   X'80'
                                        CMDPXPO         EQU   X'40'
                                        CMDPXQU         EQU   X'20'
                                                ORG   CMDSATT
                                        CMDSTAT         DS    AL1
                                        CMDSTQU         EQU   X'80'
                                                ORG   CMDSATT
                                        CMDPVAT         DS    AL1
                                        CMDPVCO         EQU   X'80'
                                        CMDPVUI         EQU   X'40'
                                                ORG   CMDSATT+1
                                        CMDODEL         EQU   *-CMDODES

*-        ***************************
*-        * OPERAND HEADER          *
*-        ***************************
*-
CMDHEAD  DSECT                   CMDHEAD         DSECT
CMDDES   DC    XL2'0'                           DS    0A
CMDOPTR  DC    AL4(0)            CMDDES         DS    XL4
CMDHEADL EQU   *-CMDHEAD         CMDOPTR        DS    A
                                 CMDHEAL        EQU   *-CMDHEAD

*-        ***************************
*-        * STRUCTURE OPERAND       *
*-        ***************************
*-
CMDSOP   DSECT                   CMDSOP   DSECT
CMDNREL  DC    XL2'0'                     DS    0A
CMDSTRT  DC    XL6'0'            CMDNREL  DS    H
CMDSTEL  EQU   *                 CMDUNU4  DS    XL2
CMDSOPL  EQU   *-CMDSOP          CMDSTRT  DS    XL8
                                 CMDSTEL  EQU   *
                                 CMDSOPL  EQU   *-CMDSOP

*-        ***************************
*-        * OPERAND VALUE           *
*-        ***************************
*-
CMDOVAL  DSECT                   CMDOVAL  DSECT
CMDLVAL  DC    XL2'0'                     DS    0A
CMDAVAL  EQU   *                 CMDLVAL  DS    H
```

```
CMDTIME  EQU   CMDAVAL            CMDUNU5  DS    XL2
CMDHOUR  DC    XL2'0'             CMDAVAL  EQU   *
CMDMINU  DC    X'0'               CMDTIME  EQU   CMDAVAL
CMDSEC   DC    X'0'               CMDHOUR  DS    H
         ORG   CMDAVAL            CMDMINU  DS    X
CMDIVAL  DC    XL4'0'             CMDSEC   DS    X
CMDOVALL EQU   *-CMDOVAL                  ORG   CMDAVAL
                                  CMDIVAL  DS    F
                                  CMDOVLL  EQU   *-CMDOVAL

*-        ***************************
*-        * LIST ELEMENT             *
*-        ***************************
*-
CMDLE    DSECT                    CMDLE    DSECT
CMDETYPE DC    XL6'0'                      DS    0A
CMDORL   EQU   *                  CMDETYP  DS    XL8
CMDNEL   DC    AL4(0)             CMDORL   EQU   *
CMDELOP  EQU   *                  CMDNEL   DS    AL4
CMDELVAL EQU   *                  CMDELOP  EQU   *
CMDLEL   EQU   *-CMDLE            CMDELVA  EQU   *
                                  CMDLEL   EQU   *-CMDLE

*-        *****************************
*-        * EQUATES FOR OPERAND TYPES *
*-        *****************************
*-
CMDC#RES EQU   1                  CMDCRES  EQU   1
CMDINT   EQU   2                  CMDINT   EQU   2
CMDX#STR EQU   4                  CMDXSTR  EQU   4
CMDC#STR EQU   5                  CMDCSTR  EQU   5
CMDNAME  EQU   6                  CMDNAME  EQU   6
CMDA#NAM EQU   7                  CMDANAM  EQU   7
CMDS#NAM EQU   8                  CMDSNAM  EQU   8
CMDLABEL EQU   9                  CMDLABE  EQU   9
CMDSTAR  EQU   10                 CMDSTAR  EQU   10
CMDF#FIL EQU   11                 CMDFFIL  EQU   11
CMDP#FIL EQU   12                 CMDPFIL  EQU   12
CMDTIM   EQU   13                 CMDTIM   EQU   13
CMDDATE  EQU   14                 CMDDATE  EQU   14
CMDCNAME EQU   15                 CMDCNAM  EQU   15
CMDTEXT  EQU   16                 CMDTEXT  EQU   16
CMDCATID EQU   17                 CMDCATI  EQU   17
CMDI#TXT EQU   18                 CMDITXT  EQU   18
CMDSTRUC EQU   19                 CMDSTRU  EQU   19
CMDLIST  EQU   20                 CMDLIST  EQU   20
CMDOR#LI EQU   21                 CMDORLI  EQU   21
CMDKEYW  EQU   22                 CMDKEYW  EQU   22
```

```
CMDVSN   EQU   24
CMDXTEXT EQU   25
CMDFIXD  EQU   26
CMDDEV   EQU   27
CMDPVER  EQU   28
CMDX#PAT EQU   29
CMDX#FIL EQU   35
```

```
CMDVSN   EQU   24
CMDXTEX  EQU   25
CMDFIXD  EQU   26
CMDDEV   EQU   27
CMDPVER  EQU   28
CMDXPAT  EQU   29
CMDXFIL  EQU   35
```

CMDTST
Anweisung analysieren

Der Makro CMDTST bewirkt, dass SDF

– eine Programmanweisung, die im Programm selbst abgelegt ist, analysiert und

– das Analyseergebnis an das Programm übergibt.

Bei der Analyse der übergebenen Anweisung können zusätzliche Anweisungen entstehen, indem ein System-Exit die übergebene Anweisung durch mehrere Anweisungen ersetzt. Die Behandlung dieser zusätzlichen Anweisungen liegt in der Verantwortung des Programms.

Voraussetzung ist, dass eine aktivierte Syntaxdatei die Definition des Programms und seiner Anweisungen enthält.

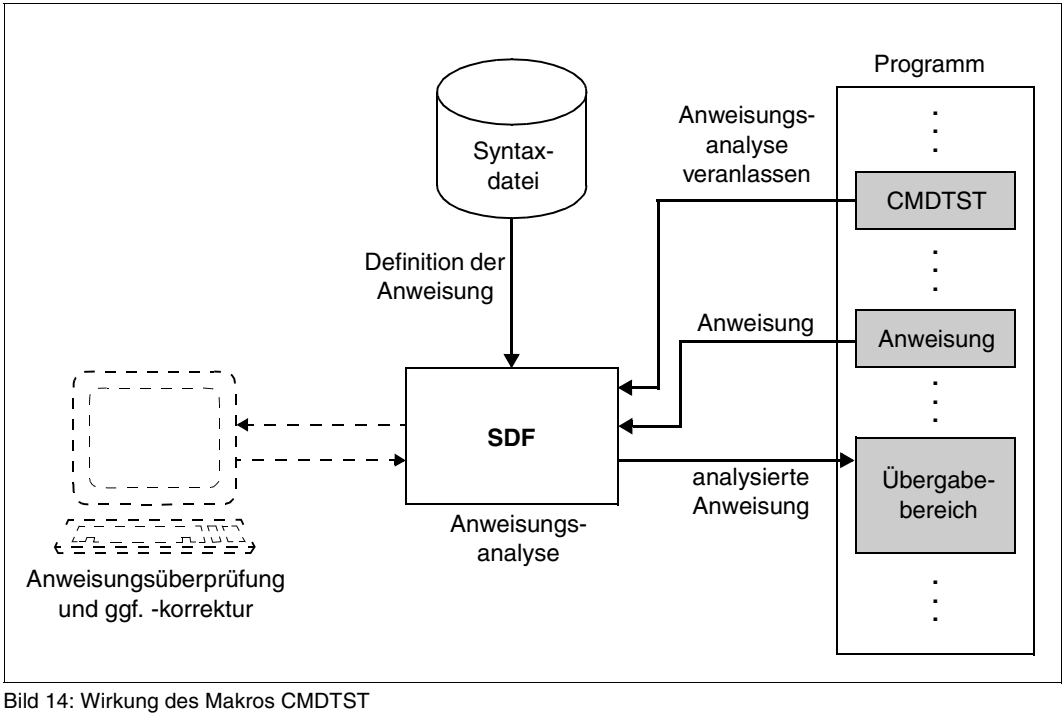


Bild 14: Wirkung des Makros CMDTST

Operation	Operanden
CMDTST	PROGRAM = *NONE / <c-string 1..8> / <var: char:8>
	,EXTNAME = *NONE / <c-string 1..8> / <var: char:8>
	,INPUT = *NO / <var: pointer>
	,OUTPUT = <var: pointer>
	,STMT = *ALL / <var: pointer>
	,DIALOG = *NO / *YES / *ERROR / <var: enum-of DIALOG_S:1>
	,MESSAGE = *NO / <var: pointer>
	,PROT = *YES / *NO / <var: bit:1>
	,BUFFER = *NO / <var: pointer>
	,INVAR = *NO
	,DEFAULT = *NO / <var: pointer>
	,ERROR = *NO / *YES / <var: bit:1>
	,CALLID = *NO / <var: pointer>
	,EXECUTE = *NO / *YES / <var: bit:1>
	,PROCMOD = *ANY / *NO / *YES / <var: enum-of PROCEDURE_S:1>
	,CCSNAME = *NO / *EXTEND / <c-string 1..8> / <var: char:8>
	,INPUTSV = *NO / *YES / <var: bit:1>
	,OUTFORM = *NEW / *OLD / <var: bit:1>
	,DEFDEF = *NO / *YES / <var: bit:1>
	,MF = C / D / L / M / E

PROGRAM =
interner Name des Programms, das den Makroaufruf absetzt. In der Syntaxdatei ist dieser Name in der Programmdefinition abgelegt[1)] (siehe Anweisung ADD-PROGRAM, Operand INTERNAL-NAME; Seite 162).

***NONE**
Der interne Name des Programms wird nicht angegeben. In diesem Fall muss im Operanden EXTNAME der externe Name des Programms angegeben werden.

<c-string 1..8> / <var: char:8>
Der interne Name des Programms wird als C-String-Konstante oder als String-Variable übergeben. Er ist mindestens ein und maximal acht Byte lang.

[1)] Wird die programmspezifische Syntaxdatei mit CMDTST überhaupt erst zugewiesen, so ist als Programmname CMDEDIT anzugeben.

EXTNAME =
externer Name des Programms, das den Makroaufrauf absetzt. In der Syntaxdatei ist dieser Name in der Programmdefinition abgelegt (siehe Anweisung ADD-PROGRAM, Operand NAME, Seite 162).

***NONE**
Der externe Name des Programms wird nicht angegeben. In diesem Fall muss im Operanden PROGRAM der interne Name des Programms angegeben werden.

<c-string 1..8> / <var: char:8>
Der externe Name des Programms wird als C-String-Konstante oder als String-Variable übergeben. Er ist mindestens ein und maximal acht Byte lang.

INPUT =
bestimmt, welche Anweisung SDF analysieren soll.

***NO**
SDF soll keine im Programm abgelegte Anweisung analysieren, sondern eine durch System-Exit zusätzlich bereitgestellte Anweisung.

<var: pointer>
SDF soll die Anweisung analysieren, deren Adresse angegeben ist. Der Bereich mit der angegebenen Adresse muss auf Halbwortgrenze ausgerichtet sein. SDF erwartet die Anweisung in folgendem Format:

2 Byte: absolute Länge des Bereiches (n+4)
2 Byte: (reserviert)
n Byte: Anweisung als Zeichenkette

OUTPUT = <var: pointer>
Adresse des normierten Übergabebereichs, der auf Wortgrenze beginnen muss. Der Übergabebereich wird mit dem Makro CMDTA generiert (oder bei OUTFORM=*OLD mit CMDSTRUC, siehe Kapitel „Anhang“ auf Seite 605).

STMT =
bestimmt, welche Anweisungen als Eingabe zulässig sind.
Nur die Anweisungen, deren interner Anweisungsname angegeben ist, sind zulässig. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT). Er ist mindestens ein und maximal acht Byte lang. Die SDF-Standardanweisungen sind unabhängig von der hier getroffenen Festlegung immer zulässig.

***ALL**
Alle Anweisungen sind zulässig.

<var: pointer>
Adresse der Liste der zulässigen Anweisungen. Diese Liste kann mit dem Makro CMDALLW generiert worden sein.
Die Liste muss auf Halbwortgrenze ausgerichtet und wie folgt aufgebaut sein:

2 Byte: Anzahl der internen Namen in der Liste (n)
8 Byte: erster interner Anweisungsname
. . .
8 Byte: n-ter interner Anweisungsname

DIALOG =
bestimmt, ob SDF bei der Anweisungsanalyse einen Dialog führen soll. Dieser Operand ist nur relevant, wenn das Programm in einer interaktiven Task abläuft.

***NO**
SDF soll keinen Dialog führen.

***YES**
SDF soll die vom Programm übergebene Anweisung dem Benutzer im Dialog zu einer eventuellen Änderung anbieten, soweit das mit den geltenden SDF-Festlegungen für den Dialog verträglich ist (siehe MODIFY-SDF-OPTIONS und SET-GLOBALS).

***ERROR**
SDF soll nur beim Erkennen von Syntaxfehlern einen Dialog führen. Falls die Anweisung Semantikfehler enthält, kann das Programm mit CMDCST einen Semantikfehlerdialog anstoßen.

<var: enum-of DIALOG_S:1>
Aufzählungs-Variable, die folgende Werte annehmen kann:
0 : wie *NO
1 : wie *YES
2 : wie *ERROR

MESSAGE =
bestimmt, ob SDF eine Meldung ausgeben soll, wenn dem Benutzer die Anweisung zur Überprüfung und ggf. Änderung angeboten wird (nur relevant für DIALOG≠*NO). SDF integriert diese Meldung in den Fragebogen.

***NO**
SDF soll keine Meldung ausgeben.

<var: pointer>
Adresse des auszugebenden Meldungstextes, auf Halbwortgrenze ausgerichtet. Der Text wird als Satz variabler Länge erwartet:

2 Byte: absolute Länge des Satzes (n+4)
2 Byte: (reserviert)
n Byte: Meldungstext

Der Meldungstext darf maximal 400 Zeichen lang sein. In SDF-formatierten Bildschirmen werden jedoch nur die ersten 280 Zeichen dargestellt. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden.

PROT =
bestimmt, ob SDF zu protokollierende Eingaben und Meldungen nach SYSOUT schreibt. Falls nicht nach SYSOUT geschrieben wird, sollte der Benutzer des Programms in der Programmdokumentation darüber informiert werden. Ein Protokollpuffer kann bereitgestellt werden (siehe BUFFER).

***NO**
SDF soll keine Protokollierung durchführen.

***YES**
SDF soll zu protokollierende Eingaben und Meldungen nach SYSOUT schreiben.

<var: bit:1>
Bitvariable: bit = 0: wie *NO
bit = 1: wie *YES

BUFFER =
Das Protokoll der Anweisung und die Fehlermeldungen können in einen vom Benutzer bereitgestellten Bereich geschrieben werden.

***NO**
Es wird kein Puffer bereitgestellt.

<var: pointer>
Adresse eines Bereichs, in dem das Protokoll der angegebenen Anweisung und die Meldungen, unabhängig vom Wert, der bei PROT angegeben wurde, abgelegt wird. Der Bereich muss auf Halbwortgrenze ausgerichtet sein und wird wie folgt belegt:

2 Byte: maximale Länge des Protokollbereiches
2 Byte: tatsächlich genutzte Länge des Protokollbereiches
n Byte: Protokollsätze

Jeder einzelne Protokollsatz hat folgendes Format:

2 Byte: absolute Länge des Protokollsatzes (m+4)
2 Byte: (reserviert)
m Byte: Inhalt des Protokollsatzes

Wenn der Puffer nicht leer ist, ist der erste Datensatz normalerweise das Protokoll des Eingabe-Kommandos. Die weiteren Datensätze enthalten Meldungen. Wenn kein Eingabeprotokoll zur Verfügung steht oder ausgegeben werden kann, wird ein doppelter Schrägstrich („//“) in den Ausgabebereich geschrieben.
Die Ausrichtung der Protokollsätze ist nicht garantiert.

INVAR =
legt fest, ob die INVARIANT-INPUT-Form der Anweisung abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task zu dieser Zeit erlaubt sind. Die INVARIANT-INPUT-Form ist damit die größtmögliche Eingabeform für eine Anweisung, die für einen Benutzer mit bestimmten Privilegien und im gewählten Dialogmodus zulässig ist. Im Gegensatz zur Protokollierungsform LOGGING=*INVARIANT-FORM (Anweisung MODIFY-SDF-OPTIONS) werden Kennwörter und geheime Operanden jedoch **nicht** ausgeblendet.

***NO**
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

<var: pointer>
gibt die Adresse eines Puffers an, in den SDF die INVARIANT-INPUT-Form der Anweisung schreibt. Der Puffer muss auf Wortgrenze ausgerichtet sein und das erste Halbwort muss die Länge des Puffers enthalten. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab.
Der Puffer hat dann folgenden Inhalt:

2 Byte: maximale Länge des Puffers
2 Byte: Ausgabelänge, die SDF schreibt (n+4)
2 Byte: (reserviert)
n Byte: INVARIANT-INPUT-Form der Anweisung, ab 7. Byte

DEFAULT =
bestimmt, ob SDF folgende Werte durch vom Programm dynamisch erzeugte Werte ersetzt:
– eingegebene Operandenwerte oder
– Default-Werte der Operanden

In der Syntaxdatei müssen die Operanden bzw. Operandenwerte entsprechend definiert sein (siehe ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES),... bzw. ADD-VALUE..., VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). Der vom Programm erzeugte Default-Wert muss gültiger Operandenwert sein.
Im geführten Dialog zeigt SDF die vom Programm erzeugten Werte im Fragebogen.

Beispiel:
SDF ersetzt in den eingegebenen MODIFY-Anweisungen den Wert *UNCHANGED durch den aktuellen Wert.

***NO**
SDF soll die eingegebenen Operandenwerte nicht durch vom Programm dynamisch erzeugte Werte ersetzen.

<var: pointer>
Adresse einer auf Wortgrenze ausgerichteten Liste, die Adressen von Umsetzbeschreibungen für Anweisungen enthält. Als Umsetzbeschreibung wird ein formatierter Übergabebereich mit dem Typ 'Struktur' verwendet (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff). Je Anweisung kann nur eine Umsetzbeschreibung angegeben werden. Eine Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche Operanden mit welchen Werten zu belegen sind. Die Liste der Adressen von Umsetzbeschreibungen ist wie folgt aufgebaut:

2 Byte: Anzahl der Umsetzbeschreibungen in der Liste (n)
2 Byte: (reserviert)
4 Byte: Adresse der ersten Umsetzbeschreibung
. . .
4 Byte: Adresse der n-ten Umsetzbeschreibung

Die Bereiche für die Umsetzbeschreibungen, die für die Default-Werte des Programms übergeben werden, müssen auf Wortgrenze ausgerichtet sein. Dies gilt ebenso für den Ausgabebereich der Makros (OUTPUT-Operand).

Stehen die zu defaultierenden Operanden in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten: Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit zu defaultierenden Operanden hängt. Auf der anderen Seite gibt der Anwender ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit zu defaultierenden Operanden hängt. SDF versucht zunächst, die vom Benutzer eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen. Ist über den struktureinleitenden Wert keine eindeutige

Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Benutzer den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

ERROR =
bestimmt, wie der beim Operanden MESSAGE angegebene Meldungstext ausgegeben wird.

<u>*NO</u>
SDF soll den Meldungstext als Meldung ausgeben.

***YES**
SDF soll den Meldungstext als Fehlermeldung ausgeben.

<var: bit:1>
Bitvariable: bit = 0: wie *NO
bit = 1: wie *YES

CALLID =
bestimmt, welcher Kontext von SDF benutzt wird, um das Kommando zu analysieren. CALLID benennt den Programmkontext (= Syntaxdateihierarchie, durch einen OPNCALL-Makro eröffnet), in dem die Anweisung analysiert werden muss.

<u>*NO</u>
Die gegenwärtig aktivierte Syntaxdateihierarchie wird verwendet.

<var: pointer>
Adresse eines 4 Byte langen Feldes, das auf Wortgrenze ausgerichtet ist. Der Aufrufende übermittelt darin die CALLID des Kontextes, den er verwenden will.

EXECUTE = <u>*NO</u> / *YES / <var: bit:1>
bestimmt, ob SDF-Standardanweisungen ausgeführt werden. EXECUTE ist ohne Bedeutung, wenn der CMDTST-Makro keinen Bezug auf eine neue Syntaxdateihierarchie nimmt (CALLID=*NO), z.B. wenn die aktuelle Syntaxdateihierarchie verwendet wird. Dann werden die SDF-Standardanweisungen immer von CMDTST ausgeführt (vorausgesetzt, sie sind in der aktuellen Syntaxdateihierarchie vorhanden). Der Operand gehört zum multihierarchischen Merkmal, das mit dem vorangehenden Operanden CALLID eingeleitet wird. Bezieht sich CMDTST auf eine parallel eröffnete Syntaxdateihierarchie (CALLID=<var: pointer>), werden die SDF-Standardanweisungen bei EXECUTE=*YES ausgeführt.

<var: bit:1>
Bitvariable: bit = 0: wie *NO
bit = 1: wie *YES

PROCMOD =
bestimmt, in welcher Umgebung der Benutzer arbeitet. SDF führt eine Prüfung durch: Anweisungen, die in der angegebenen Umgebung nicht erlaubt sind, werden von SDF nicht akzeptiert. Der Operand nimmt Bezug auf die Möglichkeit, mehrerere Syntaxdateihierarchien zu eröffnen. Er ist nur von Bedeutung, wenn der Makro-Aufruf sich auf eine neue Syntaxdateihierarchie bezieht, die zusätzlich zur aktuellen eröffnet wurde.
Ist keine CALLID bestimmt (CALLID=*NO), dann hat PROCMOD keine Bedeutung, d.h der Wert *ANY wird automatisch gesetzt und auf den aktuellen Prozedurmodus Bezug genommen.

***ANY**
Es wird keine Überprüfung vorgenommen. Die Anweisungen werden immer analysiert.

***YES**
Anweisungen werden so behandelt, als würden sie aus einer Prozedurdatei gelesen. Sie werden analysiert, wenn sie in der Syntaxdatei mit DIALOG-PROC-ALLOWED=*YES definiert sind und wenn das Programm im Dialog arbeitet, oder bei BATCH-PROC-ALLOWED=*YES in Stapelaufträgen.

***NO**
Anweisungen werden so behandelt, als würden sie von einer Hauptebene (primary level) gelesen, z.B. von der Bildschirmeingabe oder aus einem Stapelauftrag. Sie werden analysiert, wenn sie in der Syntaxdatei mit DIALOG-ALLOWED=*YES definiert wurden und wenn das Programm im Dialog läuft, oder bei BATCH-ALLOWED=*YES in Stapelaufträgen.

<var: enum-of PROCEDURE_S:1>
Aufzählungs-Variable, die folgende Werte annehmen kann:
0 : wie *ANY
1 : wie *YES
2 : wie *NO

CCSNAME =
Gibt den Namen des Zeichensatzes an, der für den Korrekturdialog auf 8-bit-Terminals und für die Konvertierung von Klein- in Großbuchstaben verwendet wird. Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Jeder codierte Zeichensatz wird durch seinen Namen (Coded Character Set Name, CCSN) bestimmt (siehe Handbuch „XHCS“ [11]). Die Ausgabe von Meldungen wird durch diesen Parameter nicht beeinflusst.

***NO**
Der 7-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

***EXTEND**
Der 8-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

<c-string 1..8> / <var: char:8>
Gibt den Namen eines speziellen 8-bit-Code an, der für Ein-/Ausgabeoperationen verwendet wird. Der Name muss 8 Byte lang sein und kann als C-String-Konstante oder als String-Variable übergeben werden.

INPUTSV =
gibt an, ob die letzten Eingaben in einer Liste gespeichert werden. Auf diese Liste kann mit Hilfe eines RESTORE-Mechanimus erneut zugegriffen werden (Anweisungen MODIFY-SDF-OPTIONS und RESTORE-SDF-INPUT).

***NO**
Die Eingaben werden nicht gespeichert.

***YES**
Die Eingaben werden in einem Puffer gespeichert.

<var: bit:1>
Bitvariable: bit = 0: wie *NO
bit = 1: wie *YES

OUTFORM =
Legt fest, welches Format des normierten Übergabebereiches SDF ausgibt. Bei der Angabe von Default-Werten (siehe Parameter DEFAULT) identifiziert SDF das Format des Übergabebereiches selbst.

***NEW**
Der normierte Übergabebereich hat das neue Format (siehe Seite 371ff).

***OLD**
Der Übergabebereich hat das bis SDF V4.0 verwendete Format (siehe Kapitel „Anhang“ auf Seite 605).

<var: bit:1>
Bitvariable: bit = 0: wie *NEW
bit = 1: wie *OLD

DEFDEF =
legt fest, ob Anweisungen zum Setzen taskspezifischer Default-Werte eingegeben werden dürfen (siehe Benutzerhandbuch „Einführung in die Dialogschnittstelle SDF“ [1]).

***NO**
Anweisungen zum Setzen taskspezifischer Default-Werte werden nicht akzeptiert.

***YES**
Anweisungen zum Setzen taskspezifischer Default-Werte werden akzeptiert.

<var: bit:1>
Bitvariable: bit = 0: wie *NO
bit = 1: wie *YES

Beschreibung der Parameter MF, PARAM, MACID und PREFIX siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8].

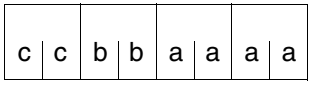
Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 371ff beschrieben. Das bis SDF V4.0 verwendete Format des Übergabebereiches finden Sie im Kapitel „Anhang“ auf Seite 605.

Mehr zur INVARIANT-INPUT-Form der Ausgabe finden Sie beim Makro CMDRST auf Seite 421.

Der Returncode wird im Standardheader der Parameterliste übergeben.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
	00	0000	erfolgreiche Beendigung:
00			– keine Besonderheiten
01			– Programm lieferte Default-Werte
00	20	0004	nicht behebbarer Systemfehler
	01	0008	Parameterfehler:
00			– falsche Parameterliste
01			– OUTPUT
02			– STMT-Liste
03			– DEFAULT
04			– MESSAGE
05			– PROT
06			– INVAR
07			– CALLID
09			– INPUT
00	40	000C	Übergabebereich zu klein

Fortsetzung ➟

(SC2)	SC1	Maincode	Bedeutung
00	40	0018	Anweisung ist zwar in Ordnung, die vom Programm übergebenen Default-Werte sind jedoch fehlerhaft
00	40	001C	Anweisung fehlerhaft
00	40	0020	Fehlerdialog nicht möglich
00	40	0024	Fehlerdialog vom Benutzer abgelehnt
00	40	0028	Puffer zu klein, Protokoll abgeschnitten
00	40	002C	END-Anweisung wurde gelesen
00	40	003C	Programm nicht in Syntaxdatei bekannt
00	40	0040	angegebene CALLID nicht gefunden
00	40	0044	im DSSM-Katalog eingetragene Syntaxdatei nicht gefunden
00	40	005C	INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten
00	20	0064	XHCS-Fehler bei Anweisungseingabe
00	40	006C	Anweisung zum Setzen taskspezifischer Defaults wurde an Stelle normaler Anweisung eingegeben

Indikator für zusätzliche Anweisungen (durch System-Exit):

Bei fehlerfreier Parameterliste wird ein Indikator für zusätzlich generierte Anweisungen im Feld <prefix><macid>EXIT_ACTIVE der Parameterliste abgelegt.

Folgende Werte können auftreten:

Wert	Bedeutung
X'00'	es existieren keine weiteren Anweisungen
X'01'	es existieren weitere Anweisungen

Migration von TRSTMT zu CMDTST

Die Migration von TRSTMT nach CMDTST ist nur dann notwendig, wenn Sie die neue Funktionalität von CMDTST nutzen wollen. Dabei sind dieselben Punkte wie bei CMDRST zu beachten (siehe „Migration von RDSTMT zu CMDRST“ auf Seite 422ff). Zusätzlich dazu gibt es folgende Änderungen:

TRSTMT	CMDTST
`PROT = YES/NO/addr/(r)`	`PROT = *YES/*NO` `BUFFER = *NO/identifier`
`INPUTSAV = NO/YES`	`INPUTSV = *NO/*YES (umbenannt!)`

Der Makro-Returncode wird im Standardheader der Parameterliste übergeben. Der Maincode von CMDTST entspricht den Werten, die bei TRSTMT im rechtsbündigen Byte im Register 15 übergeben wurden. Die Erzeugung von EQUATES für Returncodes mit dem Makro CMDANALY ist nicht mehr notwendig, da diese mit CMDTST MF=D automatisch erzeugt werden. Im Folgenden finden Sie eine Gegenüberstellung der alten (TRSTMT-) und

neuen (CMDTST-)Feldnamen:

TRSTMT	**CMDTST**
&P.NOERR	&PREFIX.MDTSUCCESSFUL
&P.SYERR	&PREFIX.MDTSYSTEM_ERROR
&P.PAERR	&PREFIX.MDTPARAMETER_ERROR
&P.TRUNC	&PREFIX.MDTAREA_TOO_SMALL
&P.CERR	&PREFIX.MDTERROR_IN_STMT
&P.DFLT	&PREFIX.MDTWRONG_DEFAULTS
&P.DIMP	&PREFIX.MDTDIALOG_IMPOSSIBLE
&P.DREJ	&PREFIX.MDTDIALOG_REJECTED
&P.PTRC	&PREFIX.MDTPROTOCOL_TRUNCATION
&P.END	&PREFIX.MDTEND_STMT
&P.NOPRG	&PREFIX.MDTPROGRAM_NOT_IN_SYNTAX_FILE
&P.INCID	&PREFIX.MDTINVALID_CALLID
&P.NOFND	&PREFIX.MDTSYNTAX_FILE_NOT_FOUND
&P.ITRC	&PREFIX.MDTINVARIANT_INPUT_TRUNCATED
&P.XHCS	&PREFIX.MDTXHCS_ERROR

CMDVAL
Wert auf Datentyp überprüfen

Der Makro CMDVAL prüft, ob ein Wert zu einer SDF-Datentyp-Beschreibung passt. Der Makro übergibt das Resultat der Überprüfung und, wenn angefordert, eine SDF-Fehlermeldung in den Puffer PROT. Die Fehlermeldungen werden nicht auf SYSOUT ausgegeben. Zusätzlich kann überprüft werden, ob die Eingabezeichenfolge zu einem vorgegebenen Wildcard-Suchmuster passt.

Operation	Operanden
CMDVAL	INPUT = addr DATATYP = { NOCHECK / INTEGER / XSTRING / CSTRING / NAME / ALPHANAME / STRUCNAME / FILENAME / PARTFILE / TIME / DATE / COMPONAME / TEXT / CATID / KEYWORD / KEYNUMBER / VSN / XTEXT / FIXED / DEVICE / PRODVERS / POSIXPATH / POSIXFILE } ,SHORTST = *ANY / integer ,LONGEST = *ANY / integer ,WCLOGL = *NONE / integer ,LOWDEC = 0 / integer ,HIGDEC = 0 / integer ,CONST = *NO / addr / (addr, ...) ,PATTERN = *NO / addr ,ATTRIB = { *NONE / ([NOCATID] [,NOUSERID] [,NOGENERATION] [,NOVERSION] [,WILDCARD] [,KEYSTAR] [,NOSEPERATORS] [,UNDERSCORE] [,NOODD] [,NOALIAS] [,VOLUMEONLY] [,NOUSERINT] [,NOCORSTATE] [,ANYCORSTATE] [,WILDCONST] [,LOWERCASE] [,NOTEMPFILE] [,QUOTESMAND] [,ANYUSERINT] [,STDDISK]) } [,DEVCLAS = list-poss(2): DISK / TAPE] [,EXCDISK = *NONE / addr / (text8,...)] [,EXCTAPE = *NONE / addr / (text8,...)]

Fortsetzung ➠

Operation	Operanden
CMDVAL (Forts.)	,PROT = *NO / addr ,PREFIX = C / p ,MACID = MDV / mac [,MF = D / C / L / E] [,PARAM = addr]

INPUT = addr
Zeichenfolge eines Wertes im variablen Satzformat (erstes Halbwort: Länge des Satzes, zweites Halbwort: Zeichen, mit dem auf die angegebene Satzlänge aufgefüllt wurde). Leerzeichen als Teil des Eingabewerts sind nicht erlaubt (Ausnahme: Leerzeichen innerhalb eines Wertes vom Typ <c-string> oder <text>).
addr muss auf Wortgrenze ausgerichtet sein.

DATATYP =
Angabe des Datentyps, auf den der Wert überprüft wird (siehe auch Anweisung ADD-VALUE TYPE=...).

NOCHECK
Keine Datentypüberprüfung erfolgt. Es soll nur geprüft werden, ob die eingegebene Zeichenfolge zu einem Wildcard-Suchmuster passt. Der Parameter PATTERN ist bei Angabe von NOCHECK Pflichtparameter.

Die folgende Tabelle zeigt die möglichen Datentypüberprüfungen:

DATATYP =	Datentyp, auf den der Wert überprüft wird
INTEGER	<integer>
XSTRING	<x-string>
CSTRING	<c-string>
NAME	<name>
ALPHANAME	<alphanum-name>
STRUCNAME	<structured-name>
FILENAME	<filename>
PARTFILE	<partial-filename>
TIME	<time>
DATE	<date>
COMPONAME	<composed-name>
CATID	<cat-id>

Fortsetzung ➟

DATATYP =	Datentyp, auf den der Wert überprüft wird
TEXT	Auf den Datentyp <text> wird der Wert geprüft, wenn er in einer Operation als Stellungsoperand eingegeben wurde. Falls Trennzeichen im Wert an falscher Stelle stehen, ist er unzulässig.
KEYWORD	<keyword>
KEYNUMBER	<keyword-number>
VSN	<vsn>
XTEXT	<x-text>
FIXED	<fixed>
DEVICE	<device>
PRODVERS	<product-version>
POSIXPATH	<posix-pathname>
POSIXFILE	<posix-filename>

SHORTST =
Bestimmt, ob die Zeichenfolge eine Mindestlänge haben muss (siehe ADD-VALUE TYPE=...,(SHORTEST-LENGTH=...)). Für die Datentypen DATE, TIME, CATID, KEYWORD und KEYNUMBER hat dieser Parameter keine Bedeutung. Beim Datentyp XSTRING ist SHORTST die Anzahl von Bytes im Wert, der bei INPUT angegeben ist. Beim Datentyp INTEGER gibt SHORTST die untere Grenze des Wertebereiches an, beim Datentyp FIXED wird SHORTST mit dem Parameter LOWDEC kombiniert. Für diese beiden Datentypen kann eine Integer-Zahl mit Vorzeichen angegeben werden.

***ANY**
Die von SDF vorgegebenen Grenzen gelten für den Datentyp.

integer
Explizite Angabe der Mindestlänge.

LONGEST =
Bestimmt, ob die Zeichenfolge eine Maximallänge nicht überschreiten darf (siehe ADD-VALUE TYPE=... (LONGEST-LENGTH=...)). Für die Datentypen DATE, TIME, CATID, KEYWORD und KEYNUMBER hat dieser Parameter keine Bedeutung. Beim Datentyp XSTRING ist LONGEST die Anzahl von Bytes im Wert, der bei INPUT angegeben ist. Beim Datentyp INTEGER gibt LONGEST die obere Grenze des Wertebereiches an, beim Datentyp FIXED wird LONGEST mit dem Parameter HIGDEC kombiniert. Für diese beiden Datentypen kann eine Integer-Zahl mit Vorzeichen angegeben werden.

***ANY**
Die von SDF vorgegebenen Grenzen gelten für den Datentyp.

integer
Explizite Angabe der Maximallänge.

WCLOGL =
Nur relevant für ATTRIB=WILDCARD. Der bei INPUT angegebene Wert kann Wildcards enthalten. WCLOGL gibt die maximale Länge der Werte an, zu dem das Wildcard-Suchmuster passt, während LONGEST die tatsächliche Länge des Eingabewertes festlegt. Für die Datentypen POSIXPATH und POSIXFILE hat dieser Parameter keine Bedeutung.

***NONE**
Die von SDF vorgegebenen Grenzen gelten.

integer
Explizite Angabe der Maximallänge.

LOWDEC= 0 / integer
Gibt die Anzahl von Dezimalstellen beim Parameter SHORTST an und ist nur relevant beim Datentyp FIXED.

HIGDEC = 0 / integer
Gibt die Anzahl von Dezimalstellen beim Parameter LONGEST an und ist nur relevant beim Datentyp FIXED.

CONST = *NO / addr / (addr,...)
Der INPUT-Wert wird mit den hier festgelegten Konstanten verglichen. CONST ist nur für DATATYP=NOCHECK relevant. Die Konstantenwerte müssen in Sätzen mit variabler Satzlänge abgelegt sein. Ist der INPUT-Wert vom Datentyp KEYWORD oder KEYNUMBER, so kann er auch eine Abkürzung einer der Konstanten sein. Listen werden jedoch nicht unterstützt. Es können maximal 2000 Konstanten angegeben werden. Die Minimal- und Maximallänge der Konstanten richtet sich nach den Standardvorgaben für die SDF-Datentypen.

addr
Die Satzadressen sind in einem Feld zusammengefasst, das an der Adresse addr beginnt. Das Feld muss auf Wortgrenze ausgerichtet sein und es muss folgendes Format haben:

Byte	Bedeutung
0	Anzahl der Elemente im Feld (N, 2 Byte lang)
2	Füllzeichen
4	Adresse des 1. Satzes (auf Wortgrenze ausgerichtet)
8	Adresse des 2. Satzes (auf Wortgrenze ausgerichtet)
...	
N*4	Adresse des N. Satzes (auf Wortgrenze ausgerichtet)

(addr,...)
Die Satzadressen werden in einer Liste angegeben.

PATTERN = *NO / addr
Der INPUT-Wert wird mit einem Wildcard-Suchmuster verglichen.
Falls eine Überprüfung auf einen Datentyp verlangt ist, müssen die Syntaxregeln für Wildcards für diesen Datentyp eingehalten werden (u.a. die maximale Länge des Wildcard-Ausdruckes).
Bei DATATYP=NOCHECK wird nur überprüft, ob der INPUT-Wert zum angegebenen Wildcard-Suchmuster passt (keine Überprüfung auf einen Datentyp). Die Länge des Wildcard-Suchmusters ist hierfür nicht begrenzt. Die Angabe von Wildcards ist nicht für alle Datentypen erlaubt (siehe Anweisung ADD-VALUE TYPE=...).
PATTERN darf nicht zusammen mit ATTRIB=(...,WILDCARD,...) angegeben werden.
Für DATATYPE=POSIXPATH oder POSIXFILE gilt nicht die BS2000-Wildcardsyntax, sondern eine spezielle POSIX-Wildcardsyntax. Für diese beiden Datentypen darf PATTERN nicht angegeben werden.

PATTERN = addr
Gibt die Adresse eines Satzes mit variabler Satzlänge an, in dem das Wildcard-Suchmuster abgelegt ist. addr muss auf Wortgrenze ausgerichtet sein. Der Wert im Satzlängenfeld des V-Record ist die Summe aus der exakten Länge des Suchmusters und der Länge des Satzfeldes (z.B. für „ABC*“ hat das Satzlängenfeld den Wert 4+4=8). Leerzeichen sind im Suchmuster nicht erlaubt.

ATTRIB = (...)
Gibt eine Liste von Attributen an, die für den Datentyp gelten soll. Unzulässige Kombinationen werden mit einer Fehlermeldung abgewiesen. Die Attribute müssen in runden Klammern eingeschlossen sein. Das gilt auch dann, wenn die Liste nur ein Attribut enthält.
Nähere Informationen siehe Anweisung ADD-VALUE. Folgende Attribute sind möglich:

Attribut	**Bedeutung und möglicher Datentyp**
*NONE	Für die Attribute gelten die SDF-Standardvorgaben.
NOCATID	Die Angabe der Katalogkennung ist nicht erlaubt (Datentypen <filename>, <partial-filename>).
NOUSERID	Die Angabe der Benutzerkennung ist nicht erlaubt (Datentypen <filename>, <partial-filename>).
NOGENERATION	Die Angabe der Generationsnummer ist nicht erlaubt (Datentyp <filename>).
NOVERSION	Die Angabe der Versionsbezeichnung ist nicht erlaubt (Datentyp <filename>).
WILDCARD	Wildcards dürfen angegeben werden. Das Attribut WILDCARD darf nicht zusammen mit dem Parameter PATTERN angegeben werden (Datentypen <alphanum-name>, <composed-name>, <filename>, <name>, <partial-filename> und <structured-name>).
KEYSTAR	Ein Stern muss der Eingabezeichenfolge vorangestellt sein (nur bei den Datentypen KEYWORD und KEYNUMBER).

Fortsetzung ➟

Attribut	Bedeutung und möglicher Datentyp
NOSEPERATORS	Trennzeichen sind nicht erlaubt (Datentyp <text>).
UNDERSCORE	Der Unterstrich '_' ist erlaubt (<name>, <composed-name>).
NOODD	Eine ungerade Anzahl von Zeichen darf eingegeben werden (<x-text>).
NOALIAS	Die Angabe des Alias-Namens ist nicht erlaubt (Datentyp <device>).
VOLUMEONLY	Der Volumetyp wird akzeptiert (Datentyp <device>).
NOUSERINT	Die Angabe der Benutzerschnittstelle ist nicht erlaubt (Datentyp <product-version>).
NOCORSTATE	Die Angabe des Korrekturstandes ist nicht erlaubt (Datentyp <product-version>). Wurde NOUSERINT in der Liste spezifiziert, so gilt NOCORSTATE automatisch auch mit.
ANYCORSTATE	Der Korrekturstand kann bei <product-version> angegeben werden. ANYCORSTATE darf nicht zusammen mit NOCORSTATE angegeben werden und umgekehrt.
WILDCONST	Der Wert kann ein Wildcard-Konstruktor sein (siehe ADD-VALUE TYPE=...(...,WILDCARD=*YES(TYPE=*CONSTRUCTOR=...)). (Datentypen <alphanum-name>, <composed-name>, <filename>, <name>, <partial-filename> und <structured-name>)
LOWERCASE	Kleinbuchstaben bleiben erhalten (Datentyp <name>).
NOTEMPFILE	Temporäre Dateinamen sind nicht erlaubt (Datentyp <filename>).
QUOTESMAND	POSIX-Pfad- und Dateinamen müssen in Hochkommas eingeschlossen sein.
ANYUSERINT	Der Freigabestand der Benutzerschnittstelle kann angegeben werden (Datentyp <product-version>). Bei gleichzeitiger Angabe von NOUSERINT ist ANYUSERINT nicht erlaubt und umgekehrt.
STDDISK	Nur Standard-Plattengeräte verwenden (Datentyp <device>).

DEVCLAS = DISK / TAPE
gibt die Geräteklasse an (Datentyp <device>, siehe Anweisung ADD-VALUE TYPE=*DEVICE(...)).

EXCDISK = *NONE / addr / (text8,...)
gibt die verbotenen Plattengeräte an (Datentyp <device>). Maximal 50 Namen können angegeben werden.

addr
gibt die Adresse eines auf Wortgrenze ausgerichteten Feldes an, in dem die Gerätenamen zusammengefasst sind. Das erste Halbwort im Feld enthält die Anzahl der Elemente im Feld, das zweite Halbwort enthält das Füllzeichen. Die Gerätenamen, die daran anschließen, sind jeweils 8 Byte lang und dürfen keine Trennzeichen enthalten. Sind die realen Gerätenamen kürzer als 8 Zeichen, so müssen sie linksbündig eingetragen und mit Leerzeichen auf die volle Länge aufgefüllt werden.

EXCDISK = (text8,...)
gibt eine Liste von 8 Zeichen langen Gerätenamen an. Für die Gerätenamen müssen die Syntaxregeln des Datentyps <text-without-sep> eingehalten werden.

EXCTAPE = *NONE / addr / (text8,...)
gibt die verbotenen Bandgeräte an. Der Parameter wird nur zusammen mit DEVICE=TAPE verwendet (siehe ADD-VALUE TYPE=*DEVICE(...)). Maximal 50 Namen können angegeben werden.

addr
gibt die Adresse eines auf Wortgrenze ausgerichteten Feldes an, in dem die Gerätenamen zusammengefasst sind. Das erste Halbwort im Feld enthält die Anzahl der Elemente im Feld, das zweite Halbwort enthält das Füllzeichen. Die Gerätenamen, die daran anschließen, sind jeweils 8 Byte lang und dürfen keine Trennzeichen enthalten. Sind die realen Gerätenamen kürzer als 8 Zeichen, so müssen sie linksbündig eingetragen und mit Leerzeichen auf die volle Länge aufgefüllt werden.

EXCTAPE = (text8,...)
gibt eine Liste von 8 Zeichen langen Gerätenamen an. Für die Gerätenamen müssen die Syntaxregeln des Datentyps <text-without-sep> eingehalten werden.

PROT = *NO / addr
Satz mit variabler Satzlänge, in den SDF die Fehlermeldung für den überprüften Wert ablegt. addr muss auf Wortgrenze ausgerichtet sein. Zum Inhalt des Satzes siehe Makro CMDRST, Parameter BUFFER (Seite 416).

Zur Beschreibung der Parameter PREFIX, MACID, MF und PARAM siehe Seite 385ff (Typen von Makroaufrufen).

Mögliche Aufrufe

```
[label]  CMDVAL MF=D [,PREFIX=p][,MACID=mac]
[label]  CMDVAL MF=C [,PREFIX=p][,MACID=mac]
[label]  CMDVAL MF=L,...
[label]  CMDVAL MF=E,PARAM=addr
```

Der Makro CMDVAL ist mit Standardheader implementiert. Die Form MF=S wird deshalb nicht angeboten.

Registerverwendung

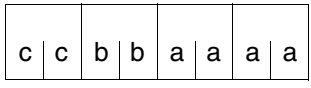
Register 1: Adresse der Parameterliste

Register 15: Returncode, der zusätzlich auch im Standardheader übergeben wird

Rückinformation und Fehleranzeigen

Der Returncode wird im Standardheader des Parameterbereiches übergeben.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	Überprüfter Wert passt zu Datentyp und/oder Wildcard-Suchmuster
01	01	0008	Der Parameter INPUT ist fehlerhaft
02	01	0008	Eine Adresse ist fehlerhaft (nicht zugewiesen,nicht auf Wortgrenze ausgerichtet,...)
03	01	0008	Fehlerhafte Angaben für: – die Kombination DATATYP/ATTRIB/PATTERN oder – die Kombination DEVCLAS/EXCDISK/EXCTAPE
04	01	0008	Fehlerhafte Angaben für Wertebereiche: – Obergrenze < Untergrenze – SDF-A-Grenzen nicht eingehalten – fehlerhafte Werte – Anzahl der Dezimalstellen nicht angegeben – Begrenzung nicht zulässig
05	01	0008	Der Parameter PATTERN ist fehlerhaft (Länge=0, Syntax fehlerhaft)
06	01	0008	Fehler im Parameter CONST
xx	40	001C	Überprüfter Wert passt nicht zum Datentyp und/oder zum Wildcard-Suchmuster. Im Puffer PROT wurde eine SDF-Fehlermeldung hinterlegt. Der Eingabewert war:
01	40	001C	– nicht vom angegebenen Datentyp bzw. hatte nicht die geforderten Attribute
02	40	001C	– keines der angegebenen Geräte
03	40	001C	– außerhalb des angegebenen oder des von SDF vorgegebenen Wertebereiches
04	40	001C	– keine der angegebenen Konstanten (Wert oder Schlüsselwort)
05	40	001C	– nicht passend zum Wildcard-Suchmuster

Hinweise

– Ist der Puffer PROT zu klein, so werden die SDF-Fehlermeldungen abgeschnitten. Es liegt in der Verantwortung des Aufrufers, diese Situation zu erkennen und darauf zu reagieren.

– Bei Überprüfung von Dateinamen mit Dateigenerationsnummer auf Übereinstimmung mit einem Wildcard-Suchmuster ist die folgende Tabelle zu beachten. Die Tabelle enthält die zurzeit geltenden Möglichkeiten, bei denen eine Übereinstimmung möglich ist. Zu Dateigenerationsgruppen (fgg, file generation group) siehe Benutzerhandbuch „Einführung in das DVS“ [7].

INPUT-Zeichenfolge	**PATTERN-Zeichenfolge**	**Übereinstimmung möglich?**
ohne fgg	ohne fgg mit fgg	ja nein
mit absoluter fgg (abs_fgg (*nnnn))	ohne fgg mit abs_fgg (*nnnn) mit rel_fgg (+/-nn)	ja ja, wenn (*nnnn) übereinstimmt nein
mit relativer fgg (rel_fgg (+/-nn)	ohne fgg mit abs_fgg (*nnnn) mit rel_fgg (+/-nn)	ja nein ja, wenn (+/-nn) übereinstimmt

abs_fgg: absolute Generationsnummer (*nnnn), 0001≤nnnn≤9999
rel_fgg: relative Generationsnummer (+/-nn), 0≤nn≤99

Diese Tabelle kann bei zukünftigen Systemerweiterungen ihre Gültigkeit verlieren und ist deshalb nicht als garantierte Schnittstelle zu betrachten.

– Suchmuster für einen Dateinamen können vom Datentyp <filename> oder <partial-filename> sein.

– Ein teilqualifizierter Dateiname kann ebenfalls als Wildcard-Suchmuster verwendet werden.

Beispiele

1. Überprüfung eines Wertes auf Datentyp:

```
        ...
        CMDVAL MF=E,PARAM=MYPL
*              returncode must be X'0140001C'
        LA     1,MYPL
        USING  MYPLD,1
        LH     2,DMDVMRET
        LTR    2,2
        BNE    ERRPROC
        ...
```

```
MYPL    CMDVAL MF=L,INPUT=BUFF@,DATATYP=FILENAME,ATTRIB=(NOCATID,WILDCARD)
        ...
BUFF@   DS     OF
BUFFL   DC     Y(BUFFEND-BUFF@)
BUFFFIL DS     XL2
BUFFCT  DC     C':OY:$TSOS.SDF-A'
BUFFEND EQU    *
        ...
MYPLD   CMDVAL MF=D,PREFIX=D
        ...
```

2. Überprüfung eines Eingabewertes auf Übereinstimmung mit einem Wildcard-Suchmuster:

```
        ...
        CMDVAL MF=E,PARAM=MYPL2
        LA     1,MYPL2
        USING  MYPLD,1
        LH     2,DMDVMRET
        LTR    2,2
        BNE    ERRPROC
        ...
MYPL2   CMDVAL MF=L,INPUT=BUFF@,PATTERN=PATT@
        ...
BUFF@   DS     OF
BUFFL   DC     Y(BUFFEND-BUFF@)
BUFFFIL DS     XL2
BUFFCT  DC     C':OY:$TSOS.SDF-A'
BUFFEND EQU    *
        ...
PATT@   DS     OF
PATTL   DC     Y(PATTEND-PATT@)
PATTFIL DS     XL2
PATTCT  DC     C':OY:$TSOS.SD/-*'
PATTEND EQU    *
        ...
MYPLD   CMDVAL MF=D,PREFIX=D
        ...
```

CMDWCC
Wildcard-Syntax prüfen und Mustervergleich durchführen

Der Makro CMDWCC prüft ein vorgegebenes Wildcard-Suchmuster auf korrekte Syntax und führt einen Mustervergleich durch. Beispiele zur Verwendung von CMDWCC finden Sie auf Seite 467f.

Operation	Operanden
CMDWCC	ACTION = list-poss(2): *CHECK / *MATCH
	,PAT@ = <var: pointer>
	,PATL = <var: int:4>
	,INP@ = NULL / <var: pointer>
	,INPL = 0 / <var: int:4>
	,FGG = *NO / *YES / <var: bit:1>
	,PART_Q = *NO / *YES / <var: bit:1>
	,SYNTAX = *BS2000 / *POSIX
	,WORK@ = NULL / <var: pointer>
	,PREFIX = C / <char:1>
	,MACID = MDW / <char:3>
	,MF = D / C / L / M / E
	,PARAM = <var: pointer>

ACTION =
legt fest, ob nur die Wildcard-Syntax geprüft oder auch ein Mustervergleich durchgeführt wird.

***CHECK**
Das Wildcard-Suchmuster wird auf korrekte Syntax geprüft.

***MATCH**
Das Wildcard-Suchmuster wird mit der bei INP@ angegebenen Zeichenkette verglichen. Das Suchmuster sollte vorher oder gleichzeitig mit *CHECK unbedingt auf korrekte Syntax geprüft werden.

PAT@ = <var: pointer>
Adresse des Wildcard-Suchmusters.

PATL = <var: int:4>
Länge des Wildcard-Suchmusters.

INP@ =
*Nur für ACTION=*MATCH:*
Gibt die Adresse einer Zeichenkette an, mit der das Wildcard-Suchmuster verglichen werden soll.

NULL
Es wird keine Vergleichszeichenkette angegeben.

<var: pointer>
Adresse der Zeichenkette.

INPL = 0 / <var: int:4>
Länge der Zeichenkette (nur für ACTION=*MATCH).

FGG =
Legt fest, ob beim Vergleich des Wildcard-Suchmusters mit der Eingabe-Zeichenkette Dateigenerationsnummern (fgg, file generation group) unterstützt werden.

***NO**
Dateigenerationsnummern werden nicht unterstützt.

***YES**
Dateigenerationsnummern werden unterstützt.

<var: bit:1>
Bitvariable: bit = 0: Dateigenerationsnummern werden nicht unterstützt.
bit = 1: Dateigenerationsnummern werden unterstützt.

PART_Q =
Legt fest, ob beim Vergleich des Wildcard-Suchmusters mit der Eingabe-Zeichenkette teilqualifizierte Dateinamen unterstützt werden.

***NO**
Teilqualifizierte Dateinamen werden nicht unterstützt.

***YES**
Teilqualifizierte Dateinamen werden unterstützt.

<var: bit:1>
Bitvariable: bit = 0: Teilqualifizierte Dateinamen werden nicht unterstützt.
bit = 1: Teilqualifizierte Dateinamen werden unterstützt.

SYNTAX =
Legt fest, welcher Typ der Wildcard-Syntax verwendet wird.

***BS2000**
Die komplette BS2000-Wildcard-Syntax kann verwendet werden.

***POSIX**
Die Wildcard-Syntax gemäß POSIX wird verwendet.

WORK@ =
*Nur für ACTION=*MATCH:*
Gibt die Adresse eines Arbeitsbereiches für den Vergleich der Eingabe-Zeichenkette mit dem Wildcard-Suchmuster an.

NULL
Es wird kein Arbeitsbereich angegeben.

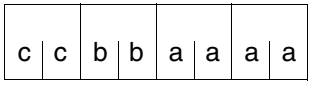
<var: pointer>
Adresse eines Arbeitsbereiches. Der Arbeitsbereich muss mindestens (5*PATL/4 + 5)*4 Bytes lang und auf Wortgrenze ausgerichtet sein.

Beschreibung der Parameter PREFIX, MACID, MF und PARAM siehe Abschnitt „Typen von Makroaufrufen“ auf Seite 385ff.

Rückinformation und Fehleranzeigen

Der Returncode wird im Standardheader des Parameterbereiches übergeben.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	kein Fehler
00	40	0001	Syntaxfehler im Suchmuster
00	40	0002	Eingabe-Zeichenkette passt nicht zum Suchmuster
00	01	0008	Parameter-Fehler
01	00	0000	kein Wildcard im Suchmuster

Hinweise

– Suchmuster für einen Dateinamen können vom Datentyp <filename> oder <partial-filename> sein.

– Ein teilqualifizierter Dateiname kann ebenfalls als Wildcard-Suchmuster verwendet werden.

– Bei Überprüfung von Dateinamen mit Dateigenerationsnummer auf Übereinstimmung mit einem Wildcard-Suchmuster ist die folgende Tabelle zu beachten. Die Tabelle enthält die zurzeit geltenden Möglichkeiten, bei denen eine Übereinstimmung möglich ist.

Zu Dateigenerationsgruppen (fgg, file generation group) siehe Benutzerhandbuch „Einführung in das DVS“ [7].

INPUT-Zeichenfolge	PATTERN-Zeichenfolge	Übereinstimmung möglich?
ohne fgg	ohne fgg mit fgg	ja nein
mit absoluter fgg (abs_fgg (*nnnn))	ohne fgg mit abs_fgg (*nnnn) mit rel_fgg (+/-nn)	ja ja, wenn (*nnnn) übereinstimmt nein
mit relativer fgg (rel_fgg (+/-nn)	ohne fgg mit abs_fgg (*nnnn) mit rel_fgg (+/-nn)	ja nein ja, wenn (+/-nn) übereinstimmt

abs_fgg: absolute Generationsnummer (*nnnn), 0001≤nnnn≤9999
rel_fgg: relative Generationsnummer (+/-nn), 0≤nn≤99

Diese Tabelle kann bei zukünftigen Systemerweiterungen ihre Gültigkeit verlieren und ist deshalb nicht als garantierte Schnittstelle zu betrachten.

Beispiele zur Verwendung von CMDWCC

```
EXWCC   START
        BASR  10,0
        USING *,10
*
* HOW TO USE CMDWCC
*
*
* EXAMPLE 1: PL1 IS INITIALIZED USING MF=L
*
EX1     LA 1,PL1
        CMDWCC MF=E,PARAM=PL1
        B EX2
PL1     CMDWCC MF=L,PAT@=A(PAT),PATL=8,INP@=A(INP),INPL=8,             *
               SYNTAX=*BS2000,ACTION=*MATCH,WORK@=A(WA)
*
* EXAMPLE 2: PL2 IS INITIALIZED USING MF=L WITH DUMMY VALUES
*            AND MODIFIED USING MF=M BEFORE EXECUTION
* NOTE: MF=M ASSUME THAT THE PARAMETER AREA IS POINTED BY REGISTER 1
*
EX2     LA 1,PL2
        USING CWCC_MDL,1
        CMDWCC MF=M,PATL=PATL,INPL=INPL,PAT@=A(PAT),INP@=A(INP),        *
               WORK@=A(WA)
        CMDWCC MF=E,PARAM=PL2
```

```
        B EX3
PL2     CMDWCC MF=L,PAT@=A(0),PATL=0,INP@=A(0),INPL=0,                *
               SYNTAX=*BS2000,ACTION=*MATCH,WORK@=A(0)
*
* EXAMPLE 3: PL3 IS INITIALIZED USING MF=L WITH DUMMY VALUES
*            AND MODIFIED USING MF=M BEFORE EXECUTION IN ANOTHER WAY
* NOTE: MF=M ASSUME THAT THE PARAMETER AREA IS POINTED BY REGISTER 1
*
EX3     LA 1,PL3
        USING CWCC_MDL,1
        LA     2,PAT
        ST     2,PATADD
        LA     2,INP
        ST     2,INPADD
        LA     2,WA
        ST     2,WAADD
        CMDWCC MF=M,PATL=PATL,INPL=INPL,PAT@=PATADD,INP@=INPADD,      *
               WORK@=WAADD
        CMDWCC MF=E,PARAM=PL3
        TERM
PL3     CMDWCC MF=L,PAT@=A(0),PATL=0,INP@=A(0),INPL=0,                *
               SYNTAX=*BS2000,ACTION=*MATCH,WORK@=A(0)
PATADD  DS     A
INPADD  DS     A
WAADD   DS     A
*
* PATTERN AND INPUT
*
PAT     DC     C'PATTERN*         '
PATL    DC     F'8'
INP     DC     C'PATTERNA         '
INPL    DC     F'8'
WA      DS     0A
        DS     XL1000
        CMDWCC MF=D
        END
```

CMDWCO
Wildcard-Konstruktion

Der Makro CMDWCO erzeugt aus einer vorgegebenen Zeichenkette, einem Auswahlmuster und einem Konstruktions-Muster eine neue Zeichenkette. Diese Ausgabe-Zeichenkette wird im Parameterbereich im Feld '<prefix><macid>NAM' und ihre Länge im Feld '<prefix><macid>LEN' abgelegt. Sowohl die Eingabe- als auch die Ausgabe-Zeichenkette können beliebige SDF-Datentypen haben. Der Benutzer sollte mit dem Makro CMDVAL selbst die Datentyp-Analyse durchführen, da hier die Eingabe-Zeichenkette nicht syntaktisch mit der Ausgabe-Zeichenkette verglichen wird.

Operation	Operanden
CMDWCO	SELECT = <c-string 1..255> / <var: char:255>
	,CONSTR = <c-string 1..255> / <var: char:255>
	,SRCNAME = <c-string 1..255> / <var: char:255>
	,MAXLEN =255 / <integer 1..255> / <var: int:1>
	,PARTIAL = *NO / *YES / <var: bit:1>
	,SYNTAX = *BS2000 / *POSIX / <var: bit:1>
	,PREFIX = C / <char:1>
	,MACID = MDW / <char:3>
	,MF = D / C / L / M / E
	,PARAM = <var: pointer>

SELECT =
gibt die Selektions-Zeichenkette an. Eine Selektions-Zeichenkette ist eine Kombination aus einem Wildcard-Suchmuster und konstanten Namensteilen (siehe Metasyntax). Die Selektions-Zeichenkette darf maximal 255 Zeichen lang sein. Ist sie kürzer, so muss sie mit mindestens einem Leerzeichen abgeschlossen sein. Bei leerer Selektions-Zeichenkette (Leerzeichen) und Angabe von PARTIAL=*YES werden alle Namen ausgewählt. Die Behandlung erfolgt wie bei teilqualifizierten Dateinamen.

<c-string 1..255>
Angabe in Form einer C-String-Konstante.

<var: char:255>
Angabe in Form einer String-Variable.

CONSTR =
gibt die Konstruktions-Zeichenkette an. Eine Konstruktions-Zeichenkette ist eine Kombination aus einem Konstruktionsmuster und konstanten Namensteilen (siehe Metasyntax). Die Konstruktions-Zeichenkette darf maximal 255 Zeichen lang sein. Ist sie kürzer, so muss sie mit mindestens einem Leerzeichen abgeschlossen sein. Bei leerer Konstruktions-Zeichenkette (Leerzeichen) und Angabe von PARTIAL=*YES werden alle Namen ausgewählt. Die Behandlung erfolgt wie bei teilqualifizierten Dateinamen.

<c-string 1..255>
Angabe in Form einer C-String-Konstante.

<var: char:255>
Angabe in Form einer String-Variable.

SRCNAME =
gibt die Zeichenkette an, auf die sich das Auswahlmuster bezieht. Die Zeichenkette darf maximal 255 Zeichen lang sein. Ist sie kürzer, so muss sie mit mindestens einem Leerzeichen abgeschlossen sein.

<c-string 1..255>
Angabe in Form einer C-String-Konstante.

<var: char:255>
Angabe in Form einer String-Variable.

MAXLEN =
gibt die maximale Länge der zu erzeugenden Ausgabe-Zeichenkette an. Ist eine erzeugte Ausgabe-Zeichenkette länger als hier angegeben, wird ein Fehlercode geliefert und im Parameterbereich wird nur die bei MAXLEN angegebene Anzahl von Zeichen eingetragen.

255 / <integer 1..255>
Angabe der maximalen Länge. Voreingestellt sind 255 Zeichen.

<var: int:1>
Angabe der maximalen Länge mit einer 1 Byte langen Integer-Variable.

PARTIAL =
Teilqualifizierung ist erlaubt. Punkte am Ende der Selektions- und der Konstruktions-Zeichenkette werden als Teilqualifizierung gedeutet. Teilqualifizierung wird auch dann angenommen, wenn Selektions- und Konstruktions-Zeichenkette leer sind. Eine leere Zeichenkette wird wie der Wildcardausdruck '*' behandelt. Auf dieses implizit gebildete Muster kann in der Konstruktions-Zeichenkette nicht mit '*' Bezug genommen werden.
Eine Zeichenkette 'xxxx.' wird wie 'xxxx.*' behandelt.

***NO**
Ein Punkt am Ende der Selektions- und der Konstruktions-Zeichenkette wird wie eine Konstante behandelt. Mit einer leeren Selektions-Zeichenkette kann nur ein leerer SRCNAME ausgewählt werden. Mit einer leeren Konstruktions-Zeichenkette kann nur ein leerer Name erzeugt werden.

***YES**
Ein Punkt am Ende der Selektions- und der Konstruktions-Zeichenkette wird als Teilqualifizierung behandelt. Das Gleiche gilt für leere Zeichenketten.

<var: bit:1>
Bitvariable: bit = 0: Teilqualifizierung ist nicht erlaubt.
bit = 1: Teilqualifizierung ist erlaubt.

SYNTAX =
Legt fest, welcher Typ der Wildcard-Syntax verwendet wird.

***BS2000**
Die komplette BS2000-Wildcard-Syntax kann verwendet werden.

***POSIX**
Die Wildcard-Syntax gemäß POSIX wird verwendet.

<var: bit:1>
Bitvariable: bit = 0: wie *BS2000
bit = 1: wie *POSIX.

Beschreibung der Parameter PREFIX, MACID, MF und PARAM siehe Abschnitt „Typen von Makroaufrufen“ auf Seite 385ff.

Rückinformation und Fehleranzeigen

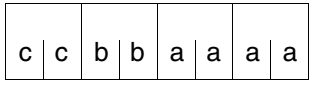
Der Returncode wird im Standardheader des Parameterbereiches übergeben.

Bei SC1=X'00' enthalten die Ausgabefelder gültige Informationen:

<prefix><macid>LEN: Länge der erzeugten Zeichenkette

<prefix><macid>NAM: Erzeugte Zeichenkette. Danach folgen Leerzeichen.

Standardheader



cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Bedeutung
00	00	0000	kein Fehler
00	40	0002	Selektion fehlerhaft oder SRCNAME nicht auswählbar
00	40	0003	Syntaxfehler in Konstruktion oder semantischer Fehler
00	01	0004	Erzeugte Zeichenkette ist länger als MAXLEN. Ausgabebereich wird nur bis MAXLEN-Länge gefüllt.
00	01	0008	Fehler im Parameterbereich (Zugriffsfehler oder Parameter nicht unterstützt)

OPNCALL
Programmkontext erstellen

OPNCALL erstellt einen Programmkontext und eröffnet die vom Aufrufer festgelegten Syntaxdateien. Bei der Angabe von *STD (= Default-Wert) wird die vom Systemkontext verwendete Syntaxdatei nochmals eröffnet. Der Makro gibt den Kontext-Identifikator (= CALLID) an den Benutzer zurück.

Mit OPNCALL kann ein dem Systemkontext entsprechender Programmkontext eröffnet werden, in dem Benutzerprogramme eine eigene Benutzersyntaxdatei aktivieren können, ohne dass die aktuelle Benutzersyntaxdatei im Systemkontext davon berührt wird. Die Benutzersyntaxdatei des Programms wird parallel zur Benutzersyntaxdatei des Systemkontexts eröffnet.

Operation	Operanden
OPNCALL	CALLID = addr / (r)] ,TASKTYP = ANYTASK / TERMINAL / BATCH / ALL ,SFSYSTM = *STD / filename ,SFGROUP = *STD / filename [,MF = { L / (E,(1)) / (E,opadr) }]

CALLID =
Adresse des Identifikators des eröffneten Kontexts, die von den weiteren Makroaufrufen bestimmt wird.

addr / (r)
Adresse eines 4 Byte langen Feldes oder Register, in dem SDF den Kontext-Identifikator übermittelt. Der Aufrufende muss den Kontext-Identifikator angeben, wenn er auf den Programmkontext Bezug nimmt. Dieses Feld muss auf Wortgrenze ausgerichtet sein.

TASKTYP =
bestimmt, in welcher Umgebung der Aufrufende bei einem Aufruf der Makros CMDTST und TRCMD arbeitet. SDF prüft, ob die angegebenen Anweisungen oder Kommandos in der festgelegten Umgebung erlaubt sind. Sind sie nicht erlaubt, werden sie von SDF abgewiesen.

ANYTASK
SDF prüft nicht, ob die Eingaben erlaubt sind.

TERMINAL
SDF akzeptiert nur die Anweisungen und Kommandos, die in den Syntaxdateien des Kontexts mit DIALOG-ALLOWED=*YES oder DIALOG-PROC-ALLOWED=*YES definiert sind (siehe Operand PROCMOD bei den Makros CMDTST und TRCMD).

BATCH
Die Anweisungen und Kommandos müssen mit BATCH-ALLOWED=*YES oder BATCH-PROC-ALLOWED=*YES definiert sein.

ALL
SDF prüft sowohl für Dialog- als auch für Stapelaufträge, ob die Anweisungen/Kommandos erlaubt sind. Wenn die Anweisung oder das Kommando für irgendeinen Auftragstyp gesperrt ist, wird es von SDF abgewiesen.

Bei nachfolgenden CMDTST- oder TRCMD-Aufrufen kann folgendes Verhalten erwartet werden:

– bei PROCMOD=ANY wird jede Anweisung und jedes Kommando abgewiesen.

– bei PROCMOD=NO werden Anweisungen und Kommandos abgewiesen, die mit DIALOG-ALLOWED=*NO oder mit BATCH-ALLOWED=*NO definiert sind.

– bei PROCMOD=YES werden Anweisungen und Kommandos abgewiesen, die mit DIALOG-PROC-ALLOWED=*NO oder BATCH-PROC-ALLOWED=*NO definiert sind.

– bei PROCMOD=ALL (nur TRCMD) werden Anweisungen und Kommandos abgewiesen, die ausschließlich im Dialog erlaubt sind (definiert mit DIALOG-PROC/ BATCH-PROC/BATCH-ALLOWED=*NO)

SFSYSTM =
bestimmt die Systemsyntaxdatei(en), die im erstellten Programmkontext aktiviert ist/sind.

***STD**
die Systemsyntaxdateien des Systemkontexts werden im Programmkontext aktiviert. Wenn diese von der Systembetreuung gewechselt werden, werden auch die Systemsyntaxdateien des Programmkontexts entsprechend verändert.

filename
Name der zu aktivierenden Systemsyntaxdatei. Die angegebene Systemsyntaxdatei kann im Programmkontext erst nach einem CLSCALL gewechselt werden. Sie gilt nur für den Benutzerauftrag, der den OPNCALL-Makro abgesetzt hat. Ein anderer Benutzerauftrag kann nur dann auf diese Syntaxdatei zugreifen, wenn er einen eigenen OPNCALL-Makro absetzt.

SFGROUP =
bestimmt die zu aktivierende Gruppensyntaxdatei.

***STD**
Die Gruppensyntaxdatei des Systemkontexts wird aktiviert. Sie ist der PROFILE-ID des Benutzerauftrags zugeordnet.

***NO**
Im Programmkontext wird keine Gruppensyntaxdatei aktiviert.

filename
Name der Gruppensyntaxdatei, die im Programmkontext eröffnet werden soll.

MF =
definiert besondere Anforderungen an die Makroauflösung (Einzelheiten siehe Seite 385ff und Handbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Registerverwendung

Register 1: Adresse der Parameterliste

Rückinformation und Fehleranzeigen

Register 15 enthält im rechtsbündigen Byte einen Returncode:

X’00’ normale Ausführung

X’44’ Syntaxdatei nicht gefunden

X’54’ maximale Anzahl von Kontexten erreicht

TRCMD
Kommando analysieren

Mit dem Makro TRCMD übergibt der Aufrufer ein Kommando als Zeichenkette an SDF. SDF analysiert dieses Kommando und generiert daraus ein Protokoll, die INVARIANT-INPUT-Form oder ACCEPTED-INPUT-Form sowie die interne Form.

Die interne Form (auch „konvertierte Form“ genannt) ist genau die Information, die zum Aufruf des Kommando-Servers und damit zur Ausführung des Kommandos notwendig ist. Die interne Form ist entweder eine Zeichenkette oder ein normierter Übergabebereich, abhängig davon, wie das Kommando in der Syntaxdatei definiert ist.

SDF übergibt an das Programm auch das Ergebnis der Syntaxanalyse.
In Dialogaufträgen kann ein Fehlerdialog zur Korrektur des Kommandos geführt werden.

Die INVARIANT-INPUT-Form der Anweisung enthält alle eingegebenen Operanden, alle durch Default-Werte vorbelegten Operanden und alle Operandenwerte, die für die Task zu dieser Zeit erlaubt sind (siehe auch CMDRST, Seite 421).

Die ACCEPTED-INPUT-Form enthält alle Namen in ihrer Langform, jeden eingegebenen Operanden mit seinem Namen und seinem Wert sowie ggf. Korrekturen. Da in der ACCEPTED-INPUT-Form keine Abkürzungen vorkommen, ist deren Eindeutigkeit auch für spätere BS2000-Versionen gewährleistet. Kennwörter und geheime Operanden sind nicht ausgeblendet.

Operation	Operanden
TRCMD	INPUT = *NO / addr / (r)
	,OUTPUT = *NO / addr / (r)
	,CMD = *ALL / (name,...)
	,DIALOG = NO / YES / ERROR
	,MESSAGE = *NO / addr / (r)
	,ERROR = NO / YES
	,PROT = *YES / *NO / addr / (r)
	,MEMORY = ACTUAL / BASIC
	,CALLINF = *NO / addr / (r)
	,EXIT = YES / NO
	,CALLID = *NO / addr / (r)
	,EXECUTE = NO / YES
	,PROCMOD = ANY / YES / NO / ALL

Fortsetzung ➠

Operation	Operanden
TRCMD (Forts.)	,CHKPRV = *YES / *NO / addr ,DUMESC = NO / YES ,OVERLNG = NO / YES ,SETVAR = NO / YES ,INVAR = *NO / addr / (r) ,ACCEPT = *NO / addr / (r) ,DEFDEF = NO / YES [,MF = { L / (E,(1)) / (E,opadr) }]

INPUT =
bestimmt, welches Kommando SDF analysieren soll.
Die Angabe des Parameters ist Pflicht.

***NO**
SDF soll kein im Programm abgelegtes Kommando analysieren, sondern das erste der durch System-Exit generierten Kommandos. Alle weiteren durch System-Exit generierten Kommandos können durch wiederholten Aufruf von TRCMD mit INPUT=*NO angefordert werden.

addr / (r)
SDF soll das Kommando analysieren, dessen Adresse angegeben ist bzw. in dem angegebenen Register steht. SDF erwartet die Kommandozeichenkette als Satz variabler Länge im üblichen BS2000-Format. Der Satzbereich muss auf Halbwortgrenze ausgerichtet sein.

OUTPUT =
bestimmt, ob die interne Form des Kommandos erzeugt wird.
Die Angabe des Parameters ist Pflicht.

***NO**
Die interne Form des Kommandos wird nicht generiert. Die Syntax des bei INPUT angegebenen Kommandos wird geprüft.

addr / (r)
Adresse eines Bereiches, in dem SDF die interne Form ablegt, bzw. Register, das diese Adresse enthält. Die interne Form kann vom Kommando-Server verarbeitet werden. In Abhängigkeit von der Kommandodefinition in der Syntaxdatei ist die interne Form entweder:

– eine Zeichenkette (siehe Anweisung ADD-CMD ...,IMPLEMENTOR=*STRING/ *PROCEDURE...) oder
– ein normierter Übergabebereich (siehe Anweisung ADD-CMD ...,IMPLEMENTOR= *NEW/*TRANSFER-AREA...).

Der Bereich muss auf Wortgrenze beginnen. Das Programm muss vor dem Aufruf von TRCMD dafür sorgen, dass im ersten Halbwort des Bereichs die maximal mögliche Bereichslänge steht.

CMD =
bestimmt, welche Kommandos als Eingabe erlaubt sind.

***ALL**
Alle Kommandos der aktuellen Syntaxdatei-Hierarchie sind erlaubt.

(name,...)
Nur die Kommandos sind erlaubt, deren RESULT-INTERNAL-NAMEs angegeben sind. Der RESULT-INTERNAL-NAME ist in der Syntaxdatei in der Kommandodefinition abgelegt (siehe Anweisung ADD-CMD). Die hier angegebenen Namen müssen vom Datentyp <alphanum-name 1..8> sein.

DIALOG =
bestimmt, ob SDF bei der Kommandoanalyse einen Dialog führen soll. Dieser Operand ist nur relevant, wenn das Programm in einer interaktiven Task abläuft.

NO
SDF soll keinen Dialog führen.

YES
SDF soll das vom Programm übergebene Kommando dem Benutzer im Dialog zu einer eventuellen Änderung anbieten, soweit das mit den geltenden SDF-Festlegungen für den Dialog verträglich ist (siehe MODIFY-SDF-OPTIONS und SET-GLOBALS).

ERROR
SDF soll nur beim Erkennen von Syntaxfehlern einen Dialog führen.

MESSAGE =
bestimmt, ob SDF eine Meldung auf SYSOUT ausgeben soll, wenn dem Benutzer das Kommando zur Überprüfung und ggf. Änderung angeboten wird (nur relevant für DIALOG≠ NO). SDF integriert diese Meldung in den Fragebogen.

***NO**
SDF soll keine Meldung ausgeben.

addr / (r)
Adresse des auszugebenden Meldungstextes (auf Halbwortgrenze ausgerichtet) bzw. Register, das diese Adresse enthält. Der Text wird als Satz variabler Länge erwartet, die maximale Länge beträgt 400 Zeichen. Im geführten Dialog werden jedoch nur die ersten 280 Zeichen dargestellt. Das erste Byte des Meldungstextes interpretiert SDF als Druckersteuerzeichen. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden.

ERROR =
bestimmt, wie der beim Operanden MESSAGE angegebene Meldungstext ausgegeben wird.

NO
SDF soll den Meldungstext als Meldung ausgeben.

YES
SDF soll den Meldungstext als Fehlermeldung ausgeben.

PROT =
Steuert die Protokollausgabe für das eingegebene Kommando.

***YES**
SDF soll die Protokoll-Form nach SYSOUT schreiben.

***NO**
SDF soll keine Protokoll-Form ausgeben.

addr / (r)
Adresse eines Protokollpuffers (auf Halbwortgrenze ausgerichtet) oder Register, das die Adresse enthält. SDF soll die Protokoll-Form des Kommandos und Meldungen nur in diesen Puffer schreiben, nicht nach SYSOUT. Die Länge des Puffers steht im ersten Halbwort des Puffers. Im zweiten Halbwort steht die tatsächlich belegte Pufferlänge. Danach folgen die Protokollsätze mit variabler Satzlänge.
Wenn der Puffer nicht leer ist, ist der erste Satz normalerweise das Protokoll des Eingabe-Kommandos. Die weiteren Sätze enthalten Meldungen aller Art. Wenn kein Eingabeprotokoll zur Verfügung steht oder ausgegeben werden kann, wird ein Schrägstrich ('/') in den Ausgabebereich geschrieben.

MEMORY =
Legt fest, ob die Definition des Kommandos aus der aktuellen Syntaxdatei-Hierarchie oder nur aus der Basis-Systemsyntaxdatei gelesen werden soll.

ACTUAL
Das Kommando wird in der aktuellen Syntaxdatei-Hierarchie analysiert.

BASIC
Nur die Kommandodefinition in der Basis-Systemsyntaxdatei wird zur Analyse herangezogen. Definitionen aus einer Gruppen- oder Benutzersyntaxdatei werden nicht beachtet.

CALLINF =
Bestimmt, ob eine Aufruf-Information für den Kommando-Server generiert wird. Wenn SDF die Kommandos MODIFY-SDF-OPTIONS und SHOW-SDF-OPTIONS ausführen soll, muss diese Aufruf-Information vom Programm übergeben werden.

CALLINF = *NO
Die Aufruf-Information wird nicht generiert. Es wird kein SDF-Kommando ausgeführt.

CALLINF = addr / (r)
Adresse eines Bereiches, der 102 Byte lang und auf Halbwortgrenze ausgerichtet ist, bzw. Register, das diese Adresse enthält. SDF legt darin die Aufruf-Information zum eingegebenen Kommando ab.

EXIT = YES / NO
Bestimmt, ob System-Exits (80/81) zur Analyse des Eingabe-Kommandos aufgerufen werden.

CALLID =
bestimmt, in welchem Kontext SDF das Kommando analysiert.

***NO**
Die gegenwärtig aktivierte Syntaxdateihierarchie wird verwendet (der Systemkontext).

addr / (r)
Adresse eines 4 Byte langen Feldes (auf Wortgrenze ausgerichtet) oder Register, das diese Adresse enthält. Der Aufrufer übermittelt darin die CALLID des Programmkontextes, den er verwenden will.

Die CALLID des Programmkontextes wird von einem vorangegangenen OPNCALL-Aufruf geliefert. Bei OPNCALL kann auch der Eingabemodus der Task festgelegt werden (Parameter TASKTYP).

EXECUTE = NO / YES
bestimmt, ob die Kommandos SHOW-SDF-OPTIONS und MODIFY-SDF-OPTIONS ausgeführt werden. Voraussetzung dafür ist, dass bei CALLINF die Aufruf-Information für das Kommando übergeben wird. EXECUTE ist ohne Bedeutung, wenn sich TRCMD nicht auf den Programmkontext, sondern auf den Systemkontext bezieht (d.h. bei CALLID=*NO). Die Kommandos werden dann immer von TRCMD ausgeführt.

PROCMOD =
bestimmt, in welcher Umgebung der Benutzer arbeitet. SDF führt eine Prüfung durch: Kommandos, die in der angegebenen Umgebung nicht erlaubt sind, werden von SDF nicht akzeptiert. Der Operand nimmt Bezug auf die Möglichkeit, mehrere Syntaxdateihierarchien zu eröffnen. Er ist nur von Bedeutung, wenn der Makro-Aufruf sich auf eine neue Syntaxdateihierarchie bezieht, die zusätzlich zur aktuellen eröffnet wurde. Bei CALLID=*NO hat PROCMOD keine Bedeutung, d.h. der Wert ANY wird automatisch gesetzt und es gilt der aktuelle Prozedurmodus. Weitere Hinweise zum PROCMOD-Parameter siehe Makro OPNCALL, Parameter TASKTYP=ALL (Seite 474).

ANY
Es wird keine Überprüfung vorgenommen.

YES
Kommandos werden so behandelt, als würden sie aus einer Prozedurdatei gelesen. D.h. sie werden analysiert, wenn sie in der Syntaxdatei mit DIALOG-PROC-ALLOWED=*YES definiert sind und wenn das Programm im Dialog arbeitet, oder bei BATCH-PROC-ALLOWED=*YES in Stapelaufträgen.

NO
Kommandos werden so behandelt, als würden sie von einer Hauptebene (primary level) gelesen, z.B. von der Bildschirmeingabe oder aus einem Stapelauftrag. Sie werden analysiert, wenn sie in der Syntaxdatei mit DIALOG-ALLOWED=*YES definiert wurden und wenn das Programm im Dialog läuft, oder bei BATCH-ALLOWED=*YES in Stapelaufträgen.

ALL
Kommandos werden auf alle Prozedurmodi geprüft. In Abhängigkeit des Parameters TASKTYP beim Makro OPNCALL ergibt sich daraus folgendes Verhalten:

- bei TASKTYP=ANY:
  Jedes Kommando wird als nicht zulässig betrachtet und abgelehnt (unzulässige Parameterkombination)
- bei TASKTYP=BATCH:
  Das Kommando wird analysiert, wenn es in der Syntaxdatei mit BATCH-ALLOWED=*YES oder BATCH-PROC-ALLOWED=*YES definiert ist.
- bei TASKTYP=TERMINAL:
  Das Kommando wird analysiert, wenn es in der Syntaxdatei mit DIALOG-ALLOWED=*YES oder DIALOG-PROC-ALLOWED=*YES definiert ist.

CHKPRV =
Bestimmt, ob SDF die Privilegien des Kommandos prüft. Der Parameter ist nur im Programmkontext von Bedeutung (d.h. bei CALLID≠*NO).

<u>*YES</u>
Die Privilegien werden in Abhängigkeit vom erlaubten Eingabemodus geprüft (TRCMD PROCMOD=..., OPNCALL TASKTYP=...).

***NO**
Die Privilegien des Kommandos werden nicht geprüft. Das Kommando wird nicht ausgeführt, d.h. es gilt immer EXECUTE=NO.

addr
Adresse eines 8 Byte langen Bereiches, in dem eine 64-Bit-Privilegienmaske abgelegt ist. Gegen diese Privilegienmaske wird geprüft, ob das Kommando in einem bestimmten Eingabemodus erlaubt ist. Die Reihenfolge der Privilegien in der Bitmaske entspricht der Reihenfolge, in der die Privilegien aktuell im System definiert sind. Diese Privilegienmasken werden u.a. auch mit der SDF-I-Anweisung SHOW-SYNTAX-FILE ausgegeben (siehe Handbuch „SDF-Verwaltung“ [2]).

DUMESC =
Legt fest, ob Ausdrucksersetzungen, die mit einem Escape-Zeichen beginnen, als Teil des Kommandos betrachtet oder ignoriert werden. Ausdrucksersetzungen werden üblicherweise in der Form '&proc-parameter' bzw. '&(ausdruck)' angegeben.

<u>NO</u>
Ausdrucksersetzungen werden von SDF als Teil des Kommandos betrachtet und syntaktisch auch so analysiert. In den meisten Fällen (außer bei <text> und <cmd-rest>) wird SDF deshalb Syntaxfehler melden.

YES
Ausdrucksersetzungen, die mit dem Escape-Zeichen beginnen, werden von SDF nicht syntaktisch analysiert. Das Schüsselwort BY-AND wird ebenfalls ignoriert.

OVERLNG =
Bestimmt, ob SDF das Längenfeld des bei OUTPUT angegebenen Bereiches (erstes Halbwort) mit der tatsächlichen Länge der internen Form (ISP-Zeichenkette oder normierter Übergabebereich) überschreiben soll. Das geschieht jedoch nur bei erfolgreichem Funktionsaufruf und wenn der OUTPUT-Bereich nicht zu kurz ist.

<u>NO</u>
Das Längenfeld des OUTPUT-Bereiches wird nicht überschrieben.

YES
Das Längenfeld wird bei erfolgreichem Funktionsaufruf mit der tatsächlichen Länge (einschließlich Längenfeld) der internen Kommando-Form überschrieben.

SETVAR =
Legt fest, ob das Kommando eine Zuweisung der Form '<variable-name>=wert' sein kann, wenn es in einer S-Prozedur eingegeben wurde. In Nicht-S-Prozeduren kann es dabei zu Konflikten mit zulässigen ISP-Eingaben für /REMARK, /TYPE, /PAUSE usw. kommen.

NO
Gleichheitsszeichen werden als Teil der Operandenliste des Kommandos betrachtet. Aus Kompatibilitätsgründen ist NO Voreinstellung.

YES
Gleichungen werden als Variablenzuweisung interpretiert. Als Kommandoname wird standardmäßig /SET-VARIABLE verwendet. Wenn dieses Kommando in der Syntaxdatei-Hierarchie nicht definiert ist, wird das eingegebene Kommando mit einer entsprechenden Fehlermeldung abgewiesen. Wenn SDF die Variablenzuweisung erfolgreich identifizieren konnte, werden die geforderten Ausgaben anhand der Kommandodefinition von SET-VARIABLE gebildet.

INVAR =
Legt fest, ob die INVARIANT-INPUT-Form des Kommandos abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task in Abhängigkeit vom Eingabemodus und von den Privilegien erlaubt sind:

1. Der Eingabemodus (Parameter PROCMOD) wird nur geprüft, wenn auch CALLID angegeben wurde. Die Abhängigkeiten zwischen CALLID und PROCMOD sind beim Parameter PROCMOD beschrieben.
2. Die Privilegien bei CHKPRV werden nur beachtet, wenn auch CALLID angegeben wird. Ohne die Angabe einer CALLID verwendet TRCMD die aktuellen Task-Privilegien.

Kennwörter und geheime Operanden werden nicht ausgeblendet.
INVAR darf nicht zusammen mit ACCEPT angegeben werden.

***NO**
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

INVAR = addr / (r)
Gibt die Adresse eines auf Wortgrenze ausgerichteten Puffers an, in den SDF die INVARIANT-INPUT-Form des Kommandos schreibt. Das erste Halbwort muss die Länge des Puffers enthalten. Falls der Puffer zu kurz ist, wird ein entsprechender Makro-Returncode zurückgegeben. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab.

Der Puffer hat dann folgenden Inhalt:

2 Byte	2 Byte	2 Byte	
buflen	reclen	filler	invariant-input

buflen: Länge des Puffers
reclen: Länge des Satzes, den SDF schreibt
filler: Füllzeichen
invariant-input: INVARIANT-INPUT-Form des Kommandos, beginnt beim 7. Byte.

ACCEPT =
Legt fest, ob die ACCEPTED-INPUT-Form des Kommandos abgespeichert wird. Die ACCEPTED-INPUT-Form enthält alle Namen in ihrer Langform und alle eingegebenen Operanden mit ihren Namen und Werten, die für die Task in Abhängigkeit vom Eingabemodus und von den Privilegien erlaubt sind:

1. Der Eingabemodus (Parameter PROCMOD) wird nur geprüft, wenn auch CALLID angegeben wurde. Die Abhängigkeiten zwischen CALLID und PROCMOD sind beim Parameter PROCMOD beschrieben.
2. Die Privilegien bei CHKPRV werden nur beachtet, wenn auch CALLID angegeben wird. Ohne die Angabe einer CALLID verwendet TRCMD die aktuellen Task-Privilegien.

Kennwörter und geheime Operanden werden nicht ausgeblendet.
ACCEPT darf nicht zusammen mit INVAR angegeben werden.

ACCEPT = *NO
Die ACCEPTED-INPUT-Form der Anweisung wird nicht abgespeichert.

ACCEPT = addr / (r)
Gibt die Adresse eines auf Wortgrenze ausgerichteten Puffers an, in den SDF die ACCEPTED-INPUT-Form des Kommandos schreibt. Das erste Halbwort muss die Länge des Puffers enthalten. Falls der Puffer zu kurz ist, wird ein entsprechender Makro-Returncode zurückgegeben. SDF legt die ACCEPTED-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab.

Der Puffer hat dann folgenden Inhalt:

2 Byte	2 Byte	2 Byte	
buflen	reclen	filler	accepted-input

buflen: Länge des Puffers
reclen: Länge des Satzes, den SDF schreibt
filler: Füllzeichen
accepted-input: ACCEPTED-INPUT-Form des Kommandos, beginnt beim 7. Byte.

DEFDEF =
legt fest, ob Kommandos zum Setzen taskspezifischer Default-Werte eingegeben werden dürfen (siehe Benutzerhandbuch „Einführung in die Dialogschnittstelle SDF“ [1]).

NO
Kommandos zum Setzen taskspezifischer Default-Werte werden nicht akzeptiert.

YES
Kommandos zum Setzen taskspezifischer Default-Werte werden akzeptiert. Die gesetzten Default-Werte sind solange gültig, bis sie explizit zurückgesetzt werden. SDF übergibt einen speziellen Returncode, wenn es eine Default-Wert-Definition erkennt, und ignoriert in diesem Fall die Ausgabe-Parameter. Bei Syntaxfehlern wird der dafür übliche Returncode zurückgegeben.

MF =
definiert besondere Anforderungen an die Makroauflösung (siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 371ff beschrieben.

Register 15 enthält im rechtsbündigen Byte einen Returncode:

X’00’ normale Beendigung; das Kommando ist syntaktisch korrekt

X’04’ nicht behebbarer Systemfehler

X’08’ Operandenfehler im Makroaufruf

X’0C’ Übergabebereich zu klein

X’10’ EOF erkannt

X’1C’ Kommando fehlerhaft oder unbekannt

X’20’ Fehlerdialog nicht möglich

X’24’ Fehlerdialog wurde abgelehnt

X’28’ Fehler- oder Protokollbereich zu klein

X’30’ Timeout

X’38’ SDF ist nicht verfügbar

X’44’ Syntaxdatei nicht gefunden

X’48’ SDF-Kommando ausgeführt

X’5C’ INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten

X’6C’ Kommando zum Setzen taskspezifischer Default-Werte erkannt

X’70’ übergebene Zeichenkette ist leer

Indikator für zusätzliche Kommandos (durch System-Exit):

Wenn durch System-Exit eine 1:n-Umsetzung des eingegebenen Kommandos stattgefunden hat, zeigt das Flag FURTH in der Aufruf-Information (CALLINF) an, dass weitere Kommandos mit TRCMD INPUT=*NO angefordert werden können.

6.5 Schnittstelle zwischen SDF und höheren Programmiersprachen

SDF bietet ab Version 3 eine Schnittstelle für Programme höherer Programmiersprachen (High Level Languages, HLL) wie COBOL, FORTRAN und C an. Sie können damit von solchen Programmen aus auf die SDF-Makros zugreifen, ohne Assemblerprogramme zum Aufruf der SDF-Makros zu schreiben. Das folgende Bild zeigt das Prinzip der HLL-Schnittstelle.

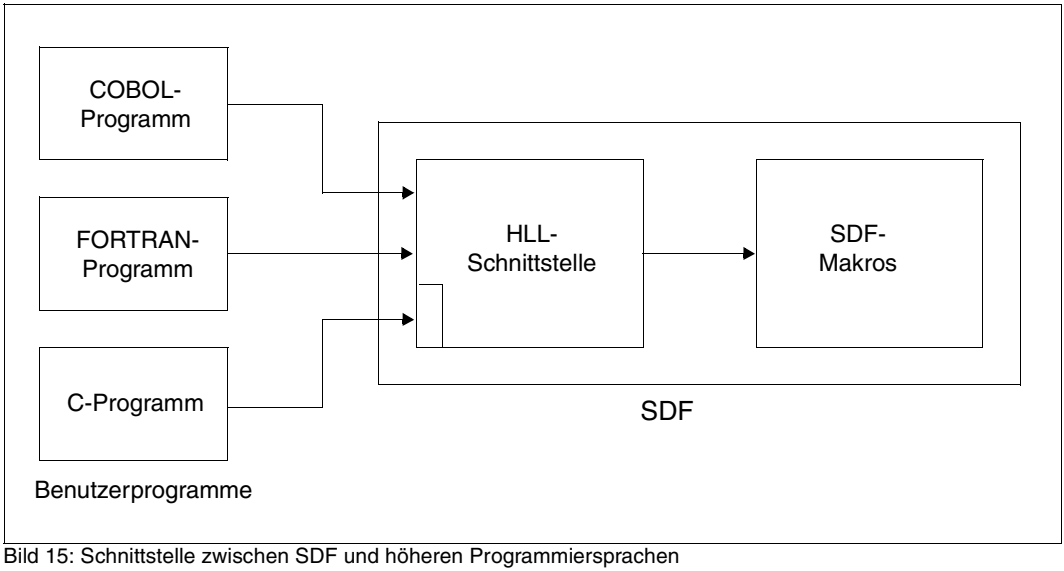


Bild 15: Schnittstelle zwischen SDF und höheren Programmiersprachen

Folgende SDF-Makros werden von der HLL-Schnittstelle unterstützt:

– Makros zur Bearbeitung von Anweisungen (RDSTMT, CORSTMT, TRSTMT)

– der Makro zum Aufruf eines System-Kommandos (CMD)

– der Makro zum Setzen des Kommando-Returncodes (CMDRC)

– der Makro zum Ausgeben von Informationen über aktivierte Syntaxdateien (CMDSTA).

Der normierte Übergabebereich kann über die HLL-Schnittstelle ausgewertet werden. Zur Informationsgewinnung aus dem normierten Übergabebereich bietet die HLL-Schnittstelle einige zusätzliche Funktionen, die die Analyse des Übergabebereiches erleichtern.

Die Schnittstelle zu höheren Programmiersprachen unterstützt nur den normierten Übergabebereich im alten Format (siehe Seite 605). Die Nutzung des neuen Formats (siehe Seite 371ff) ist nur über die Assembler-Schnittstelle möglich!

6.5.1 Schnittstellenkonventionen

Für alle Programmiersprachen, die dieselben Konventionen wie COBOL und FORTRAN für Übergabeparameter an Unterprogramme verwenden, ist die HLL-Schnittstelle direkt verfügbar. Für diese Programmiersprachen gelten die ab Seite 489 beschriebenen Funktionsaufrufe.

Folgende Konventionen gelten:

– Register 1 muss einen Zeiger auf die Liste der Adressen der Übergabeparameter enthalten,

– das höchstwertige Bit der letzten Parameteradresse muss gesetzt sein (logisch OR-verknüpft mit X'80000000'),

– Register 13 enthält die Adresse des Registersicherungsbereiches (wird normalerweise vom Programm-Manager gesetzt).

Aufrufe, die sich an die ILCS-Konventionen halten, werden ebenfalls unterstützt. Dafür muss:

– Register 1 einen Zeiger auf die Liste der Parameteradressen enthalten,

– Register 0 die Anzahl der Parameter enthalten,

– Register 13 die Adresse des Registersicherungsbereiches enthalten (wird normalerweise vom Programm-Manager gesetzt).

Für die Programmiersprache C wurde eine extra Schnittstelle entwickelt, die diese Anforderungen erfüllt. Die C-Schnittstelle ist in einem separaten Abschnitt beschrieben (siehe Seite 509ff).

Unter bestimmten Voraussetzungen kann die HLL-Schnittstelle auch von Assemblerprogrammen verwendet werden. Folgende Voraussetzungen sind notwendig:

– die erforderliche Parameterliste muss aufgebaut sein,

– die Adresse der Parameterliste muss in Register 1 an SDF übergeben werden,

– Register 13 muss die Adresse eines 18 Worte langen Registersicherungsbereiches enthalten, der vor dem Aufruf gelöscht werden muss,

– der Entry-Name ist „SDF“.

Sowohl die Interface-Module als auch die Include-Elemente stehen in der Bibliothek SYSLIB.SDF.045 zur Verfügung.

6.5.2 COBOL- und FORTRAN-Schnittstelle

6.5.2.1 Beschreibung der Funktionsaufrufe

Funktionsaufrufe

Die Beschreibung der Funktionsaufrufe erfolgt in der Form

```
CALL SDF(<parameterliste>)
```

Der Inhalt von <parameterliste> ist den jeweiligen Funktionsaufrufen zu entnehmen. Die genaue Syntax des Aufrufes ist sprachenabhängig.

Metasyntax

Da die genaue Syntax der Funktionsaufrufe sprachenabhängig ist, wird in den folgenden Abschnitten eine Umschreibung verwendet, wobei die Parameterliste jeweils in Klammern angegeben wird. Eckige Klammern innerhalb der Parameterliste bedeuten, dass der oder die Parameter optional sind.

Daran anschließend erfolgt die Beschreibung der Parameterliste in Tabellenform:

Spalte 1:	Name des Parameters	
Spalte 2:	Datentyp des Parameters mit Angabe der Länge Folgende Datentypen kommen vor:	
	char	Zeichenkette
	integer	Ganzzahl (immer 4 Byte lang)
	ptr	Adresse
	V-rec@	Adresse eines Satzes mit variabler Satzlänge
Spalte 3:	in	Eingabeparameter
	out	Ausgabeparameter
	inout	Ein-/Ausgabeparameter
Spalte 4:	Erläuterung des Parameters	

Rückinformation der Funktionen

Jede Funktion liefert eine Rückinformation in Form eines Returncodes. Dieser Returncode wird von SDF im Parameter „error“ eingetragen, der in der Parameterliste jeder Funktion enthalten sein muss.

Drei Klassen von Returncode-Werten können auftreten:

error=0: Fehlerfreie Funktionsausführung.

error>0: Fehler beim zugehörigen Makroaufruf. Der Fehlercode wurde aus Register 15 in den Parameter error kopiert. Die Werte des Fehlercodes entsprechen den Makro-Fehlercodes.

error<0:
- -1: Funktion unbekannt
- -2: zu wenig Operanden
- -3: der letzte Operand in der Struktur ist erreicht
- -4: der letzte Operand in der Liste ist erreicht
- -5: die Position ist negativ
- -6: der Operand ist nicht vom Typ LIST
- -7: der Operand ist nicht vom Typ STRUCTURE

6.5.2.2 Übersicht über die SDF-Funktionsaufrufe

CCMD	Systemkommando ausführen
CORR	Korrekturbit setzen
INIT	Pufferinitialisierung
LEVL	Auf ein Operanden-Array positionieren
OPER	Operandenwert aus dem Übergabebereich lesen
READ	Anweisung lesen und analysieren
SEMA	Semantikfehlerdialog anstoßen
STAT	Informationen über Syntaxdateien ausgeben
STMT	Anweisungsname aus dem Übergabebereich lesen
STRU	Struktureinleitenden Wert auf Datentyp und Länge analysieren
TRNS	Anweisung analysieren
TYPE	Operanden auf Datentyp und Länge analysieren
WRRC	Kommando-Returncode setzen

CCMD
Systemkommando ausführen

Die Funktion CCMD ermöglicht den Aufruf eines Kommandos, ohne den Programmodus zu verlassen. Die Funktion CCMD stützt sich auf den Makro CMD, der im Handbuch „Makroaufrufe an den Ablaufteil“ [8] beschrieben ist.

Aufruf

```
CALL SDF('CCMD',area,error,sysout,dialog,list,cmdrc,buff)
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
CCMD	char(4)	in	Funktionsname: Schlüsselwort CCMD
area	V-rec@	in	Adresse eines Satzes mit variabler Satzlänge, der das auszuführende Kommando enthält
sysout	integer	in	1 : SYSOUT = YES 0 : SYSOUT = NO
dialog	integer	in	1 : DIALOG = YES 0 : DIALOG = NO
list	integer	in	1 : LIST = YES 0 : LIST = NO
cmdrc	char(9)	in	Adresse eines 9 Byte langen Bereiches, in dem der Kommandoprozessor den Kommando-Returncode ablegen soll. Muss auf Halbwortgrenze ausgerichtet sein.
buff	V-rec@	in	Adresse eines Satzes mit variabler Satzlänge, für den Ausgabepuffer (BUFMOD=SHORT, siehe Makro CMD)

Die Parameterliste entspricht PARMOD=31 mit Angabe der Parameter CMDRC oder LIST (siehe Makro CMD, „Makroaufrufe an den Ablaufteil“ [8]).

CORR
Korrekturbit setzen

Die Funktion CORR setzt für einen bestimmten Operanden ein Korrekturbit. Das gesetzte Korrekturbit bewirkt, dass dieser Operand im Fehlerdialog zu einer Anweisung unterstrichen dargestellt wird, falls er fehlerhaft war.

Aufruf

```
CALL SDF('CORR',area,error,pos)
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
CORR	char (4)	in	Funktionsname: Schlüsselwort CORR
area	char ()	inout	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
pos	integer	in	Position des Operanden, dessen Korrekturbit gesetzt werden soll

INIT
Pufferinitialisierung

Mit der Funktion INIT wird der Puffer initialisiert, in dem der normierte Übergabebereich abgelegt wird. Der interne Name des Programmes muss dabei so angegeben werden, wie er in der Syntaxdatei definiert ist.

Die Funktion INIT muss einmalig vor allen anderen SDF-Funktionen aufgerufen werden, die den normierten Übergabebereich verwenden.

Aufruf

```
CALL SDF('INIT',area,error,lng,program)
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
INIT	char(4)	in	Funktionsname: Schlüsselwort INIT
area	char(lng)	inout	Puffer zum Anlegen des normierten Übergabebereiches, muss auf Wortgrenze ausgerichtet sein
error	integer	out	Returncode
lng	integer	in	Länge des Puffers für den normierten Übergabebereich
program	char(8)	in	Interner Programmname (wie in der Syntaxdatei definiert)

LEVL
Auf ein Operanden-Array positionieren

Mit der Funktion LEVL positionieren Sie auf ein Operanden-Array , das zu einer Strukturbeschreibung gehört (siehe Abschnitt „Aufbau des normierten Übergabebereichs“ auf Seite 371ff).

Nach dem Aufruf der Funktion INIT ist zunächst auf das Operanden-Array der höchsten Ebene positioniert, d.h auf das Operanden-Array, dessen Operanden mit der SDF-A-Anweisung ADD-OPERAND ...,RESULT-OPERAND-LEVEL=1 definiert wurden.

Die Funktion LEVL bezieht sich immer auf das zurzeit aktuelle Operanden-Array. Das kann auch das Operanden-Array einer Strukturbeschreibung sein, wenn die Funktion LEVL zuvor schon einmal aufgerufen wurde.

Die Strukturbeschreibung kann auch Element einer Liste sein. In diesem Fall müssen Sie sowohl die Position der Liste im aktuellen Operanden-Array als auch die Position der Strukturbeschreibung in dieser Liste angeben.
Ist die Strukturbeschreibung nicht Element einer Liste, so müssen Sie nur die Position der Strukturbeschreibung im Operanden-Array angeben.

Um zum Operanden-Array der höchsten Ebene zurückzukehren, müssen Sie für die Position den Wert 0 eingeben.

Aufruf

```
CALL SDF('LEVL',area,error,pos[,lst])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
LEVL	char(4)	in	Funktionsname: Schlüsselwort LEVL
area	char()	in	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
pos	integer	in	– Position der Strukturbeschreibung im aktuellen Operanden-Array oder – Position der Liste, die die Strukturbeschreibung enthält Bei pos=0 wird auf das Operanden-Array der höchsten Ebene positioniert.
lst	integer	in	nur relevant, wenn die Strukturbeschreibung Element einer Liste ist: Position der Strukturbeschreibung in der Liste

i Der direkte Zugriff auf Operanden-Array-Positionen größer als 2 ist nicht möglich.

Rückwärts kann nur auf das Operanden-Array der höchsten Ebene positioniert werden. Für die Rückwärtspositionierung auf eine Zwischenebene muss der Benutzer selbst für die Aufzeichnung des Pfades sorgen.

OPER
Operandenwert aus dem Übergabebereich lesen

Die Funktion OPER liest den Wert eines Operanden an einer bestimmten Position aus dem normierten Übergabebereich. Dem Funktionsaufruf OPER muss entweder ein Aufruf der Funktion TYPE oder der Funktion STRU vorausgegangen sein, mit denen Operandentyp und Operandenlänge festgestellt werden konnten.

Aufruf

```
CALL SDF('OPER',area,error,pos,val,lng[,lst])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
OPER	char(4)	in	Funktionsname: Schlüsselwort OPER
area	char()	in	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
pos	integer	in	Position des Operanden
val	char(lng)	inout	Adresse einer Zeichenkette, in der der Wert abgelegt werden soll
lng	integer	in	Länge der val-Zeichenkette, muss ≥ gleich dem Wert sein, der bei TYPE oder STRU für den Parameter lng zurückgegeben wurde
lst	integer	in	nur relevant, wenn der Operand Element einer Liste ist: Position des Operanden in der Liste

i Wird die Funktion OPER für einen Operanden vom Typ STRUCTURE aufgerufen, gibt die Funktion den struktureinleitenden Operandenwert zurück, dessen Länge und Typ bereits mit der Funktion STRU analysiert wurde.

READ
Anweisung lesen und analysieren

Die Funktion READ bewirkt, dass SDF

– eine Programmanweisung von SYSSTMT einliest (Für die Systemdatei SYSSTMT gilt die gleiche Zuweisung, die für die Systemdatei SYSDTA getroffen ist. Bezüglich Folgezeilen, Fortsetzungszeichen und Angabe von Bemerkungen gelten für die Anweisungseingabe von SYSSTMT die gleichen Regeln wie für die Kommandoeingabe von SYSCMD.)

– die eingelesene Anweisung analysiert und

– das Analyseergebnis an das Programm übergibt.

Voraussetzung ist, dass eine aktivierte Syntaxdatei die Definition des Programms und seiner Anweisungen enthält. Eine detaillierte Erläuterung finden Sie beim Makro RDSTMT (Seite 618ff).

Aufruf

```
CALL SDF('READ',area,error[,msg[,allow[,prefer[,default1,...]]]])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
READ	char(4)	in	Funktionsname: Schlüsselwort READ
area	char()	inout	Puffer zum Anlegen des normierten Übergabereiches (siehe INIT)
error	integer	out	Returncode
msg	V-rec@	in	Adresse eines Satzes mit variabler Satzlänge, der den Text einer Meldung enthält, die vor dem Lesen der Anweisung ausgegeben werden soll (max. 400 Zeichen lang)
allow	ptr	in	Adresse einer Struktur, die die Liste der zulässigen Anweisungen enthält [1)]
prefer	char(8)	in	nur im geführten Dialog: interner Name der Anweisung, die als nächste erwartet wird
default1, ...	char() (normierte Übergabe-bereiche)	in	Liste von bis zu 5 Umsetzbeschreibungen, mit denen Default-Werte von Operanden durch vom Programm dynamisch erzeugte Werte ersetzt werden sollen [2)]

1) Format der Liste der zulässigen Anweisungen:

N	anweisung $_1$	anweisung $_2$	.....	anweisung $_n$

N (2 Byte): Anzahl der Anweisungen in der Liste
anweisung $_x$ (8 Byte): interner Name einer zulässigen Anweisung

2) Mit Hilfe von Umsetzbeschreibungen können Default-Werte von Operanden, die mit ADD-VALUE ...,VALUE=<c-string>...(OVERWRITE-POSSIBLE=*YES) bzw. ADD-OPERAND ...,OVERWRITE-POSSIBLE=*YES in der Syntaxdatei definiert sind, durch vom Programm dynamisch erzeugte Werte ersetzt werden. Je Anweisung kann nur eine Umsetzbeschreibung angegeben werden. Die Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche Operandenwerte wie umzusetzen sind. Eine Umsetzbeschreibung kann entweder durch einen vorhergehenden READ-Aufruf oder durch einen TRNS-Aufruf, bei dem die neuen Werte eingetragen sind, erzeugt werden. Sie hat das Format eines normierten Übergabebereiches.

Der Puffer „area“ enthält Verweise auf absolute Adressen. Es ist deshalb nicht möglich, ihn in einen anderen Speicherbereich zu kopieren und dort auszuwerten.

Wird für die Parameter msg, allow und prefer ein Nullwert eingegeben (Adresse=0 oder leere Zeichenkette), so wird der entsprechende Wert ignoriert.

Ignoriert werden:

– ein bei prefer angegebener Anweisungsname, der mit einem Leerzeichen „␣“ beginnt,

– ein bei msg angegebener Satz mit der Länge 0,

– eine bei allow angegebene Liste mit 0 Elementen.

Der Makro RDSTMT wird mit folgenden Default-Werten aufgerufen:

```
PROT=*YES
BUFFER=*NO
INVAR=*NO
SPIN=*NO
ERRSTMT=*STEP
CALLID=*NO
CCSNAME=*NO
```

SEMA
Semantikfehlerdialog anstoßen

Die Funktion SEMA bewirkt, dass SDF mit dem Benutzer einen Dialog führt, in dem Sie Semantikfehler in einer Anweisung korrigieren können. SDF hat die Anweisung unmittelbar zuvor analysiert und syntaktisch richtig an das Programm übergeben. Detaillierte Informationen finden Sie beim Makro CORSTMT (Seite 613ff).

Aufruf

```
CALL SDF('SEMA',area,error,msg)
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
SEMA	char(4)	in	Funktionsname: Schlüsselwort SEMA
area	char()	inout	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
msg	V-rec@	in	Adresse eines Satzes mit variabler Satzlänge, der den Text einer Meldung enthält, die während des Fehlerdialoges ausgegeben werden soll (max. 400 Zeichen lang)

Der Makro CORSTMT wird mit folgenden Default-Werten aufgerufen:

```
DEFAULT=*NO
INVAR=*NO
CALLID=*NO
CCSNAM=*NO
```

STAT
Informationen über Syntaxdateien ausgeben

Die Funktion STAT gibt Informationen über aktivierte Syntaxdateien und über die Festlegungen aus, die für die Kommando/Anweisungseingabe und -verarbeitung gelten.
Die Beschreibung des Übergabebereiches der Informationen finden Sie beim Makro CMDSTA (Seite 431ff).

Aufruf

```
CALL SDF('STAT',area,error[,lng])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
STAT	char(4)	in	Funktionsname: Schlüsselwort STAT
area	char()	out	Übergabebereich, in dem die Informationen von SDF abgelegt werden. Muss auf Halbwortgrenze ausgerichtet sein.
error	integer	out	Returncode
lng	integer	in	Länge des Übergabebereiches: – bei Angabe von lng wird die Länge des Übergabebereiches auf lng festgelegt und es wird FORM=LONG ausgegeben (siehe CMDSTA) – wird lng nicht angegeben, so muss der Übergabebereich 530 Byte lang sein und es wird FORM=SHORT ausgegeben (siehe CMDSTA)

STMT
Anweisungsname aus dem Übergabebereich lesen

Die Funktion STMT liest den internen Namen einer Anweisung, die vorher mit der Funktion READ gelesen wurde, aus dem normierten Übergabebereich. Zusätzlich können Sie die Anzahl der Operanden dieser Anweisung abfragen.

Aufruf

```
CALL SDF('STMT',area,error,stmt[,num])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
STMT	char(4)	in	Funktionsname: Schlüsselwort STMT
area	char()	in	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
stmt	char(8)	out	Adresse eines 8 Byte langen Feldes, das den internen Anweisungsnamen aufnehmen soll
num	integer	out	Anzahl der Operanden der gelesenen Anweisung

STRU
Struktureinleitenden Wert auf Datentyp und Länge analysieren

Die Funktion STRU analysiert einen struktureinleitenden Operandenwert auf seinen Typ und seine Länge. Dem Funktionsaufruf STRU muss ein Aufruf der Funktion TYPE vorangegangen sein, der den Operandentyp Struktur (X'13') als Ergebnis hatte. Ein Operandenwert ist nur dann struktureinleitend, wenn er in der SDF-A-Anweisung ADD-VALUE mit STRUCTURE=YES(FORM=NORMAL) definiert wurde.

Aufruf

```
CALL SDF('STRU',area,error,pos,typ,lng[,lst])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
STRU	char(4)	in	Funktionsname: Schlüsselwort STRU
area	char()	in	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
pos	integer	in	Position des struktureinleitenden Wertes
typ	integer	out	Datentyp des Operandenwertes (Typbeschreibung siehe Seite 378ff)
lng	integer	out	Länge des Operandenwertes
lst	integer	in	nur relevant, wenn der Operandenwert Element einer Liste ist: Position der Struktur in der Liste

TRNS
Anweisung analysieren

Die Funktion TRNS analysiert eine Anweisung im Text-Format und übersetzt sie in das Format für den normierten Übergabebereich. Das Analyseergebnis wird im normierten Übergabebereich abgelegt. Detaillierte Informationen finden Sie beim Makro TRSTMT (Seite 626ff).

Mit der Funktion TRNS können Sie u.a. auch Umsetzbeschreibungen für die Funktion READ erzeugen.

Aufruf

```
CALL SDF('TRNS',area,error,stmt)
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
TRNS	char(4)	in	Funktionsname: Schlüsselwort TRNS
area	char()	inout	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
stmt	V-rec@	in	Adresse eines Satzes mit variabler Satzlänge, der die zu analysierende Anweisung im Textformat enthält

Der Makro TRSTMT wird mit folgenden Default-Werten aufgerufen:

```
STMT=*ALL
DIALOG=*ERROR
MESSAGE=*NO
PROT=*NO
INVAR=*NO
DEFAULT=*NO
ERROR=*NO
CALLID=*NO
EXECUTE=*NO
PROCMOD=*ANY
CCSNAME=*NO
```

TYPE
Operanden auf Datentyp und Länge analysieren

Die Funktion TYPE analysiert einen Operanden an einer bestimmten Position auf seinen Typ und seine Länge. Die Position entspricht der Position, die mit der SDF-A-Anweisung ADD-OPERAND ...,RESULT-OPERAND-NAME=*POS(...) festgelegt wurde.

Aufruf

```
CALL SDF('TYPE',area,error,pos,typ,lng[,lst])
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
TYPE	char(4)	in	Funktionsname: Schlüsselwort TYPE
area	char()	in	Puffer, in dem der normierte Übergabebereich abgelegt wurde (siehe INIT)
error	integer	out	Returncode
pos	integer	in	Operandenposition (wie bei ADD-OPERAND ..., RESULT-OPERAND-NAME=*POS(...))
typ	integer	out	Typ des Operanden (Typbeschreibung siehe Seite 378ff)
lng	integer	out	Länge des Operanden
lst	integer	in	nur relevant, wenn der Operand Element einer Liste ist: Position des Operanden in der Liste

WRRC
Kommando-Returncodes setzen

Mit der Funktion WRRC kann das Benutzerprogramm eigene Werte als „Kommando-Returncode“ für das Programm sichern, die der Kommandoprozessor nach Programmbeendigung an Stelle des offiziellen Kommando-Returncodes zur Verfügung stellt. Dieser Kommando-Returncode kann dann von anderen Systemfunktionen (z.B. SDF-P) weiterverwendet werden. Die Funktion stützt sich auf den Makro CMDRC (Seite 408ff).

Aufruf

```
CALL SDF('WRRC',area,error)
```

Beschreibung der Parameterliste

Parameter	Datentyp (Länge)	Ein-/ Ausgabe	Bedeutung
WRRC	char(4)	in	Funktionsname: Schlüsselwort WRRC
area	char(9)	in	Bereich, in dem der 9 Byte lange Kommando-Returncode angegeben wird. Muss auf Halbwortgrenze ausgerichtet sein.
error	integer	out	Returncode der Funktion

6.5.2.3 Beispiele

Die folgenden Beispiele demonstrieren das Prinzip der SDF-Funktionsaufrufe in den Programmiersprachen COBOL und FORTRAN.

Verwendung der COBOL-Schnittstelle (Programmausschnitt)

```
      .
      .
      DATA DIVISION.
      WORKING-STORAGE SECTION.
     *work variable
      01 SDF-DATA
          02 SDF-INIT PIC X(4) VALUE "INIT".
          02 SDF-READ PIC X(4) VALUE "READ".
          02 SDF-STMT PIC X(4) VALUE "STMT".
          02 SDF-TYPE PIC X(4) VALUE "TYPE".
          02 SDF-VAL  PIC X(4) VALUE "OPER".
          02 SDF-TYP  PIC 9(6) COMP.
          02 SDF-LNG  PIC 9(6) COMP.
```

```
     02 SDF-POS  PIC 9(6) COMP.
     02 SDF-LST  PIC 9(6) COMP.
     02 SDF-ERR  PIC S9(6) COMP.
     02 SDF-STMT PIC X(8).
     02 SDF-PROG PIC X(8).
 77 BUF PIC X(500).
 77 MAX PIC 9(6) COMP VALUE 500.
 01 VAL.
     02 FILLER PIC X OCCURS 1 TO 50 DEPENDING ON SDF-LNG.
 .
 .
*
 PROCEDURE DIVISION.
*
*INIT
*
    MOVE "TEST" TO SDF-PROG.
    CALL "SDF" USING SDF-INIT, BUF, SDF-ERR, MAX, SDF-PROG.
*
*READ
*
    CALL "SDF" USING SDF-READ, BUF, SDF-ERR.
*
*STMT
*
    CALL "SDF" USING SDF-STMT, BUF, SDF-ERR, SDF-STMT.
*
*TYPE
*
    CALL "SDF" USING SDF-TYPE, BUF, SDF-ERR, SDF-POS,
                     SDF-TYP, SDF-LNG.
*
*VAL
*
    CALL "SDF" USING SDF-VAL, BUF, SDF-ERR, SDF-POS, VAL,
                     SDF-LNG.
.
.
```

Das Programm muss mit der Bibliothek SYSLIB.SDF.04x (z.B. 045 für SDF V4.5) gebunden werden. In dieser Bibliothek ist unter anderem auch ein COBOL-Copy-Element mit der Beschreibung der SDF-Deklarationen verfügbar (Typ-S-Element SDFCOPY).

Verwendung der FORTRAN-Schnittstelle (Programmausschnitt)

```
.
.
C*****Work variable
      CHARACTER*4 SDF$INIT/'INIT'/
      CHARACTER*4 SDF$READ/'READ'/
      CHARACTER*4 SDF$STMT/'STMT'/
      CHARACTER*4 SDF$VAL/'OPER'/
      CHARACTER*4 SDF$TYPE/'TYPE'/
      INTEGER SDF$LNG,SDF$POS,SDF$LST,SDF$TYP,SDF$ERR
      CHARACTER*8 SDF$STMT,SDF$PROG
      CHARACTER*500 AREA,VAL*50
      INTEGER MTYP(2),MLNG(2)

      *
      *INIT
      *
          CALL SDF(SDF$INIT,AREA,SDF$ERR,500,'TEST␣␣␣␣')
      *
      *READ
      *
          CALL SDF(SDF$READ,AREA,SDF$ERR)
      *
      *STMT
      *
          CALL SDF(SDF$STMT,AREA,SDF$ERR,SDF$STMT)
      *
      *TYPE
      *
          CALL SDF(SDF$TYPE,AREA,SDF$ERR,SDF$POS,SDF$TYP,SDF$LNG)
      *
      *VAL
      *
          CALL SDF(SDF$VAL,AREA,SDF$ERR,SDF$POS,VAL,SDF$LNG)
      .
      .
```

Das Programm muss mit der Bibliothek SYSLIB.SDF.04x (z.B. 045 für SDF V4.5) gebunden werden. In dieser Bibliothek ist unter anderem auch ein FORTRAN-Include-Element mit der Beschreibung der SDF-Deklarationen verfügbar (Typ-S-Element SDFINCL).

6.5.3 C-Schnittstelle

Für die Programmiersprache C gelten andere Konventionen für Übergabeparameter an Funktionen als für COBOL und FORTRAN. SDF bietet deshalb neben der allgemeinen HLL-Schnittstelle auch eine C-Schnittstelle an, die jedoch die gleiche Funktionalität hat wie die allgemeine HLL-Schnittstelle. Funktionen, bei denen optionale Parameter vorkommen, wurden in mehrere C-Funktionen aufgespaltet.

Die C-Funktionen bieten die Funktionalität der „alten“ Makroaufrufe RDSTMT, CORSTMT und TRSTMT, die den normierten Übergabebereich im alten Format verwenden (siehe Abschnitt „Änderungen der SDF-Programmschnittstelle“ auf Seite 605ff).

Soll die erweiterte Funktionalität der entsprechenden „neuen“ Makroaufrufe CMDRST, CMDCST und CMDTST (z.B. Statement-Returncode) für den normierten Übergabebereich im neuen Format genutzt werden, muss die Assembler-Schnittstelle verwendet werden (siehe Seite 385ff).

6.5.3.1 Beschreibung der C-Funktionen

Die Beschreibung der C-Funktionen für SDF finden Sie in alphabetischer Reihenfolge im Nachschlageteil (Seite 510ff). Die Funktionen, die auf Grund optionaler Parameter auf mehrere Funktionen aufgespaltet wurden, sind unter einem Funktionsnamen beschrieben. Z.B. sind die Funktionen sdfrd, sdfrdmsg, sdfrdall, sdfrdpre und sdfrddef alle unter der Überschrift sdfrd beschrieben und als Format 1 bis 5 gekennzeichnet.

Ergebnis der Funktionen

Jede C-Funktion liefert eine Integerzahl als Ergebnis:

Ergebnis = 0:	Fehlerfreie Ausführung
Ergebnis > 0:	Fehler beim zugehörigen Makroaufruf. Der Fehlercode wurde aus Register 15 übernommen. Die Werte des Fehlercodes entsprechen den Makro-Fehlercodes.
Ergebnis < 0:	Fehler beim Aufruf der Funktion. Folgende Werte können auftreten:

-1 : Funktion unbekannt
-2 : zu wenig Operanden
-3 : der letzte Operand in der Struktur ist erreicht
-4 : der letzte Operand in der Liste ist erreicht
-5 : die Position ist negativ
-6 : der Operand ist nicht vom Typ LIST
-7 : der Operand ist nicht vom Typ STRUCTURE

Hinweise

Zum Aufruf einer C-Funktion für SDF benötigen Sie in Ihrem Programm folgende Include-Elemente aus der Bibliothek SYSLIB.SDF.04x (z.B. 045 für SDF V4.5):

`sdfc.h` (enthält Konstanten- und Typdefinitionen)

`sdfcext.h` (enthält die extern-Deklaration der Funktionen)

Beachten Sie bitte, dass einige Zeichenketten auf Halbwort- oder Wortgrenze ausgerichtet sein müssen. Da Variablen vom Datentyp char jedoch nur auf Bytegrenze ausgerichtet sind, müssen diese Variablen mit der Funktion malloc() zusätzlich ausgerichtet werden.

Die Variablen für interne Anweisungsnamen müssen 8 Zeichen lang und mit „\0“ abgeschlossen sein. Dafür kann der vordefinierte Typ STR8 aus dem Include-Element sdfc.h verwendet werden.

Ein Beispiel zur Verwendung der C-Funktionen finden Sie ab Seite 526.

6.5.3.2 Übersicht über die C-Funktionen

sdfcbit	Korrekturbit setzen
sdfcmd	Systemkommando ausführen
sdfcor	Semantikfehlerdialog anstoßen
sdfinit	Pufferinitialisierung
sdflev	Auf ein Operanden-Array positionieren
sdfrc	Kommando-Returncode setzen
sdfrd	Anweisung lesen und analysieren
sdfsta	Informationen über Syntaxdateien ausgeben
sdfstmt	Anweisungsname aus dem Übergabebereich lesen
sdfstv	Struktureinleitenden Wert auf Datentyp und Länge analysieren
sdftr	Anweisung analysieren
sdftyp	Operanden auf Datentyp und Länge analysieren
sdfval	Operandenwert aus dem Übergabebereich lesen

sdfcbit
Korrekturbit setzen

Die Funktion sdfcbit setzt für einen bestimmten Operanden ein Korrekturbit. Das gesetzte Korrekturbit bewirkt, dass dieser Operand im Fehlerdialog zu einer Anweisung unterstrichen dargestellt wird, falls er fehlerhaft war.

Format

```
int sdfcbit (char *area, int pos);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
int pos	Position des Operanden, dessen Korrekturbit gesetzt werden soll

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfcmd
Systemkommando ausführen

Die Funktion sdfcmd ermöglicht den Aufruf eines Kommandos, ohne den Programmodus zu verlassen. Die Funktion sdfcmd stützt sich auf den Makro CMD, der im Handbuch „Makroaufrufe an den Ablaufteil“ [8] beschrieben ist.

Format

```
int sdfcmd (char *area, int sysout, int dialog, int list, char *cmdrc, char *buff);
```

Beschreibung der Parameter

char *area	Zeiger auf eine Zeichenkette, die das auszuführende Kommando enthält. Die Zeichenkette wird intern von SDF in einen Satz mit variabler Satzlänge umgewandelt.
int sysout	gibt an, ob das Protokoll nach SYSOUT und/oder in den Protokollpuffer ausgegeben wird: 0 : SYSOUT = NO (nur SYSOUT) 1 : SYSOUT = YES (SYSOUT und Puffer)
int dialog	gibt an, ob ein Fehlerdialog beim Erkennen von Syntaxfehlern geführt werden soll: 0 : DIALOG = NO 1 : DIALOG = YES
int list	gibt an, ob die Zeichenkette bei area mehrere durch Semikolon getrennte Kommandos enthält: 0 : LIST = NO (nur ein Kommando in der Zeichenkette) 1 : LIST = YES (mehrere Kommandos in der Zeichenkette)
char *cmdrc	Zeiger auf eine Zeichenkette (9 Byte), in die der Kommandoprozessor den Kommando-Returncode ablegen soll. cmdrc muss auf Halbwortgrenze ausgerichtet sein.
char *buff	Zeiger auf eine Zeichenkette, in die das Protokoll ausgegeben wird (BUFMOD=SHORT, siehe CMD-Makro).

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

Es wird eine Parameterliste Version 3 (VER=3, PARMOD=31 - siehe CMD-Makro im Handbuch „Makroaufrufe an den Ablaufteil“ [8]) generiert.

sdfcor
Semantikfehlerdialog anstoßen

Die Funktion sdfcor bewirkt, dass SDF mit dem Benutzer einen Dialog führt, in dem er Semantikfehler in einer Anweisung korrigieren kann. SDF hat die Anweisung unmittelbar zuvor syntaktisch analysiert, keinen Syntaxfehler festgestellt und deshalb an das Programm übergeben. Detaillierte Informationen finden Sie beim Makro CORSTMT (Seite 613ff).

Format

```
int sdfcor (char *area, char *msg);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
char *msg	Zeiger auf eine Meldung, die während des Fehlerdialoges ausgegeben werden soll. Die Zeichenkette wird intern von SDF in einen Satz mit variabler Satzlänge umgewandelt und darf maximal 400 Zeichen lang sein. Bei Angabe eines NULL-Zeigers wird keine Meldung angezeigt.

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfinit
Pufferinitialisierung

Mit der Funktion sdfinit wird der Puffer initialisiert, in dem der normierte Übergabebereich abgelegt wird. Der interne Name des Programmes muss dabei so angegeben werden, wie er in der Syntaxdatei definiert ist.

Die Funktion sdfinit muss einmalig vor allen anderen SDF-C-Funktionen aufgerufen werden, die den normierten Übergabebereich verwenden.

Format

```
int sdfinit (char *area, int lng, STR8 progname);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer zum Anlegen des normierten Übergabebereiches, muss auf Wortgrenze ausgerichtet sein
int lng	Länge des Puffers für den normierten Übergabebereich
STR8 progname	interner Programmname (wie in der Syntaxdatei definiert), 8 Zeichen lang, mit Null-Zeichen („\0“) abgeschlossen

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdflev
Auf ein Operanden-Array positionieren

Mit der Funktion sdflev können Sie auf ein Operanden-Array positionieren, das zu einer Strukturbeschreibung gehört (siehe Seite 371ff). Alle Funktionen, die auf Daten im normierten Übergabebereich zugreifen, beziehen sich immer auf das zurzeit aktuelle Operanden-Array.

Nach dem Aufruf der Funktion sdfinit ist zunächst auf das Operanden-Array der höchsten Ebene positioniert, d.h auf das Operanden-Array, dessen Operanden mit der Anweisung ADD-OPERAND ...,RESULT-OPERAND-LEVEL=1 in der Syntaxdatei definiert wurden.

Die Funktion sdflev bezieht sich immer auf das zurzeit aktuelle Operanden-Array. Das kann auch das Operanden-Array einer Strukturbeschreibung sein, wenn die Funktion sdflev zuvor schon einmal aufgerufen wurde.

Die Strukturbeschreibung kann auch Element einer Liste sein (die Überprüfung des Operanden mit sdftyp lieferte den Operandentyp „Liste“). In diesem Fall müssen Sie sowohl die Position der Liste im aktuellen Operanden-Array als auch die Position der Strukturbeschreibung in dieser Liste angeben.
Ist die Strukturbeschreibung nicht Element einer Liste, so müssen Sie nur die Position der Strukturbeschreibung im Operanden-Array angeben.

Um zum Operanden-Array der höchsten Ebene zurückzukehren, müssen Sie für die Position den Wert 0 eingeben.

Da die Funktion einen optionalen Parameter hat, gibt es 2 verschiedene Formate.

Format 1

```
int sdflev (char *area, int pos);
```

Format 2

```
int sdflevls (char *area, int pos, int lst);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
int pos	Position der Strukturbeschreibung im aktuellen Operanden-Array oder Position der Liste, die die Strukturbeschreibung enthält. Bei pos=0 wird auf das Operanden-Array der höchsten Ebene positioniert.
int lst	ist nur relevant, wenn die Strukturbeschreibung Element einer Liste ist; gibt die Position der Strukturbeschreibung in der Liste an.

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfrc
Kommando-Returncodes setzen

Mit der Funktion sdfrc kann das Benutzerprogramm eigene Werte als „Kommando-Returncode“ für das Programm sichern, die der Kommandoprozessor nach Programmbeendigung an Stelle des offiziellen Kommando-Returncodes zur Verfügung stellt. Dieser Kommando-Returncode wird von anderen Systemfunktionen (z.B. SDF-P) ausgewertet (siehe auch Handbuch „Kommandos, Band 1-5“ [4]). Die Funktion sdfrc stützt sich auf den Makro CMDRC (Seite 408ff).

Format

```
int sdfrc (char *area);
```

Beschreibung der Parameter

char *area	Zeiger auf eine Zeichenkette, die den Kommando-Returncode enthält, muss auf Halbwortgrenze ausgerichtet sein. Dafür sollte der vordefinierte Struktur-Typ SDF_COMMAND_RC verwendet werden (siehe Beispiel Seite 526ff).

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfrd
Anweisung lesen und analysieren

Die Funktion sdfrd bewirkt, dass SDF

– eine Programmanweisung von SYSSTMT einliest (Für die Systemdatei SYSSTMT gilt die gleiche Zuweisung, die für die Systemdatei SYSDTA getroffen ist. Bezüglich Folgezeilen, Fortsetzungszeichen, Protokollierung und Angabe von Bemerkungen gelten für die Anweisungseingabe von SYSSTMT die gleichen Regeln wie für die Kommandoeingabe von SYSCMD.)

– die eingelesene Anweisung analysiert und

– das Analyseergebnis an das Programm übergibt.

Voraussetzung ist, dass eine aktivierte Syntaxdatei die Definition des Programms und seiner Anweisungen enthält. Eine detaillierte Erläuterung finden Sie beim Makro RDSTMT (Seite 618ff).

Da die Funktion einige optionale Parameter hat, gibt es 5 verschiedene Formate.

Format 1

```
int sdfrd (char *area);
```

Format 2

```
int sdfrdmsg (char *area, char *msg);
```

Format 3

```
int sdfrdall (char *area, char *msg, char *allow);
```

Format 4

```
int sdfrdpre (char *area, char *msg, char *allow, STR8 prefer);
```

Format 5

```
int sdfrddef  (char *area, char *msg, char *allow, STR8 prefer, char *def1,
               char *def2, char *def3, char *def4, char *def5);
```

Beschreibung der Parameter

char *area — Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)

char *msg — Zeiger auf eine Meldung, die vor dem Lesen der Anweisung ausgegeben werden soll. Die Zeichenkette wird intern von SDF in einen Satz mit variabler Satzlänge umgewandelt und darf maximal 400 Zeichen lang sein. Bei Angabe eines NULL-Zeigers wird keine Meldung angezeigt.

char *allow — Zeiger auf die Liste der zulässigen Anweisungen (muss auf Wortgrenze ausgerichtet sein)

Format der Liste der zulässigen Anweisungen:

N	anweisung $_1$	anweisung $_2$	. . . . .	anweisung $_n$

N	(2 Byte):	Anzahl der Anweisungen in der Liste
anweisung$_x$	(8 Byte):	interner Name einer zulässigen Anweisung

STR8 prefer — nur im geführten Dialog: interner Name der Anweisung, die als nächste erwartet wird, 8 Zeichen lang, mit Null-Zeichen („\0“) abgeschlossen

char *def1,*def2,*def3,*def4,*def5
Zeiger auf maximal 5 Umsetzbeschreibungen, mit denen Default-Werte von Operanden durch vom Programm dynamisch erzeugte Werte ersetzt werden. Für alle 5 Zeiger muss ein Wert angegeben werden. Nicht benötigte Zeiger sind als NULL-Zeiger anzugeben. Ab dem ersten gefundenen NULL-Zeiger werden alle weiteren Zeiger nicht mehr berücksichtigt.

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

Lesen mit Umsetzen von Default-Werten

Mit Hilfe von Umsetzbeschreibungen können Default-Werte von Operanden, die mit ADD-VALUE ...,VALUE=<c-string>...(OVERWRITE-POSSIBLE=*YES) bzw. ADD-OPERAND ..., OVERWRITE-POSSIBLE=*YES in der Syntaxdatei definiert sind, durch vom Programm dynamisch erzeugte Werte ersetzt werden. Je Anweisung kann nur eine Umsetzbeschreibung angegeben werden. Die Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche Operandenwerte wie umzusetzen sind. Eine Umsetzbeschreibung kann entweder durch einen vorhergehenden sdfrd-Aufruf oder durch einen sdftr-Aufruf, bei dem die neuen Werte eingetragen sind, erzeugt werden. Sie hat das Format eines normierten Übergabebereiches.

sdfsta
Informationen über Syntaxdateien ausgeben

Die Funktion sdfsta gibt Informationen über aktivierte Syntaxdateien und über die Festlegungen aus, die für die Eingabe und Verarbeitung von Kommandos und Anweisungen gelten. Die Beschreibung des Übergabebereiches der Informationen finden Sie beim Makro CMDSTA (Seite 431ff).

Da die Funktion einen optionalen Parameter hat, gibt es 2 verschiedene Formate.

Bei Aufruf von Format 1 wird immer die Kurzform ausgegeben:
aktivierte Syntaxdateien, jedoch ohne Subsystem-Syntaxdateien, sowie die aktuellen Optionen für die Eingabe und die Verarbeitung von Kommandos und Anweisungen. Für area sollte der vordefinierte Struktur-Typ SDF_STATUS_SHORT verwendet werden (siehe auch Beispiel Seite 526ff).

Bei Aufruf von Format 2 wird die Langform ausgegeben:
aktivierte Syntaxdateien (einschließlich der Subsystem-Syntaxdateien) sowie die aktuellen Optionen für die Eingabe und die Verarbeitung von Kommandos und Anweisungen. Die Menge der ausgegebenen Informationen über Subsystem-Syntaxdateien hängt davon ab, wieviel Platz im dem Übergabebereich zur Verfügung steht. Passten nicht alle Subsystem-Informationen in den Übergabebereich, so wird ein entsprechender Fehlercode übergeben.

Format 1

```
int sdfsta (char *area);
```

Format 2

```
int sdfstal (char *area, int lng);
```

Beschreibung der Parameter

char *area	Zeiger auf einen Übergabebereich, in dem die Informationen von SDF abgelegt werden, muss auf Halbwortgrenze ausgerichtet sein
int lng	Länge des Übergabebereiches für FORM=LONG (siehe Makro CMDSTA). Die Länge muss größer als SDF_STATUS_SIZE_SHORT sein (SDF_STATUS_SIZE_SHORT=530)

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfstmt
Anweisungsname aus dem Übergabebereich lesen

Die Funktion sdfstmt liest den internen Namen einer Anweisung, die vorher mit der Funktion sdfrd gelesen wurde, aus dem normierten Übergabebereich. Zusätzlich können Sie die Anzahl der Operanden der Anweisung abfragen.

Da die Funktion einen optionalen Parameter hat, gibt es 2 verschiedene Formate dafür.

Format 1

```
int sdfstmt (char *area, STR8 stmt);
```

Format 2

```
int sdfstmtn (char *area, STR8 stmt, int *num);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
STR8 stmt	Zeichenkette, die den internen Namen der Anweisung aufnehmen soll, 8 Zeichen lang, mit Null-Zeichen („\0“) abgeschlossen
int *num	Zeiger auf die Anzahl der Operanden der gelesenen Anweisung

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfstv
Struktureinleitenden Wert auf Datentyp und Länge analysieren

Die Funktion sdfstv analysiert einen struktureinleitenden Operandenwert auf seinen Typ und seine Länge. Dem Funktionsaufruf sdfstv muss ein Aufruf der Funktion sdftyp vorangegangen sein, der den Operandentyp Struktur (19) als Ergebnis brachte. Ein Operandenwert ist nur dann struktureinleitend, wenn er in der SDF-A-Anweisung ADD-VALUE mit STRUCTURE=YES(FORM=NORMAL) definiert wurde.

Da die Funktion einen optionalen Parameter hat, gibt es 2 verschiedene Formate.

Format 1

```
int sdfstv (char *area, int pos, int *typ, int *lng);
```

Format 2

```
int sdfstvls (char *area, int pos, int *typ, int *lng, int lst);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
int pos	Position des struktureinleitenden Operandenwertes
int *typ	Zeiger auf einen Wert, der den Typ des Operandenwertes enthält (Typbeschreibung siehe Seite 378ff)
int *lng	Zeiger auf die Länge des Operandenwertes
int lst	ist nur relevant, wenn der Operandenwert Element einer Liste ist; gibt die Position der Struktur in der Liste an

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdftr
Anweisung analysieren

Die Funktion sdftr analysiert eine Anweisung im Text-Format und übersetzt sie in das Format für den normierten Übergabebereich. Das Analyseergebnis wird im normierten Übergabebereich abgelegt. Detaillierte Informationen finden Sie beim Makro TRSTMT (Seite 626ff).

Mit der Funktion sdftr können Sie u.a. auch Umsetzbeschreibungen für die Funktion sdfrddef erzeugen.

Da die Funktion einen optionalen Parameter hat, gibt es 2 verschiedene Formate.

Format 1

```
int sdftr (char *area, char *stmt);
```

Format 2

```
int sdftrall (char *area, char *stmt, char *allow);
```

Beschreibung der Parameter

char *area — Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)

char *stmt — Zeiger auf eine Zeichenkette, die die zu analysierende Anweisung im Textformat enthält. Die Zeichenkette wird intern von SDF in einen Satz mit variabler Satzlänge umgewandelt.

char *allow — Zeiger auf die Liste der Anweisungen, die als Eingabe zulässig sind (muss auf Wortgrenze ausgerichtet sein).
Format der Liste der zulässigen Anweisungen:

N	anweisung $_1$	anweisung $_2$	. . . . .	anweisung $_n$

N	(2 Byte):	Anzahl der Anweisungen in der Liste
anweisung$_x$	(8 Byte):	interner Name einer zulässigen Anweisung

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdftyp
Operanden auf Datentyp und Länge analysieren

Die Funktion sdftyp analysiert einen Operanden an einer bestimmten Position auf seinen Typ und seine Länge. Die Position entspricht der Position, die mit der SDF-A-Anweisung ADD-OPERAND ...,RESULT-OPERAND-NAME=*POS(...) festgelegt wurde.

Format 1

```
int sdftyp (char *area, int pos, int *typ, int *lng);
```

Format 2

```
int sdftypls (char *area, int pos, int *typ, int *lng, int lst);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
int pos	Position des Operanden (wie bei ADD-OPERAND ..., RESULT-OPERAND-NAME=*POS(...))
int *typ	Zeiger auf einen Wert, der den Typ des Operanden enthält (Typbeschreibung siehe Seite 378ff)
int *lng	Zeiger auf die Länge des Operanden
int lst	ist nur relevant, wenn der Operand Element einer Liste ist; gibt die Position des Operanden in der Liste an

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

sdfval
Operandenwert aus dem Übergabebereich lesen

Die Funktion sdfval liest den Wert eines Operanden an einer bestimmten Position aus dem normierten Übergabebereich. Dem Funktionsaufruf sdfval muss entweder ein Aufruf der Funktion sdftyp oder der Funktion sdfstv vorausgegangen sein, mit denen Operandentyp und Operandenlänge festgestellt werden konnten.

Da die Funktion einen optionalen Parameter hat, gibt es 2 verschiedene Formate.

Format 1

```
int sdfval (char *area, int pos, char *val, int lng);
```

Format 2

```
int sdfvalls (char *area, int pos, char *val, int lng, int lst);
```

Beschreibung der Parameter

char *area	Zeiger auf den Puffer, in dem der normierte Übergabebereich angelegt wurde (siehe sdfinit)
int pos	Position des Operanden im aktuellen Operanden-Array
char *val	Zeiger auf eine Zeichenkette, in der der Wert abgelegt werden soll
int lng	Länge der val-Zeichenkette, muss größer gleich dem Wert sein, der bei sdfstv oder sdftyp für den Parameter lng zurückgegeben wurde
int lst	ist nur relevant, wenn der Operand Element einer Liste ist; gibt die Position des Operanden in der Liste an

Ergebnis

Die Funktion liefert eine Integerzahl als Ergebnis (siehe Seite 509).

6.5.3.3 Beispiel zur Verwendung der C-Funktionen

Im Abschnitt „Beispiel: Programm zum Kopieren von Dateien“ auf Seite 106ff wurde ein Assembler-Programm KOP erstellt, mit dem SAM- und ISAM-Dateien kopiert werden können. In diesem Beispiel soll nun gezeigt werden, wie die gleiche Funktionalität mit einem C-Programm erreicht werden kann. Das Programm KOPC hat neben den SDF-Standardanweisungen die folgende Anweisung:

COPY-FILE

FROM-FILE = <filename 1..54>

,TO-FILE = <filename 1..54 without-gen-vers>(...)

<filename 1..54 without-gen-vers>(...)

ACCESS-METHOD = ***SAME** / ***ISAM**(...) / ***SAM**

***ISAM**(...)

KEY-LENGTH = ***STD** / <integer 1..50>

,RECORD-SIZE = ***SAME** / ***VARIABLE** / <integer 1..2048>

,PASSWORD = ***NONE** / <c-string 1..4> / ***SECRET-PROMPT**

Programm in der Benutzersyntaxdatei definieren

Das Programm KOPC wird in der Syntaxdatei SDF.KOP.SYNTAX definiert. Dazu werden die gleichen Anweisungen (Ausnahme: 4 und 5) wie ab Seite 107 an SDF-A gegeben. Die Anweisungen 4 und 5 werden wie folgt eingeben:

```
//add-prog kopc ————————————————————————————————————————————— (4)
//add-stmt name=copy-file,prog=kopc,intern-name=copyfi,stmt-vers=1 ——— (5)
```

Programm erstellen

Im Folgenden ist das Programm KOPC ausschnittsweise wiedergegeben.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*******************************/
/*   SDF includes              */
/*******************************/
#include "sdfc.h"
#include "sdfcext.h"

/*******************************/
/*   Constants & types         */
/*******************************/
#define OUTPUTL 200
typedef char STR8[8+1];         /* string of 8 chars, null terminated */
typedef char FILENAME[54+1];
typedef char STR4[4+1];

main () {

/*******************************/
/*   data                      */
/*******************************/

/* SDF interface data */

char *output;
SDF_STATUS_SHORT *status;
char *usfname;
STR8 name;
int sdf_err;
int sdf_op_l,sdf_op_t;
char val_char[16];

/* program data */

FILENAME from_file,to_file;
int  access;
int  keyl;
int  recs_var;
int  recs;
int  passwd_given;
STR4 passwd;
int nocorr;
```

```
/* SDF interface data  allocation */

if ( (output=malloc(OUTPUTL)) == NULL ) exit(2);
if ( (status=malloc(SDF_STATUS_SIZE_SHORT)) == NULL )exit(2);

/*******************************/
/*   check syntax file         */
/*******************************/

sdf_err = sdfsta((char *)status);
if (sdf_err) exit(3);
usfname = strchr((status)->sdf_user_sf_name,'.') + 1;
if (strncmp( usfname,
         "SDF.KOP.SYNTAX",14))
                     exit(3);

/*******************************/
/*   initialization            */
/*******************************/

sdf_err = sdfinit (output,OUTPUTL, "KOP     ");
if (sdf_err) exit(4);

/*******************************/
/*   read first statement      */
/*******************************/

sdf_err = sdfrd (output);
switch (sdf_err) {
    case SDF_END:
    case SDF_EOF: return(0);
    case SDF_OK:  break;
    default:      exit(5);
    }

/*******************************/
/*   analyse loop              */
/*******************************/

do {
    /* * * * * * * * * * * * * * * * * */
    /*   transfer area analysis      */
    /* * * * * * * * * * * * * * * * * */

    /* statement name */

    sdf_err = sdfstmt (output,name);
```

```
if (sdf_err) exit(6);
if (strncmp (name,"COPYFI  ",8)) exit(7);

/* operand from file */

sdf_err = sdftyp (output,1,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(8);
sdf_err = sdfval (output,1,from_file,sdf_op_l);
if (sdf_err) exit(9);
from_file[sdf_op_l] = '\0';

/* operand password */

passwd_given = 0;
sdf_err = sdftyp (output,6,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(10);
if (sdf_op_t == SDF_CSTR) {
    passwd_given = 1;
    sdf_err = sdfval (output,6,passwd,sdf_op_l);
    if (sdf_err) exit(11);
    passwd[sdf_op_l] = '\0';
}

/* operand to file */

sdf_err = sdftyp (output,2,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(13);
sdf_err = sdfval (output,2,to_file,sdf_op_l);
if (sdf_err) exit(14);
to_file[sdf_op_l] = '\0';

/* sub operand access-method */

sdf_err = sdftyp (output,3,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(15);
sdf_err = sdfval (output,3,val_char,sdf_op_l);
if (sdf_err) exit(16);
if (sdf_op_l == 3)
    access = 1;      /* assume SAM is 1 */
else if ( val_char[0] == 'I' )
    access = 2;      /* assume ISAM is 2 */
else
    access = 0;      /* assume SAME is 0 */


/* sub sub operand key length */

if (access == 2) {   /* only for ACCESS-METHOD=ISAM */
```

```
    sdf_err = sdftyp (output,4,&sdf_op_t,&sdf_op_l);
    if (sdf_err) exit(17);
    switch (sdf_op_t) {
        case (SDF_NOTY): return(17); /* operand not occupied */
        case (SDF_INTG):
          sdf_err = sdfval (output,4,(char *)&keyl,sdf_op_l);
          if (sdf_err) exit(18);
          break;
        default: keyl = 8;
                 break;
    }
}

/* sub operand record-size */

recs = 0;
recs_var = 0;
sdf_err = sdftyp (output,5,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(21);
if (sdf_op_t == SDF_INTG) {
    sdf_err = sdfval (output,5,(char *)&recs,sdf_op_l);
    if (sdf_err) exit(22);
}
else {
    sdf_err = sdfval (output,5,val_char,sdf_op_l);
    val_char[sdf_op_l] = '\0';
    if (sdf_err) exit(23);
    if (!strcmp (val_char,"SAME"))
        recs = 0;    /* assume 0 is same */
    else if (!strcmp (val_char,"VARIABLE"))
        recs_var = 1;
    else exit(24);
}

/* * * * * * * * * * * * * * * * * */
/*      actual processing          */
/* * * * * * * * * * * * * * * * * */

/* semantic validation of input parameter and
   correction if necessary */
/* example : open of input file
   call DMS
   IF error
      sdf_err = sdfcbit (output,1);
      sdf_err = sdfcor (output,"Input file can not be opened");
      if (!sdf_err) continue; restart analysis       */

 /* example : assume password 'AAAA' wrong : prompt for
```

```
         correction */
        nocorr = 0;
        if (!strcmp (passwd,"AAAA")) {
           sdf_err = sdfcbit (output,6);
           sdf_err = sdfcor (output,"Invalid password");
           if (!sdf_err) continue; /* restart analysis */
           else nocorr = 1;
        }


      if (!nocorr) {
        /* copy processing .... */
        printf ("from file %s\n",from_file);
        printf ("to file %s\n",to_file);
        printf ("access %d \n",access);
        if (access == 2) {
           printf ("key 1 %d\n",key1);
        }
        if (!recs_var)
           printf ("recs %d \n",recs);
        else
           printf ("rec var\n");
        if (passwd_given)
           printf ("password %s\n",passwd);
      }

     /* * * * * * * * * * * * * * * * * */
     /*      read next statement      */
     /* * * * * * * * * * * * * * * * * */

     sdf_err = sdfrd (output);
     switch (sdf_err) {
         case SDF_END:
         case SDF_EOF: return(0);
         case SDF_OK:  break;
         default:      exit(3);
         }
} while (1);   /* endless loop */

}
```

Programm testen

Das gezeigte Quellprogramm KOPC wird übersetzt und gebunden. Nun soll es getestet werden:

```
/set-logon-parameters sdfusr,...  ————————————————————————————————  (1)
 .
 .
/modify-sdf-options syntax-file=*add(sdf.kop.syntax)  ————————————————  (2)
/start-exe *lib-elem(lib=kop.lib,elem=kopc)  ————————————————————————  (3)
%  BLS0524 LLM 'KOPC', VERSION ' ' OF '2001-10-10 14:00:22' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2001. ALL RIGHTS
RESERVED
%// ...  ————————————————————————————————————————————————————————————  (4)
 .
 .
```

(1) Unter der Benutzerkennung SDFUSR wird ein Prozess gestartet.

(2) Die Benutzersyntaxdatei SDF.KOP.SYNTAX, in der die Anweisungen für das Programm KOPC definiert sind, wird aktiviert.

(3) Das Programm KOPC wird gestartet. Das hier verwendete Kommando START-EXECUTABLE-PROGRAM steht ab BLSSERV V2.3 zur Verfügung (ggf. ist das Kommando START-PROGRAM mit RUN-MODE=*ADVANCED zu verwenden).

(4) Das Programm KOPC erwartet die Eingabe einer Anweisung. Zur Verfügung stehen die SDF-Standardanweisungen und die Anweisung COPY-FILE. Zum Testen des Programmes KOPC kann wie beim Programm KOP ab Arbeitsschritt 8 (Seite 122ff) verfahren werden.

7 SDF-SIM

Bei der Definition von Kommandos und Anweisungen mit SDF-A ergibt sich zwangsläufig die Notwendigkeit, die Syntax dieser Kommandos und Anweisungen zu testen. Mit SDF-SIM (System Dialog Facility-Simulator) steht Ihnen ein Werkzeug zur Verfügung, mit dem Sie den Syntaxtest in einer von Ihnen gewählten Testumgebung effizient und risikolos durchführen können.

SDF-SIM ist durch eine einfache Bedienoberfläche leicht einsetzbar und bietet eine Vielzahl unterstützender Hilfsmittel, wie z.B.:

- flexible Testumgebungen,
- Simulation von Prozedur- oder Batchmodus mit realistischem Spin-Off-Verhalten,
- Eingabe über Jobvariablen,
- Berücksichtigung von Privilegien,
- differenzierte Ausgabemöglichkeiten des Simulationsergebnisses.

Simulationsumgebung festlegen

Mit SDF-SIM kann eine Umgebung simuliert werden, in der die Syntax von Kommandos oder Anweisungen analysiert wird. Diese Umgebung ist unabhängig von der BS2000-Systemumgebung. Ein Benutzer ohne besondere Privilegien kann in einem wesentlich größeren Umfang auf die Umgebung Einfluss nehmen, als es ihm sonst erlaubt ist (z.B. Zuweisen von System- und Gruppensyntaxdatei).

Folgende SDF-Optionen können für die Umgebung definiert werden:

- Batch- oder Dialog-Modus
- Prozedur-Modus
- geführter Dialog oder ungeführter Dialog
- Fortsetzungsmodus (CONTINUATION)
- Protokollierung (INPUT-FORM, ACCEPTED-FORM, INVARIANT-FORM)
- Privilegien, die die Task besitzt.

Kommando- und Anweisungssyntax testen

Die Syntax der Kommandos bzw. Anweisungen, die in einer bestimmten SDF-Syntaxdateihierarchie (System-, Gruppen-, Benutzersyntaxdatei) beschrieben sind, wird auf TU-Ebene getestet. Der Benutzer kann damit prüfen, wie SDF ein eingegebenes Kommando oder eine Anweisung syntaktisch aufbereitet.

Die Prozeduren, Programme oder Kommando-Server sind für den Syntaxtest nicht erforderlich, da SDF-SIM nur auf die Syntaxdefinitionen in der festgelegten Syntaxdateihierarchie zugreift. Dies ist auch in Bild 16 verdeutlicht.

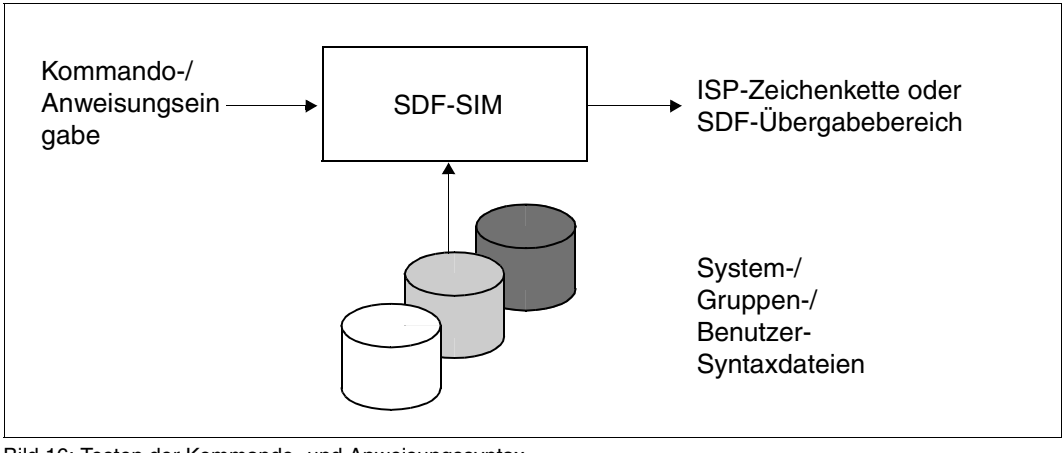


Bild 16: Testen der Kommando- und Anweisungssyntax

Informationen über das Kommando oder die Anweisung ausgeben

Für jedes simulierte Kommando bzw. für jede simulierte Anweisung werden folgende Informationen ausgegeben:

- Protokoll des Kommandos bzw. der Anweisung
- Schnittstellentyp für die Kommandos (ASS/ISL/SPL/PROCEDURE)
- Name des Einsprungpunktes in die kommandoausführende Routine (nur für TPR-Kommandos)
- Ausgabeformat:
  - Zeichenkette, wenn ein TPR-Kommando auf ein altes ISP-Kommando abgebildet wird oder das Kommando durch eine Prozedur implementiert ist
  - SDF-Übergabebereich bei neuen SDF-Kommandos bzw. -Anweisungen
  - bei Kommandos, die durch eine Prozedur implementiert sind, wird der zugehörige Prozeduraufruf ausgegeben. Damit kann geprüft werden, wie die Prozedurparameter an die Prozedur übergeben werden (siehe Beispiel Seite 572).

7.1 Arbeiten mit SDF-SIM

Komponenten für die Installation von SDF-SIM

Die Anweisungen, die an SDF-SIM gegeben werden können, sind in der Syntaxdatei SYSSDF.SDF-SIM.045 definiert.

Folgende Dateien werden mit SDF-SIM ausgeliefert:

SYSLNK.SDF-SIM.045	Bibliothek, die das Bindelademodul SDF-SIM enthält.
SYSSDF.SDF-SIM.045	Syntaxdatei, die die SDF-SIM-Anweisungen und das Kommando START-SDF-SIM enthält.
SYSMES.SDF-SIM.045	Meldungsdatei für SDF-SIM.

Koexistenz verschiedener SDF-SIM-Versionen

Mit dem Kommando START-SDF-SIM (siehe Seite 536) starten Sie die SDF-SIM-Version, die der Systembetreuung Ihrer Anlage festgelegt hat. Das wird im Normalfall die aktuellste SDF-SIM-Version sein. Sie haben jedoch auch die Möglichkeit, eine ältere Version von SDF-SIM aufzurufen. Dazu sind folgende Schritte notwendig:

1. Erzeugen Sie sich mit SDF-A eine Benutzersyntaxdatei. Kopieren Sie mit der COPY-Anweisung alle SDF-SIM-Anweisungen der gewünschten SDF-SIM-Version hinein. Die SDF-SIM-Anweisungen sind in der Systemsyntaxdatei enthalten, die mit der gewünschten SDF-SIM-Version ausgeliefert wird (SYSSDF.SDF-SIM.<version>).
2. Aktivieren Sie die erzeugte Benutzersyntaxdatei mit dem Kommando MODIFY-SDF-OPTIONS (siehe Handbuch „Kommandos, Band 3“ [4]).
3. Rufen Sie die gewünschte SDF-SIM-Version mit folgendem Kommando auf:

   ```
   /START-EXE *LIB-ELEM(LIBRARY=$.SYSLNK.SDF-SIM.vvv,ELEMENT=SDF-SIM) 1)
   ```

   vvv ist die gewünschte Versionsnummer (z.B. 041 für SDF-SIM V4.1)

   bzw.

   ```
   /START-SDF-SIM VERSION=<version>
   ```

   <version> ist die gewünschte Version (z.B. 4.1 für SDF-SIM V4.1)

[1] Das hier verwendete Kommando START-EXECUTABLE-PROGRAM steht ab BLSSERV V2.3 zur Verfügung (ggf. ist das Kommando START-PROGRAM zu verwenden).

Starten von SDF-SIM

SDF-SIM wird mit folgendem Kommando gestartet:

START-SDF-SIM	Kurzname: **SDF-SIM**
VERSION = ***STD** / <product-version>	
,MONJV = ***NONE** / <filename 1..54 without-gen-vers>	
,CPU-LIMIT = ***JOB-RE**ST / <integer 1..32767 *seconds*>	

VERSION =
Sind mehrere Versionen von SDF-SIM mit IMON installiert, so kann der Benutzer die Version auswählen, mit der er arbeiten möchte. Bei Angabe der Versionsbezeichnung mit Hochkommas kann der Buchstabe C vorangestellt werden (C-STRING-Syntax).
Falls das Produkt nicht mit IMON installiert wurde oder die angegebene Version nicht existiert, gilt VERSION=*STD (siehe auch „Koexistenz verschiedener SDF-SIM-Versionen“ auf Seite 535).

VERSION = *STD
Aufruf der SDF-SIM-Version mit der höchsten Versionsnummer.

VERSION = <product-version>
Angabe der SDF-SIM-Version im Format mm.n[a[so]] (vgl. „product-version“ auf Seite 15).

MONJV =
Angabe einer Monitor-Jobvariablen zur Überwachung des SDF-SIM-Laufs.

MONJV = *NONE
Es wird keine Monitor-Jobvariable verwendet.

MONJV = <filename 1..54 without-gen-vers>
Name der zu verwendenden Jobvariablen.

CPU-LIMIT =
Maximale CPU-Zeit in Sekunden, die das Programm beim Ablauf verbrauchen darf.

CPU-LIMIT = *JOB-REST
Es soll die verbleibende CPU-Zeit für die Aufgabe verwendet werden.

CPU-LIMIT = <integer 1..32767 *seconds*>
Es soll nur die angegebene Zeit verwendet werden.

Anschließend können Sie die Anweisungen zur Simulationsvorbereitung eingeben (DEFINE-TEST-OBJECT, DEFINE-ENVIRONMENT).

Die Simulation wird nach der Eingabe der Anweisung START-SIMULATION gestartet. SDF-SIM zeigt zuerst die aktuelle Syntaxdateihierarchie an. Danach fordert es mit mit einem Stern (*) zur Eingabe des zu simulierenden Kommandos bzw. der zu simulierenden Anweisung auf.

Beenden von SDF-SIM

Während der Simulation („*“-Prompting) kann SDF-SIM durch Eingabe der Zeichenfolge „/*“ beendet werden.
Während der Simulationsvorbereitung („%//“-Prompting) steht dafür die SDF-Standardanweisung END zur Verfügung.

Ablaufschema

Das folgende Beispiel zeigt den prinzipiellen Ablauf einer Simulation.

```
/START-SDF-SIM
%// ...                  }
%// ...                  }  ——— Anweisungen zur Simulationsvorbereitung
%// ...                  }
%//START-SIMULATION       ——— Starten der Simulation
* ...                    }
* ...                    }  ——— Kommandos/Anweisungen, die simuliert werden
* ...                    }      sollen
*/*
```

Die Anweisungen zur Simulationsvorbereitung können auch mehrfach angegeben werden. Dann gelten die Festlegungen, die mit der letzten dementsprechenden Anweisung gegeben wurden.

7.1.1 Kommandoumgebung

SDF-SIM prüft die Richtigkeit der Syntax des Kommandos, das in der aktiven Syntaxdateihierarchie definiert ist. Die Syntaxprüfung wird innerhalb einer Umgebung durchgeführt, die mit der Anweisung DEFINE-ENVIRONMENT definiert wurde.

Simulationsvorbereitung

Wird beim Operanden PARAMETER-FILE eine Parameterdatei angegeben (*STD oder <filename>), dann werden die Namen der System- und der Gruppensyntaxdatei, die der PROFILE-ID von TSOS (SYS-TSOS) zugeordnet sind, aus der Parameterdatei gelesen und für die Simulation aktiviert. Bei fehlerhafter Parameterdatei oder ungültigen Syntaxdateien werden Systemsyntaxdatei und TSOS-Gruppensyntaxdatei mit den Standardnamen verwendet.
Wird der Name der System-, Gruppen- oder Benutzersyntaxdatei eingegeben und es tritt während der Aktivierung der Syntaxdatei ein Fehler auf, so wird eine Fehlermeldung ausgegeben. Bei fehlerhafter System- oder Gruppensyntaxdatei werden die Syntaxdateien mit den Standardnamen aktiviert. Bei fehlerhafter Benutzersyntaxdatei wird keine Benutzersyntaxdatei aktiviert. Es muss mindestens eine System- oder Gruppensyntaxdatei aktiviert sein, sonst kann auf Grund fehlender Globalinformation keine Simulation stattfinden. Wird als Systemsyntaxdatei *STD eingegeben, dann wird nur die aktuelle Basis-Systemsyntaxdatei aktiviert. Wird als Systemsyntaxdatei *CURRENT eingegeben, so werden die aktuelle Basis-Systemsyntaxdatei und die aktuellen Subsystem-Syntaxdateien aktiviert.

Simulation

Nach dem Start der Simulation kann das zu simulierende Kommando im geführten oder ungeführten Dialog eingegeben werden. Dies ist auch abhängig von der Festlegung im Kommando MODIFY-SDF-OPTIONS (siehe Beispiel Seite 562). Sowohl im simulierten Prozedur- als auch im Batchmodus (Anweisung DEFINE-ENVIRONMENT PROC-MODE=*YES oder TASK-TYPE=*BATCH) muss ein Kommando mit vorausgehendem „/“ eingegeben werden. Die Kommandos können auch in Kleinbuchstaben eingegeben werden (siehe Beispiel Seite 567).

Ausgaben

Nach der Syntaxanalyse wird ein Protokoll ausgegeben. Es enthält den Namen der Einsprungstelle, den Typ der Programmiersprachen-Schnittstelle, die generierte Zeichenkette (ISP-Format) bzw. den normierten Übergabebereich für Kommandos, die mit der SDF-A-Anweisung ADD-CMD ...,IMPLEMENTOR=*TPR(.., CMD-INTERFACE=*NEW/*TRANSFER-AREA...) definiert wurden.

Jobvariablen ersetzen

Das Ersetzen von Jobvariablen wird in SDF-SIM genauso durchgeführt wie außerhalb der Simulation (siehe Beispiel Seite 566). Das Ersetzen von Prozedurparametern ist jedoch nicht möglich.

Spezielle SDF-Kommandos

Die nachfolgenden SDF-Kommandos können verwendet werden, wenn sie in mindestens einer der Syntaxdateien aus der Syntaxdateihierarchie enthalten sind. Diese Kommandos werden nicht simuliert, sondern von SDF-SIM wirklich ausgeführt.

Diese Kommandos sind:

– MODIFY-SDF-OPTIONS:
Benutzersyntaxdatei aktivieren bzw. deaktivieren und SDF-Optionen für die Simulation ändern (siehe Seite 557ff).

– SHOW-SDF-OPTIONS:
Aktive Syntaxdateien und eingestellte SDF-Optionen für die Simulation anzeigen

– SHOW-INPUT-HISTORY:
Liste der letzten Eingaben anzeigen

– RESTORE-SDF-INPUT:
Eingaben wieder anzeigen

– SHOW-INPUT-DEFAULTS:
Taskspezifische Default-Werte anzeigen

– RESET-INPUT-DEFAULTS:
Taskspezifische Default-Werte zurücksetzen

– WRITE-TEXT:
Eingebenen Text anzeigen

– SHOW-SYNTAX-VERSIONS:
Namen und Versionen der Syntaxdateien einzelner Produkte anzeigen, die in gerade aktiven Gruppen- und Systemsyntaxdateien enthalten sind

– MODIFY-SDF-PARAMETERS:
Parameterdatei für SDF ändern oder erzeugen
Um dieses Kommando ausführen zu können, muss das Privileg *TSOS oder *ALL in der Anweisung DEFINE-ENVIRONMENT angegeben sein. Außerhalb der Simulation ist dieses Kommando der Systembetreuung vorbehalten.

– SHOW-SDF-PARAMETERS:
  Inhalt der SDF-Parameterdatei anzeigen
  Um dieses Kommando ausführen zu können, muss das Privileg *TSOS oder *ALL in der Anweisung DEFINE-ENVIRONMENT angegeben sein. Außerhalb der Simulation ist dieses Kommando der Systembetreuung vorbehalten.

Ausgabe von SDF-SIM

Folgende Informationen gibt SDF-SIM aus:

- das Protokoll, das von SDF in der Form geliefert wird, wie es beim Operanden LOGGING des Kommandos MODIFY-SDF-OPTIONS festgelegt wurde.
- den Namen des Einsprungpunktes der TPR-Routine (nicht, wenn das Kommando mit ADD-CMD ...,IMPLEMENTOR=*PROCEDURE (siehe SDF-A) definiert wurde).
- den Schnittstellentyp: ASS/SPL/ISL/PROCEDURE
- die Ausgabe, die dem Kommandoserver geliefert wird. Das ist entweder:
  – eine Zeichenkette, wenn ein Kommando auf ein altes ISP-Kommando abgebildet wird bzw. das Kommando durch eine Prozedur implementiert ist oder
  – ein normierter Übergabebereich, der maximal 2042 Byte lang sein kann.

Bei Kommandos, die durch eine Prozedur implementiert sind, wird der generierte Prozedur-Aufruf ausgegeben. Für Kommandos, die das neue Format des nomierten Übergabebereiches (ab SDF V4.1) verwenden, wird auch die Version des Übergabebereiches und die Version des Kommandos angezeigt.

Beispiel

Kommando, das durch eine Prozedur implementiert ist (definiert mit ADD-CMD ..., IMPL=*PROCEDURE..., siehe SDF-A, Seite 135ff)

```
/x-write 'date 2001-12-31'
(IN)       /x-write 'date 2001-12-31'

ENTRY : CLICOLD
INTERFACE : ISL
STRUCTURE-FORM
GENERATED CALL COMMAND :
/CALL-PROCEDURE $USER1.PROC.TEXT.1,(TEXT='date 2001-12-31')
```

7.1.2 Anweisungsumgebung

SDF-SIM prüft die Richtigkeit der Syntax von Anweisungen, die in der aktiven Syntaxdateihierarchie definiert sind. In der Simulationsvorbereitung müssen dazu der interne Name des Programms, zu dem die zu simulierenden Anweisungen gehören, sowie das bei den Anweisungen verwendete Format des normierten Übergabebereiches angegeben werden (siehe DEFINE-TEST-OBJECT TEST-MODE=*STMT(PROGRAM-NAME=..., LAYOUT= *OLD/*NEW).

Die Anweisungen werden in der Umgebung simuliert, die mit DEFINE-ENVIRONMENT definiert wurde. Mit der Anweisung MODIFY-SDF-OPTIONS kann die Simulation wie in der Kommandoumgebung beeinflusst werden (geführter oder ungeführter Dialog, Protokollierungsform,...).

Nach der Syntaxanalyse wird ein Protokoll und der normierte Übergabebereich ausgegeben, wenn der Benutzer dessen Ausgabe anfordert.

Ausgabe von SDF-SIM

Folgende Informationen gibt SDF-SIM aus:

- das Protokoll der Anweisung in der Form, die mit dem Operanden LOGGING von MODIFY-SDF-OPTIONS festgelegt ist
- den normierten Übergabebereich. Ein ausführliches Beispiel dazu finden Sie auf Seite 573.

  Der normierte Übergabebereich wird im folgenden Schema gezeigt, wenn:

  – in der Anweisung DEFINE-ENVIRONMENT der Operand DISPLAY=*LONG gesetzt ist oder

  – die Meldung
  ```
  %  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
  ```
  mit „LONG“ beantwortet wird.

  Bei DISPLAY=*SHORT wird nur der erste Teil (Inhalt des Übergabebereiches in sedezimaler Form) ausgegeben. Die Strukturbeschreibung (STRUCTURED DESCRIPTION) wird nur bei DISPLAY=*LONG ausgegeben. Der interne Name der Anweisung wird angezeigt. Die Strukturbeschreibung der Anweisung kann maximal 15000 Byte lang sein.

Beispiel

```
 .
 .
%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
*long

     *** TRANSFER AREA ***
     ~~~~~~~~~~~~~~~~~~~~
                 x x  x x x x
     (0FD540) 07FAC3D9 C6C9D3C5 40400000 00000000 00000020 ...
     (0FD5C0) 00000000 00000000 00000000 00000000 00008014 ...
     (0FD600) 00000000 00000000 00000000 00000000 ...
     (0FD6E0) 00000000 00000000 80160005 ...
      .
      .
     *** STRUCTURED DESCRIPTION ***
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     INTERNAL NAME OF THE CMD/STMT : xxxxxxxx
     VERSION OF THE TRANSFER AREA  : x   } (nur beim neuen Format des Über-
     VERSION OF THE CMD/STMT       : x   } gabebereiches (ab SDF V4.1))
     MAXIMUM NUMBER OF OPERANDS    : xx
     OP( 1) :
      - TYPE   : KEYWORD
      - LENGTH :   xx
      - VALUE  : xxxxxx

     OP( 2) :
      - TYPE   : STRUCTURE
            VALUE INTRODUCING THE STRUCTURE:
           - TYPE   : xxxx
           - LENGTH :   xx
           - VALUE  : xxxxxxx
           MAXIMUM NUMBER OF OPERANDS   :    xx
           OP( 1) :
            - TYPE   : xxxxxx
            - LENGTH :  ....
            - VALUE  :  ...
           OP( 2) :     ....
       .
       .

     OP( 3) :
      - TYPE   : LIST
      - VAL(1) : - TYPE   : xxxx
                 - LENGTH : xxxxxxx
                 - VALUE  : xxxxxxx
      - VAL(2) : .........
```

```
OP( 4) :
 - TYPE   : OR_LIST
 - LENGTH : xxxx
 - VALUE  : xxxxx
 .
 .
```

Bei TYPE sind folgende Werte möglich:

STRUCTURE	Struktur
LIST	Liste
OR_LIST	OR-Liste
X_STR	X-Zeichenkette (x-string)
ALPHA_NAME	alphanumerischer Name (alphanum-name)
NAME	Name (name)
STRUCT_NAME	strukturierter Name (structured-name)
COM_R	Kommandorest (command-rest)
C_STR	C-Zeichenkette (c-string)
TEXT	Text (text)
INT	Ganzzahl (integer)
KEYW:	Schlüsselwort (KEYWORD)
KEYW_NUM:	reserviert für internen Gebrauch
F_FILENAME	Dateiname (filename)
P_FILENAME	teilqualifizierter Dateiname (partial-filename)
TIME	Uhrzeit (time)
DATE	Datum (date)
CAT_ID	Katalogkennung (cat-id)
REAL	real
LABEL	Marke (label)
STAR-ALPHA	star-alphanumeric
COMPOSED_N	zusammengesetzter Name (composed-name)
INPUT_TEXT	Eingabe-Text (input-text)
VSN	VSN (vsn)
X_TEXT	X-Text (x-text)
FIXED	Festpunktzahl (fixed)
DEVICE	Gerät (device)
POSIX_PATHN	POSIX-Pfadname (posix-pathname)
POSIX_FILEN	POSIX-Dateiname (posix-filename)
PRODUCT_V	Produkt-Version (product-version)

Spezielle SDF-Anweisungen

Die nachfolgenden Anweisungen sind SDF-Standardanweisungen und deshalb für alle Programme definiert, wenn die aktivierte Syntaxdatei richtig erzeugt wurde. Sie werden durch SDF-SIM nicht simuliert, sondern wirklich ausgeführt. Ab SDF V4.1 sind die SDF-Standardanweisungen in der Syntaxdatei von SDF definiert. Sie sind nicht mehr der SDF-U-Syntaxdatei zugeordnet und können auch nicht in die Syntaxdatei des Anwenderprogrammes kopiert werden. Deshalb muss bei Verwendung der Standardanweisungen in der Simulation eine SDF-Parameterdatei angegeben werden, die als Basis-Systemsyntaxdatei die Syntaxdatei von SDF enthält (siehe Beispiel auf Seite 555).

Von SDF-SIM unterstützte Standardanweisungen sind:

– MODIFY-SDF-OPTIONS:
  Benutzersyntaxdatei aktivieren bzw. deaktivieren und SDF-Optionen für die Simulation ändern
– SHOW-SDF-OPTIONS:
  Aktive Syntaxdateien und eingestellte SDF-Optionen für die Simulation anzeigen
– SHOW-INPUT-HISTORY:
  Liste der letzten Eingaben anzeigen
– RESTORE-SDF-INPUT:
  Eingaben wiederanzeigen
– SHOW-INPUT-DEFAULTS:
  Taskspezifische Default-Werte anzeigen
– SHOW-STMT:
  Syntax einer Anweisung anzeigen
– RESET-INPUT-DEFAULTS:
  Taskspezifische Default-Werte zurücksetzen
– WRITE-TEXT:
  Eingebenen Text anzeigen
– REMARK:
  Anweisungsfolgen kommentieren
– STEP:
  Spin-off unterbrechen
– END:
  Simuliertes Programm beenden

Die SDF-Standardanweisungen EXECUTE-SYSTEM-CMD und HOLD-PROGRAM (ab SDF V4.0 und BS2000/OSD-BC V2.0) werden in SDF-SIM nicht unterstützt.

Hinweise zu Programmen

– Wenn der Programmname, der in der Anweisung DEFINE-TEST-OBJECT angegeben wurde, nicht in der aktivierten Syntaxdateihierarchie definiert ist, wird nach START-SIMULATION die Syntaxdateihierarchie angezeigt, die Meldung
```
% SDS0006 PROGRAM '...' NOT DEFINED IN ACTIVATED SYNTAX FILES
```
ausgegeben und SDF-SIM beendet.

– Mit der SDF-A-Anweisung ADD-PROGRAM ... COMMENT-LINE=... kann Programmen eine Kommentarzeile zugeordnet werden. Wurde dabei COMMENT-LINE=*STD angegeben, dann erscheint in der Simulation keine Kommentarzeile, da SDF-SIM auf diese internen Programm-Informationen nicht zugreifen kann.

7.1.3 SDF-SIM-Lauf innerhalb einer Prozedur oder eines Batchauftrages

SDF-SIM kann wie andere Dienstprogramme auch in Prozeduren und Batchaufträgen aufgerufen werden. Das bedeutet jedoch nicht implizit, dass Kommandos bzw. Anweisungen automatisch im simulierten Prozedur-/Batchmodus getestet werden. Für den Test im simulierten Prozedur- oder Batchmodus muss die Anweisung DEFINE-ENVIRONMENT mit den Operanden PROC-MODE=*YES oder TASK-TYPE=*BATCH eingegeben werden.

Wird für den Operanden DISPLAY der Anweisung DEFINE-ENVIRONMENT der Wert *QUESTION explizit oder implizit (als Standardwert) angegeben, so wirkt diese Angabe in Prozeduren und Batchaufträgen wie DISPLAY=*NO.

Bei Kommandos bzw. Anweisungen, die im Prozedur-/Batchmodus innerhalb einer Prozedur oder eines Batchauftrages getestet werden, muss vor „/“ bzw. „//“ unbedingt ein „*“ angegeben werden, da die Eingaben sonst als reale Kommandos/Anweisungen interpretiert und tatsächlich abgearbeitet werden.

7.2 Anweisungen von SDF-SIM

Mit SDF-SIM stehen dem Benutzer folgende Anweisungen zur Verfügung:

– DEFINE-TEST-OBJECT definiert den Typ des Testobjektes:
Testobjekt kann entweder ein Kommando oder eine Anweisung sein.

– DEFINE-ENVIRONMENT definiert eine Testumgebung.

– START-SIMULATION beendet die Simulationsvorbereitung und startet die Simulation.

Die SDF-Standardanweisungen (MODIFY-SDF-OPTIONS, REMARK, RESTORE-SDF-INPUT, SHOW-INPUT-HISTORY, SHOW-SDF-OPTIONS, SHOW-STMT, STEP, SHOW-INPUT-DEFAULTS, RESET-INPUT-DEFAULTS, WRITE-TEXT) können Sie in SDF-SIM ebenfalls angeben. Die Standardanweisungen werden zu jedem Zeitpunkt des SDF-SIM-Laufs wirklich ausgeführt, also auch nach START-SIMULATION (während „*“-Prompting) und beeinflussen so den Simulationsablauf. So können Sie z.B. in der Simulation Anweisungen im geführten Dialog eingeben oder einen Fehlerdialog führen.

Die Beschreibung der Metasyntax zu den SDF-SIM-Anweisungen finden Sie im Abschnitt „Verwendete Metasprache“ auf Seite 7. Die SDF-Standardanweisungen sind im Handbuch „Einführung in die Dialogschnittstelle SDF“ [1] beschrieben.

DEFINE-ENVIRONMENT
Testumgebung festlegen

Die Anweisung DEFINE-ENVIRONMENT definiert die Testumgebung für die Simulation.

Folgende Optionen können für die Umgebung definiert werden:

– Syntaxdateihierarchie (Operand PARAMETER-FILE)
– Prozedur-Modus (Operand PROC-MODE)
– Batch- oder Dialog-Modus (Operand TASK-TYPE)
– Verhalten im Fehlerfall (Operand SPINOFF)
– Anzeigen des normierten Übergabebereiches (Operand DISPLAY)
– Privilegien, die die Task besitzt (Operand PRIVILEGES)
– Anzeigen der Aufruf-Information

(Teil 1 von 2)

DEFINE-**ENV**IRONMENT

PARAMETER-**FI**LE = ***STD** (...) / ***NO**(...) / <filename 1..54 without-gen-vers>(...)
  ***STD**(...)
    USER = ***STD** / ***NO** / <filename 1..54>
  ***NO**(...)
    SYSTEM = ***STD** / ***NO** / ***CUR**RENT / <filename 1..54>
    ,**GR**OUP = ***STD** / ***NO** / <filename 1..54>
    ,**USER** = ***STD** / ***NO** / <filename 1..54>
  <filename 1..54 without-gen-vers>(...)
    USER = ***STD** / ***NO** / <filename 1..54>
,**PROC-MODE** = ***NO** / ***YES**
,**TASK-TYPE** = ***DIALOG** / ***BATCH**
,**SPINOFF** = ***NO** / ***YES**
,**DISP**LAY = ***QUESTION** / ***NO** / ***SHORT** / ***LONG**

Fortsetzung ➠

(Teil 2 von 2)

,**PRIV**ILEGES = ***ALL** / list-poss(2000): ***TSOS** / ***SEC**URITY-**ADM**INISTRATION /
***USER-ADM**INISTRATION / ***HSMS-ADM**INISTRATION / ***SEC**URE-**OLTP** /
***TAPE-ADM**INISTRATION / ***SAT-FILE-MANAG**EMENT / ***NET-ADM**INISTRATION /
***FT-ADM**INISTRATION / ***FTAC-ADM**INISTRATION / ***HARDWARE-MAINT**ENANCE /
***SAT-FILE-EVAL**UATION / ***SUBSYS**TEM-**MANAG**EMENT /
***SW-MONITOR-ADM**INISTRATION / ***ACS-ADM**INISTRATION /
***VM2000-ADM**INISTRATION / ***VIRT**UAL-**MACH**INE-**ADM**INISTRATION /
***SEC**URE-**UTM** / ***PROP-ADM**INISTRATION / ***OPERA**TING / ***STD-PROC**ESSING
***POSIX-ADM**INISTRATION / ***PRINT-SERVICE-ADM**INISTRATION

,**CALL-INF**ORMATION = ***YES** / ***NO**

PARAMETER-FILE =
bestimmt die Syntaxdateihierarchie, in der die Kommandos/Anweisungen simuliert werden.

PARAMETER-FILE = *STD(...)
Die Standard-Parameterdatei von SDF wird verwendet.

USER =
bestimmt die Benutzersyntaxdatei.

USER = *STD
Die aktuelle Benutzersyntaxdatei wird verwendet.

USER = *NO
Es wird keine Benutzersyntaxdatei aktiviert.

USER = <filename 1..54>
Name der Benutzersyntaxdatei, die aktiviert werden soll.

PARAMETER-FILE = *NO
Es wird keine Parameterdatei verwendet. Der Benutzer legt die Syntaxdateihierarchie selbst fest.

SYSTEM =
bestimmt die Systemsyntaxdatei.

SYSTEM = *STD
Die aktuelle Basis-Systemsyntaxdatei wird verwendet.

SYSTEM = *NO
Es wird keine Systemsyntaxdatei verwendet.

SYSTEM = *CURRENT
Die aktuelle Basis-Systemsyntaxdatei und die aktuellen Subsystem-Syntaxdateien werden verwendet.

SYSTEM = <filename 1..54>
Name der Systemsyntaxdatei, die aktiviert werden soll.

GROUP =
bestimmt die Gruppensyntaxdatei.

GROUP = *STD
Die aktuelle Gruppensyntaxdatei wird verwendet.

GROUP = *NO
Es wird keine Gruppensyntaxdatei aktiviert.

GROUP = <filename 1..54>
Name der Gruppensyntaxdatei, die aktiviert werden soll.

USER =
bestimmt die Benutzersyntaxdatei.

USER = *STD
Die aktuelle Benutzersyntaxdatei wird verwendet.

USER = *NO
Es wird keine Benutzersyntaxdatei aktiviert.

USER = <filename 1..54>
Name der Benutzersyntaxdatei, die aktiviert werden soll.

PARAMETER-FILE = <filename 1..54 without-gen-vers>(...)
Name der Parameterdatei, die verwendet wird.

USER =
bestimmt die Benutzersyntaxdatei.

USER = *STD
Die aktuelle Benutzersyntaxdatei wird verwendet.

USER = *NO
Es wird keine Benutzersyntaxdatei aktiviert.

USER = <filename 1..54>
Name der Benutzersyntaxdatei, die aktiviert werden soll.

Wird eine Parameterdatei als Eingabe verwendet (*STD oder <filename>), dann werden die Namen der System- und der Gruppensyntaxdatei, die der PROFILE-ID von TSOS (SYS-TSOS) zugeordnet sind, aus der Parameterdatei gelesen und für die Simulation aktiviert.
Bei fehlerhafter Parameterdatei oder ungültigen Syntaxdateien werden Systemsyntaxdatei und TSOS-Gruppensyntaxdatei mit den Standardnamen verwendet und die Meldungen CMD0685 und CMD0686 ausgegeben.

Wird der Name der System-, Gruppen- oder Benutzersyntaxdatei eingegeben und es tritt während der Aktivierung der Syntaxdatei ein Fehler auf, so wird eine Fehlermeldung ausgegeben. Bei fehlerhafter System- oder Gruppensyntaxdatei werden die Syntaxdateien mit den Standardnamen aktiviert. Bei fehlerhafter Benutzersyntaxdatei wird keine Benutzersyntaxdatei aktiviert.
Es muss mindestens eine System- oder Gruppensyntaxdatei aktiviert sein, sonst kann auf Grund fehlender Globalinformation keine Simulation stattfinden.

Wird als Systemsyntaxdatei *STD eingegeben, dann wird nur die aktuelle Basis-Systemsyntaxdatei aktiviert.

PROC-MODE =
bestimmt, ob der Prozedurmodus simuliert werden soll.

PROC-MODE = *NO
Kein Prozedurmodus in der Testumgebung.

PROC-MODE = *YES
Prozedurmodus in der Testumgebung.

Im simulierten Prozedurmodus muss ein Kommando mit vorausgehendem „/“ bzw. eine Anweisung mit „//“ eingegeben werden. Das Kommando bzw. die Anweisung kann auch in Kleinbuchstaben eingegeben werden.
Nach einem Fehler kann der Spin-Off-Mechanismus aktiviert werden (siehe Operand SPINOFF, Seite 551). Um den Spin-Off zu stoppen, muss das Kommando SET-JOB-STEP bzw. die Anweisung STEP eingegeben werden.
Die Ersetzung von Prozedurparametern ist nicht möglich.
Diese Umgebung ermöglicht den Test von Kommandos/Anweisungen, die nur im Prozedurmodus erlaubt sind.

TASK-TYPE =
bestimmt die Art des Auftrags.

TASK-TYPE = *DIALOG
Dialogmodus in der Testumgebung.

TASK-TYPE = *BATCH
Batchmodus in der Testumgebung.

Im simulierten Batch-Modus muss dem Kommando ein „/“ bzw. der Anweisung „//“ vorangestellt werden. Das Kommando bzw. die Anweisung kann auch in Kleinbuchstaben eingegeben werden. Nach einem Fehler kann der Spin-Off-Mechanismus aktiviert werden (siehe Operand SPINOFF, Seite 551). Um den Spin-Off zu stoppen, muss das Kommando SET-JOB-STEP bzw. die Anweisung STEP eingegeben werden.
Diese Umgebung erlaubt das Testen von Kommandos/Anweisungen, die nur im Batch-Modus zulässig sind.

SPINOFF =
legt die Spin-Off-Behandlung bei fehlerhaften Kommandos oder Anweisungen fest. Der Operand wird nur verwendet, wenn PROC-MODE=*YES oder TASK-TYPE=*BATCH eingestellt ist.

SPINOFF = *NO
Es wird kein Spin-Off im Fehlerfall ausgelöst.

SPINOFF = *YES
Spin-Off wird im Fehlerfall durchgeführt. Um den Spin-Off zu beenden, muss das Kommando SET-JOB-STEP bzw. die Anweisung STEP abgearbeitet werden.

DISPLAY =
bestimmt, ob der Inhalt des normierten Übergabebereiches angezeigt wird. Der Operand gilt *nicht* für Kommandos, die mit ADD-CMD..., IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING...) definiert wurden (siehe SDF-A, auf Seite 141).

DISPLAY = *QUESTION
Nach der Syntaxanalyse wird für jedes Kommando/jede Anweisung folgende Frage ausgegeben:

```
%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
```

In Prozeduren und Batchaufträgen wirkt DISPLAY=*QUESTION wie DISPLAY=*NO.

DISPLAY = *NO
Der Inhalt des normierten Übergabebereiches wird nicht angezeigt.

DISPLAY = *SHORT
Der Inhalt des normierten Übergabebereiches wird im Zeichenformat und im sedezimalen Format mit einer Zeilenlänge von 80 Zeichen/Zeile ausgegeben.

DISPLAY = *LONG
Zusätzlich zur Ausgabe von DISPLAY=*SHORT wird eine Strukturbeschreibung (STRUCTURED DESCRIPTION) angezeigt. Sie enthält Typ, Länge und Wert jedes Operanden und kann maximal 15000 Byte lang sein.
Bei Anweisungen, die das ab SDF V4.1 übliche Format des normierten Übergabebereiches verwenden, muss bei DEFINE-TEST-OBJECT *STMT(...LAYOUT=*NEW) angegeben werden, um eine korrekte Ausgabe zu erhalten.

PRIVILEGES =
bestimmt die Privilegien, die die Task in der Testumgebung besitzt.

PRIVILEGES = *ALL
Die Task hat alle Privilegien.

PRIVILEGES = list-poss(2000): *TSOS / *SECURITY-ADMINISTRATION / *USER-ADMINISTRATION / *HSMS-ADMINISTRATION / *SECURE-OLTP / *TAPE-ADMINISTRATION / *SAT-FILE-MANAGEMENT / *NET-ADMINISTRATION / *FT-ADMINISTRATION / *FTAC-ADMINISTRATION / *HARDWARE-MAINTENANCE / *SAT-FILE-EVALUATION / *SUBSYSTEM-MANAGEMENT / *SW-MONITOR-ADMINISTRATION / *ACS-ADMINISTRATION / *VM2000-ADMINISTRATION / *VIRTUAL-MACHINE-ADMINISTRATION / *SECURE-UTM / *PROP-ADMINISTRATION / *OPERATING / *STD-PROCESSING / *POSIX-ADMINISTRATION / *PRINT-SERVICE-ADMINISTRATION
Die Task hat genau die Privilegien, die in dieser Liste angegeben werden.

CALL-INFORMATION = *YES / *NO
Legt fest, ob die Aufruf-Information angezeigt wird. Die Aufruf-Information enthält:
– ein Protokoll des Kommandos oder der Anweisung
– den Schnittstellentyp des Kommandos (ASS/ISL/SPL/PROCEDURE)
– den Namen des Einsprungpunktes bei TPR-Kommandos.

DEFINE-TEST-OBJECT
Testobjekt festlegen

Die Anweisung DEFINE-TEST-OBJECT legt fest, ob die Syntax von Kommandos oder Anweisungen geprüft wird.

```
DEFINE-TEST-OBJECT

TEST-MODE = *CMD / *STMT(...)
  *STMT(...)
      PROGRAM-NAME = <name 1..8>
      ,LAYOUT = *OLD / *NEW
```

TEST-MODE =
definiert die Art der Testobjekte.

TEST-MODE = *CMD
Testobjekte sind Kommandos.

TEST-MODE = *STMT(...)
Testobjekte sind Anweisungen.

PROGRAM-NAME = <name 1..8>
Interner Name des Programms, zu dem die zu simulierenden Anweisungen gehören.

LAYOUT = *OLD / *NEW
Format des normierten Übergabebereiches. Wenn Anweisungen das neue Format (ab SDF V4.1) verwenden, muss *NEW angegeben werden, um korrekte Ausgaben von SDF-SIM zu erhalten.

START-SIMULATION
Simulation starten

Die Anweisung START-SIMULATION startet die Simulation in der mit DEFINE-ENVIRONMENT definierten Umgebung für das mit DEFINE-TEST-OBJECT festgelegte Testobjekt.

SDF-SIM gibt nach dem Start der Simulation die aktuelle Syntaxdateihierarchie aus. Anschließend fordert es mit '*' zur Eingabe des zu simulierenden Kommandos bzw. der zu simulierenden Anweisung auf.

START-SIMULATION

Die Anweisung hat keine Operanden.

7.3 Beispiele zur Anwendung von SDF-SIM

7.3.1 SDF-Standardanweisungen in der Simulation zur Verfügung stellen

Ab SDF V4.1 sind die SDF-Standardanweisungen in der Syntaxdatei von SDF definiert. Sie sind nicht mehr der SDF-U-Syntaxdatei zugeordnet und können auch nicht in die Syntaxdatei des Anwenderprogrammes kopiert werden. Deshalb muss bei Verwendung der Standardanweisungen in der Simulation eine SDF-Parameterdatei angegeben werden, die als Basis-Systemsyntaxdatei die Syntaxdatei von SDF enthält.

Folgende Schritte sind zu unternehmen, um die Standardanweisungen in SDF-SIM verwenden zu können:

1. Erzeugen einer Parameterdatei, die die Syntaxdatei von SDF als Basis-Systemsyntaxdatei enthält. Wenn die Syntaxdatei des Anwenderprogrammes eine Subsystem-Syntaxdatei ist, dann muss auch sie in der SDF-Parameterdatei eingetragen sein (siehe „SDF-Parameterdatei mit Hilfe von SDF-SIM erzeugen“ auf Seite 556).
2. Starten von SDF-SIM (Kommando START-SDF-SIM)
3. Simulationsumgebung und Testobjekt festlegen,
   - wenn die Syntaxdatei des Anwenderprogrammes eine Systemsyntaxdatei ist:

     ```
     //DEFINE-ENVIRONMENT PARAMETER-FILE=MY-PARAMETER-FILE
     //DEFINE-TEST-OBJECT *STMT(PROGRAM-NAME=<internal-program-name>)
     ```

   - wenn die Syntaxdatei des Anwenderprogrammes eine Benutzersyntaxdatei ist:

     ```
     //DEFINE-ENVIRONMENT PARAMETER-FILE=MY-PARAMETER-FILE,
                          USER=<program-syntax-file>
     //DEFINE-TEST-OBJECT *STMT(PROGRAM-NAME=<name>,LAYOUT=<layout>)
     ```

4. Simulation starten (`//START-SIMULATION`).
   Danach sind die SDF-Standardanweisungen zusätzlich zu den Anweisungen des Anwenderprogrammes verfügbar.

SDF-Parameterdatei mit Hilfe von SDF-SIM erzeugen

Auch unprivilegierte Benutzer können in SDF-SIM das Kommando MODIFY-SDF-PARAMETERS verwenden und damit eine SDF-Parameterdatei für die Simulation erzeugen:

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//define-environment parameter-file=*no(system=$.syssdf.sdf.045)
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$.SYSSDF.SDF.045' ACTIVATED
%  SDS0005 'USER' SYNTAX FILE 'SDF.USER.SYNTAX' ACTIVATED
*modify-sdf-parameters scope=*next-session(parameter-file-name=my-parameter-
file),syntax-file-type=*system(name=$.syssdf.sdf.045)
(IN)       modify-sdf-parameters scope=*next-session(parameter-file-name=my-
parameter-file),syntax-file-type=*system(name=$.syssdf.sdf.045)
%  CMD0681 SYNTAX FILE '$.SYSSDF.SDF.045' INSERTED IN PARAMETER FILE 'MY-
PARAMETER-FILE'
*modify-sdf-parameters scope=*next-session(parameter-file-name=my-parameter-
file),syntax-file-type=*subsystem(name=$.syssdf.sdf-a.041,sub-name=sdf-a)
(IN)       modify-sdf-parameters scope=*next-session(parameter-file-name=my-
parameter-file),syntax-file-type=*subsystem(name=$.syssdf.sdf-a.041,sub-
name=sdf-a)
%  CMD0709 SYSTEM SYNTAX FILE '$.SYSSDF.SDF-A.041' INSERTED IN PARAMETER FILE
'MY-PARAMETER-FILE'
*/*
```

7.3.2 Verwendung von MODIFY-SDF-OPTIONS

Mit dem Kommando bzw. der Anweisung MODIFY-SDF-OPTIONS können während der Simulation ('*'-Prompting) die SDF-Optionen geändert werden. MODIFY-SDF-OPTIONS wird nicht simuliert, sondern wirklich abgearbeitet.

Folgende Optionen sind für die Simulation nützlich:

GUIDANCE = *EXPERT / *NO / *MINIMUM / *MEDIUM / *MAXIMUM

für die Art der Dialogführung beim Syntaxtest

LOGGING = *INPUT-FORM / *ACCEPTED-FORM / *INVARIANT-FORM

für die Form des Protokolls, das SDF-SIM ausgibt

PROCEDURE-DIALOGUE = *YES / *NO

für die Simulation eines Prozedurdialogs im Fehlerfall (nur gültig, wenn in der Anweisung DEFINE-ENVIRONMENT PROC-MODE=*YES eingestellt ist)

CONTINUATION = *OLD-MODE / *NEW-MODE

nur für den simulierten Prozedur- bzw. Batchmodus

Test mit interaktiven Korrekturen im simulierten Prozedurmodus

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*std(user=*no),proc-mode=*yes,display=*no
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX' ACTIVATED
%  SDS0005 'GROUP' SYNTAX FILE '$TSOS.SYS.SDF.GROUP.SYNTAX.TSOS' ACTIVATED
*/mod-sdf-options guid=*no,proc-dial=*yes ------------------------------ (1)
(IN)        /mod-sdf-options guid=*no,proc-dial=*yes
CMD:
*/sh-fi-att file-ne=aaaa
(IN)        /sh-fi-att file-ne=aaaa
%  CMD0185 OPERAND NAME 'FILE-NE' COULD NOT BE IDENTIFIED.
ENTER OPERANDS: ------------------------------------------------------- (2)
file-ne=aaaa
*file-name=aaaa
(IN)       /show-file-attributes file-name=aaaa

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTATUS AAAA,LIST=(SYSOUT)
CMD:
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. Das Kommando MODIFY-SDF-OPTIONS wird im Simulationskontext ausgeführt. Da die Simulation im Prozedurmodus abläuft (DEFINE-ENV ...,PROC-MODE=*YES), wird beim Auftreten eines Fehlers ein Fehlerdialog geführt (GUIDANCE=*NO).
2. Durch die fehlerhafte Eingabe des Operandennamens wird ein Fehlerdialog ausgelöst.

Protokollierung während der Simulation

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test-obj *cmd
%//def-env *std(user=*no),display=*no
%//start-simulation
 .
 .
*mod-sdf-opt logging=*inv ------------------------------------------------ (1)
(IN)       mod-sdf-opt logging=*inv
*sh-f-att aaa.
(IN)       SHOW-FILE-ATTRIBUTES FILE-NAME=AAA.,INFORMATION=*NAME-AND-
SPACE,SELECT=*ALL,OUTPUT=*SYSOUT,OUTPUT-OPTIONS=*PARAMETERS(SORT-LIST=
*BY-FILENAME)        ------------------------------------------------------ (2)

ENTRY : DCOFSDF
INTERFACE : ISL
STRUCTURE-FORM
*mod-fi-att aaa,bbb,wr-pass=c'111'
(IN)       MODIFY-FILE-ATTRIBUTES FILE-NAME=AAA,NEW-NAME=BBB,SUPPORT=
*UNCHANGED,PROTECTION=*PARAMETERS(PROTECTION-ATTR=*UNCHANGED,ACCESS=*BY-
PROTECTION-ATTR,USER-ACCESS=*BY-PROTECTION-ATTR,BASIC-ACL=*BY-PROTECTION-
ATTR,GUARDS=*BY-PROTECTION-ATTR,WRITE-PASSWORD=P,READ-PASSWORD=P,EXEC-
PASSWORD=P,DESTROY-BY-DELETE=*BY-PROTECTION-ATTR,AUDIT=*UNCHANGED,SPACE-
RELEASE-LOCK=*BY-PROTECTION-ATTR,EXPIRATION-DATE=*BY-PROTECTION-ATTR,FREE-
FOR-DELETION=*BY-PROT-ATTR-OR-UNCH),SAVE=*UNCHANGED,MIGRATE=*STD,CODED-
CHARACTER-SET=*UNCHANGED,DIALOG-CONTROL=*STD,OUTPUT=*NO

ENTRY : CMDFCAT
INTERFACE : ISL
STRUCTURE-FORM
 .
 .
```

1. Es soll in der ausführlichsten Form protokolliert werden (INVARIANT-FORM).
2. Die simulierten Kommandos werden in der ausführlichsten Form protokolliert. Deshalb erscheinen auch die mit Default-Werten vorbelegten Operanden.

7.3.3 Testen im temporär geführten Dialog

Im temporär geführten Dialog können Anwendungsbereiche und Kommando-/Anweisungsstrukturen (Operanden und Default-Werte) geprüft werden.

Das nachfolgende Beispiel zeigt den Test des Kommandos COPY-FILE, das in der Systemsyntaxdatei $TSOS.SYS.SDF.SYSTEM.SYNTAX enthalten ist.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*no(system=$.sys.sdf.system.syntax,group=*no,user=*no)
%//start-simulation
 .
 .
*copy-file? ──────────────────────────────────────────────────────────────── (1)
```

```
COMMAND  : COPY-FILE


------------------------------------------------------------------------------
FROM-FILE             = a
TO-FILE               = b
PROTECTION            = *STD
REPLACE-OLD-FILES     = *YES
BLOCK-CONTROL-INFO    = *KEEP-ATTRIBUTE
IGNORE-PROTECTION     = *NO
DIALOG-CONTROL        = *STD
OUTPUT                = *NO









------------------------------------------------------------------------------
NEXT = *EXECUTE
       *EXECUTE"F3" / Next-cmd / *CONTINUE / *EXIT"K1" / *EXIT-ALL"F1" /
       *TEST"F2"



LTG                                                              TAST
```

```
(IN)       /COPY-FILE FROM-FILE=A,TO-FILE=B

ENTRY : DCOCOPF
INTERFACE : ISL
STRUCTURE-FORM
%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT?short
```

```
     *** TRANSFER AREA ***
     ßßßßßßßßßßßßßßßßßßßßßß
                  C O  P F I L
     (2250E0) 0052C3D6 D7C6C9D3 40400000 00000000

     (2250F0) 00000009 800B0022 512A800B 0022512E
                                          A
     (225120) 00000000 00000000 00000001 C1000001
               B
     (225130) C200
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. Um im temporär geführten Dialog zu testen, muss das „?“ ebenso wie beim normalen Arbeiten mit SDF eingegeben werden.
2. Die Operanden des Kommandos bzw. der Anweisung erscheinen mit ihrer Vorbelegung auf dem Bildschirm. Die zu prüfenden Werte (hier: a und b) werden eingegeben.

Zum Testen mit permanent geführtem Dialog oder mit Fehlerdialog muss MODIFY-SDF-OPTIONS in der Simulation verwendet werden (GUIDANCE=*MINIMUM/ *MEDIUM/*MAXIMUM/*NO).

7.3.4 Testen mit maximaler Führungsstufe

Bei maximaler Führungsstufe können neben den Kommandos/Anweisungen, Operanden und Werten auch die Anwendungsbereiche und die Hilfetexte zu Kommandos/Anweisungen und Operanden getestet bzw. überprüft werden.

Das nachfolgende Beispiel zeigt den Test des Kommandos MODIFY-USER-SWITCHES, das in der aktuellen Systemsyntaxdatei der Task definiert ist.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*no(system=*std)
%//start-simulation
 .
 .
*mod-sdf-opt guidance=*max ──────────────────────────────────────────────── (1)
(IN)       mod-sdf-opt guidance=*max
```

```
──────────────────────────────────────────────────────────────────────────────
AVAILABLE APPLICATION DOMAINS:

  1  ACCOUNTING                   : Output of informations about the user
                                    identification and introduction of data
                                    into the accounting record
  2  ALL-COMMANDS                 : Output of all command names in alphabetic
                                    order
  3  CONSOLE-MANAGEMENT           : Control of operator console/terminal
  4  DATABASE                     : Management and administration of databases
  5  DCAM                         : Control of transaction-driven system (DCAM)
  6  DCE                          : Management of DCE (Distributed Computing
                                    Environment)
  7  DEVICE                       : Information about devices and volumes
  8  FILE                         : Management of files
  9  FILE-GENERATION-GROUP        : Management of file generation groups
──────────────────────────────────────────────────────────────────────────────
NEXT = +
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN




LTG                                                              TAST
```

 .
 .

```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 10  MULTI-CATALOG-AND-PUBSET-MGMT : Control of file accesses to local area
                                     network
 11  PROCEDURE                     : Control of the command procedures
 12  PROGRAM                       : Control of program flow
 13  SDF                           : Control of dialogue interfaces
 14  SECURITY-ADMINISTRATION       : Management of access controls and security
                                     audit trails
 15  SYSTEM-MANAGEMENT             : Dynamic control of the subsystem
                                     configuration
 16  SYSTEM-TUNING                 : Performance control and tuning of system
                                     parameters to optimize system throughput,
                                     response time, disk access and resource
                                     management
--------------------------------------------------------------------------------
NEXT = +
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F8=+   F9=REST-SDF-IN




LTG                                                                      TAST
```

```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 12  PROGRAM                       : Control of program flow
 13  SDF                           : Control of dialogue interfaces
 14  SECURITY-ADMINISTRATION       : Management of access controls and security
                                     audit trails
 15  SYSTEM-MANAGEMENT             : Dynamic control of the subsystem
                                     configuration
 16  SYSTEM-TUNING                 : Performance control and tuning of system
                                     parameters to optimize system throughput,
                                     response time, disk access and resource
                                     management
 17  USER-ADMINISTRATION           : Modification of the user identification
                                     passwords and switches
 18  UTILITIES                     : Start of utility programs
--------------------------------------------------------------------------------
NEXT = 17
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F9=REST-SDF-IN




LTG                                                                      TAST
```

```
AVAILABLE APPLICATION DOMAINS:

 12  PROGRAM                        : Control of program flow
 13  SDF                            : Control of dialogue interfaces
 14  SECURITY-ADMINISTRATION        : Management of access controls and security
                                      audit trails
 15  SYSTEM-MANAGEMENT              : Dynamic control of the subsystem
                                      configuration
 16  SYSTEM-TUNING                  : Performance control and tuning of system
                                      parameters to optimize system throughput,
                                      response time, disk access and resource
                                      management
 17  USER-ADMINISTRATION            : Modification of the user identification
                                      passwords and switches
 18  UTILITIES                      : Start of utility programs
--------------------------------------------------------------------------------
NEXT = 17
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F9=REST-SDF-IN




LTG                                                              TAST
```

```
DOMAIN   : USER-ADMINISTRATION
--------------------------------------------------------------------------------
AVAILABLE COMMANDS:

  1  MODIFY-USER-SWITCHES           : Modifies the user switch settings
  2  SHOW-USER-STATUS               : Provides information on a group of user
                                      tasks
  3  SHOW-USER-SWITCHES             : Displays the user switches which are set
                                      to 1
  4  WAIT-EVENT                     : Places a task in the wait state until a
                                      defined event occurs or until the
                                      specified time has elapsed




--------------------------------------------------------------------------------
NEXT = 1
       Number / Next-command / (Next-domain) / *DOMAIN-MENU
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F9=REST-SDF-IN   F12=*CANCEL




LTG                                                              TAST
```

```
DOMAIN   : USER-ADMINISTRATION            COMMAND: MODIFY-USER-SWITCHES


------------------------------------------------------------------------------
USER-IDENTIFICATION  = *OWN
                       *OWN or name_1..8
                       Specifies the user ID whose user switches are to be
                       changed
ON                   = *UNCHANGED
                       *UNCHANGED or  -list-possible (32)-: integer_0..31
                       Specifies the user switches that are to be set to 1
OFF                  = *UNCHANGED
                       *UNCHANGED or  -list-possible (32)-: integer_0..31
                       Specifies the user switches that are to be set to 0



------------------------------------------------------------------------------
NEXT = +
       Next-command / (Next-domain) / *CONTINUE / *DOMAIN-MENU / *TEST
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL



LTG                                                                TAST
```

.
.

1. Das Kommando MODIFY-SDF-OPTIONS wird im Kontext der Simulation ausgeführt. Die möglichen Anwendungsbereiche und ihre Hilfetexte erscheinen auf dem Bildschirm. Der Benutzer kann einen Anwendungsbereich auswählen.
2. Der Benutzer wählt mit Eingabe der Nummer 17 den Anwendungsbereich USER-ADMINISTRATION.
3. Das Kommando MODIFY-USER-SWITCHES wird mit Eingabe der Nummer 1 ausgewählt. Der Operandenfragebogen des Kommandos erscheint mit seinen Hilfetexten und Standardwerten. Der Benutzer kann nun die zu prüfenden Werte eingeben.

7.3.5 Jobvariablen ersetzen

Das Ersetzen von Jobvariablen wird in SDF-SIM genauso durchgeführt wie außerhalb der Simulation.

```
 .
 .
/create-jv jv-name=jv1
/modify-jv jv-contents=jv1,set-value='bbb' ————————————————————————— (1)
 .
 .
/start-sdf-sim
 .
 .
*sh-f-att &(jv1)
(IN)       sh-f-att bbb ————————————————————————————————————————— (1)

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTATUS BBB,LIST=(SYSOUT)
*enter-job &(jv1),j-class=jcb00200
(IN)       enter-job bbb,j-class=jcb00200

ENTRY : JMGENTR
INTERFACE : ISL
STRING FORM
/ENTER BBB,ERASE=NO,JOB-CLASS=JCB00200,JOB-PRIO=STD,RERUN=NO,FLUSH=NO,
START=STD,REPEAT=STD,RUN-PRIO=STD,TIME=STD,PRINT=STD,PUNCH=STD,
LOG=(LISTING=NO)
*
 .
 .
```

1. Der Benutzer erzeugt eine Jobvariable mit dem Namen jv1 und weist ihr den Wert 'bbb' zu.
2. SDF-SIM ersetzt die Jobvariable durch deren Wert.

Bitte beachten Sie, dass die Ersetzung von Prozedurparametern nicht möglich ist.

7.3.6 Prozedur- und/oder Batchmodus simulieren

SDF-SIM ermöglicht den Test in einem simulierten Prozedurmodus, wenn PROC-MODE=*YES in der Anweisung DEFINE-ENVIRONMENT angegeben wird. Bei Angabe von TASK-TYPE=*BATCH in dieser Anweisung ist das Testen im simulierten Batchmodus möglich. Kommandos/Anweisungen müssen mit „/“ bzw. „//“ beginnen und können in Kleinbuchstaben eingegeben werden. Falls SPINOFF=*YES in der Anweisung DEFINE-ENVIRONMENT angegeben wurde, wird bei Auftreten eines Fehlers Spin-Off ausgelöst. Wird SDF-SIM in Prozeduren oder Batchdateien aufgerufen, muss bei zu simulierenden Kommandos/Anweisungen im Prozedur-/Batchmodus ein „*“ an erster Position der Zeile stehen. Die Eingaben werden sonst als reale Kommandos bzw. Anweisungen interpretiert und wirklich abgearbeitet.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*std(user=*no),proc-mode=*yes,spinoff=*yes,display=*no  (1)
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX' ACTIVATED
%  SDS0005 'GROUP' SYNTAX FILE '$TSOS.SYS.SDF.GROUP.SYNTAX.TSOS' ACTIVATED
*fstat aaa.                                                          }
(IN)       FSTAT AAA.                                                } ---------- (2)
%  CMD0661 DATA RECORD WAS READ INSTEAD OF COMMAND                   }
*/fstat aaa.
(IN)       /FSTAT AAA.

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTAT AAA.
*/shh-file-att aaa.                                                  )
(IN)       /SHH-FILE-ATT                                             )
%  CMD0186 OPERATION NAME 'SHH-FILE-ATT' UNKNOWN                     )
%  CMD0205 ERROR IN PRECEDING COMMAND OR PROGRAM AND PROCEDURE STEP  } (3)
TERMINATION: COMMANDS WILL BE IGNORED UNTIL /SET-JOB-STEP OR /LOGOFF )
OR /ABEND IS RECOGNIZED                                              )
*/sh-file-att aaa.
*/set-job-step
(IN)       /SET-JOB-STEP
ENTRY : SSMSTEP
INTERFACE : ISL
STRING FORM
/STEP
```

```
*/sh-file-att aaa.
(IN)       /SH-FILE-ATT AAA.

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTATUS AAA.,LIST=(SYSOUT)
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. Der Prozedurmodus (PROC-MODE=*YES) wird simuliert. Außerdem soll im Fehlerfall Spin-Off (SPINOFF=*YES) ausgelöst werden.
2. Da der „/“ nach „*“ fehlt, der im Prozedurmodus notwendig ist, erscheint eine Fehlermeldung.
3. Ein fehlerhafter Kommandoname wird eingegeben. Dadurch wird Spin-Off ausgelöst. Der nachfolgende Korrekturversuch wird ignoriert und kann erst dann erfolgreich sein, nachdem /SET-JOB-STEP abgearbeitet wurde.

7.3.7 SDF-SIM-Lauf innerhalb einer Prozedur oder eines Batchauftrages

SDF-SIM kann wie andere Dienstprogramme auch in Prozeduren und Batchaufträgen aufgerufen werden. Das bedeutet jedoch nicht implizit, dass Kommandos bzw. Anweisungen automatisch im simulierten Prozedur-/Batchmodus getestet werden. (Für den Test im simulierten Prozedur- oder Batchmodus muss die Anweisung DEFINE-ENVIRONMENT mit den Operanden PROC-MODE=*YES oder TASK-TYPE=*BATCH eingegeben werden.)

Bei Kommandos bzw. Anweisungen, die im Prozedur-/Batchmodus innerhalb einer Prozedur oder Batchdatei getestet werden, muss unbedingt ein Stern (*) an erster Position der Zeile stehen, da die Eingaben sonst als reale Kommandos bzw. Anweisungen interpretiert und tatsächlich abgearbeitet werden.

SDF-SIM in einer Prozedur

Das folgende Beispiel zeigt die Verwendung von SDF-SIM in der Prozedur SIM.PROC.

Prozedurdatei SIM.PROC

```
/BEGIN-PROCEDURE LOGGING=A
/ASSIGN-SYSDTA *SYSCMD
/START-SDF-SIM
//DEFINE-TEST-OBJECT *CMD                                    }
//DEFINE-ENVIRONMENT PAR-FILE=*STD(USER=*NO)                 } ————————————————————————  (1)
//START-SIMULATION                                           }
*CR-FILE AAA,SUPP=LLMLM
*/SH-FI-ATT BBBB
*/*
/ASSIGN-SYSDTA *PRIMARY
/END-PROCEDURE
```

Ablaufprotokoll der Prozedur SIM.PROC

```
(IN)     CALL-PROC SIM.PROC
(IN)     /BEGIN-PROCEDURE LOGGING=A
(IN)     /ASSIGN-SYSDTA *SYSCMD
(IN)     /START-SDF-SIM
(OUT)    %  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY
( )      ':2OSH:$TSOS.SYSLNK.SDF-SIM.045' IN PROCESS
(OUT)    %  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06
(  )      16:09:43' LOADED
(OUT)    %  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS
(  )      RESERVED
(OUT)   %  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
(IN)     //DEFINE-TEST-OBJECT *CMD
(IN)     //DEFINE-ENVIRONMENT PAR-FILE=*STD(USER=*NO)
(IN)     //START-SIMULATION
```

```
(OUT)    %  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX'
(   )     ACTIVATED
(IN)     *CR-FILE AAA,SUPP=LLMLM
(OUT)    (IN)       CR-FILE
(OUT)    %  CMD0051 INVALID OPERAND 'SUPPORT'
(OUT)    %  CMD0081 VALUE 'P' NOT CONTAINED IN KEYWORD LIST OF VALUE RANGE
(   )    '*PUBLIC-DISK() OR *PRIVATE-DISK() OR *TAPE() OR *NONE'
(IN)     */SH-FI-ATT BBBB ————————————————————————————————————————————  (2)
(OUT)    (IN)       /SH-FI-ATT BBBB
(OUT)
(OUT)    ENTRY : DCOFSTAT
(OUT)    INTERFACE : ISL
(OUT)    STRING FORM
(OUT)    /FSTATUS BBBB,LIST=(SYSOUT)
(IN)     */*
(OUT)    %  SDS0002 SDF-SIM TERMINATED NORMALLY
(IN)     /ASSIGN-SYSDTA *PRIMARY
(IN)     /END-PROCEDURE
```

1. Die Anweisungen zur Simulationsvorbereitung werden eingegeben und die Simulation wird gestartet.
2. Das Kommando CREATE-FILE wird simuliert. Da für den Operanden SUPPORT ein falscher Wert eingegeben wird, meldet SDF-SIM einen Fehler.
3. Es wurde kein Spin-off ausgelöst, obwohl ein Eingabefehler vorlag. Deshalb kann sofort das nächste Kommando simuliert werden. Wird ein „/“ vor dem Kommando eingegeben, muss unbedingt ein „*“ am Zeilenanfang stehen. Der „/“ ist hier jedoch nicht erforderlich, da nicht der Prozedurmodus simuliert wird.

SDF-SIM in einem Batchauftrag

Im folgenden Beispiel wird die Simulation der Anweisungen des Testprogrammes TEST-PRO in einem Batchauftrag durchgeführt. Dafür gelten dieselben Voraussetzungen wie in Prozeduren (siehe „SDF-SIM in einer Prozedur“ auf Seite 569). Aus diesem Grund wird auf das Ablaufprotokoll verzichtet.

Batchdatei SIM.ENTER

```
/LOGON
/START-SDF-SIM
//DEFINE-TEST-OBJECT *STMT(PROG-NAME=TESTPRO)
//DEFINE-ENVIRONMENT PAR-FILE=*STD(USER=TEST.USER),DISPLAY=*SHORT
//START-SIMULATION
*OPEN AAAA,TYPE=*USER, ...  ------------------------------------------  (1)
 .
 .
*END  ---------------------------------------------------------------------  (2)
/LOGOFF
```

1. OPEN ist eine Anweisung, die im Programm TESTPRO definiert ist.
2. Wurde bei DEFINE-TEST-OBJECT *STMT angegeben, so beendet END den SDF-SIM-Lauf mit der Meldung `END OF PROGRAM`. Die Anweisung END wird nicht simuliert.

7.3.8 Kommando, das durch eine Prozedur implementiert ist

Im folgenden Beispiel wird die Syntax eines selbstdefinierten Kommandos getestet. Das Kommando ist in der Benutzersyntaxdatei SF.TEST definiert.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*no(sys=*std,group=*n,user=sf.test),displ=*long ———— (1)
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX' ACTIVATED
%  SDS0005 'USER' SYNTAX FILE 'SF.TEST' ACTIVATED
*my-com first-op=aaa,second-op=bbbb ———————————————————————————— (2)
(IN)       my-com first-op=aaa,second-op=bbbb

INTERFACE : PROCEDURE                                    }
STRING FORM                                              } ———————————— (3)
/CALL MY-PROC,(FIRST-OP='AAA',SECOND-OP='BBBB')          }
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. In der Syntaxdateihierarchie der Testumgebung muss die Benutzersyntaxdatei SF.TEST aktiviert werden, die die Syntaxdefinition des Kommandos MY-COMMAND enthält.
2. Nach dem Starten der Simulation gibt der Benutzer das zu simulierende Kommando mit zwei Operanden ein.
3. SDF-SIM gibt ein Protokoll aus, in dem folgende Informationen enthalten sind:
   - Schnittstellentyp PROCEDURE
   - Übergabe in Form einer Zeichenkette
   - generierter Prozeduraufruf mit den Operanden als Prozedurparametern.

7.3.9 Normierten Übergabebereich anzeigen

Das folgende Beispiel zeigt die Ausgabe des normierten Übergabebereiches für eine simulierte SDF-A-Anweisung. Für Kommandos wird der normierte Übergabebereich nur dann ausgegeben, wenn diese mit ADD-CMD ...,IMPLEMENTOR=*TPR(...,CALL= *NEW...) definiert wurden (siehe SDF-A). Der normierte Übergabebereich wird wie im Beispiel ausgegeben, wenn:

– in der Anweisung DEFINE-ENVIRONMENT der Operand DISPLAY=*LONG gesetzt ist

  oder

– die Meldung
  `% SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT`
  mit „LONG“ beantwortet wird.

```
/start-sdf-sim
% BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A21' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
% BLS0524 LLM 'SDF-SIM', VERSION 'V04.5A21' OF '2001-12-06 16:09:43' LOADED
% BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTER 2001. ALL RIGHTS RESERVED
% SDS0001 SDF-SIM VERSION 'V04.5A20' STARTED
%//def-test *stmt(prog=$sdaed41) ————————————————————————————— (1)
%//def-env par-fi=*no(sys=$tsos.syssdf.sdf-a.041,group=*n,user=*n),-
%//display=*long ————————————————————————————————————————————— (2)
%//start-simulation
% CMD0692 THE SYSTEM SYNTAX FILE $TSOS.SYSSDF.SDF-A.041 DOESN‘T CONTAIN THE
COMMAND MODIFY-SDF-PARAMETERS
% SDS0005 ’SYSTEM’ SYNTAX FILE ’$TSOS.SYSSDF.SDF-A.041’ ACTIVATED
//
*add-cmd name=my-command,int-name=mycom,batch-allowed=*no,-
//
*impl=*proc(name='my-proc')
(IN)       add-command name=my-command,int-name=mycom,batch-allowed=*no,impl=
*proc(name=’my-proc’)
```

1. Als Testobjekte werden die Anweisungen des Programmes SDF-A festgelegt.
   Der interne Programmname von SDF-A V4.1A ist $SDAED41.
2. Für die Umgebung wird festgelegt, dass nur die Systemsyntaxdatei von SDF-A aktiviert wird. Mit DISPLAY=*LONG wird festgelegt, dass der normierte Übergabebereich und die Strukturbeschreibung ohne Abfrage sofort angezeigt wird. Die Ausgabe sehen Sie auf den folgenden Seiten.

```
*** TRANSFER AREA ***
~~~~~~~~~~~~~~~~~~~~
              A D  D C M D
(129418) 3A98C1C4 C4C3D4C4 40400000 00000000

(129428) 00000046 80080012 95D00000 00000000

(129438) 80160012 95F88016 001295EA 80160012

(129448) 95F08016 001295F4 00000000 00008007

(129458) 001295DC 00000000 00000000 00000000

(129468) 00000000 00008016 0012962C 80160012

(129478) 96308016 00129698 80130012 965E8013

(129488) 00129636 80130012 964A8013 00129672

(129498) 00000000 00000000 00000000 80050012

(1294A8) 960C8016 001295E4 80160012 95FC8013

(1294B8) 00129684 80160012 96000000 00000000

(1294D8) 00000000 00000000 00000000 00008016

(1294E8) 0012969E 80160012 96A80000 00000000

(129508) 00000000 00000000 00000000 00008016

(129518) 00129616 80160012 96260000 00000000

(129558) 00000000 00000000 00000000 80160012

(129568) 96AE0000 00000000 80160012 967E8016

(129578) 00129692 80160012 96448016 00129658

(129588) 80160012 966C8016 001296A2 00000000
                                 M Y  - C O M
(1295C8) 00000000 00000000 000AD4E8 60C3D6D4
          M A N D      M Y  C O M       S A
(1295D8) D4C1D5C4 0005D4E8 C3D6D400 0004E2C1
          M E      N A M E      N O      N O
(1295E8) D4C50004 D5C1D4C5 0002D5D6 0002D5D6
              N O      N O      P R  O C E D
(1295F8) 0002D5D6 0002D5D6 0009D7D9 D6C3C5C4
```

```
          U R E        M Y  - P R O  C
(129608) E4D9C500 0007D4E8 60D7D9D6 C300000E
          C A L L  - P R O  C E D U  R E
(129618) C3C1D3D3 60D7D9D6 C3C5C4E4 D9C50004
          N O N E      N O      Y E  S
(129628) D5D6D5C5 0002D5D6 0003E8C5 E2000000
                           Y E S        A L
(129638) 80160012 963E0003 E8C5E200 0003C1D3
          L                          Y E S
(129648) D3000000 80160012 96520003 E8C5E200
              A L  L
(129658) 0003C1D3 D3000000 80160012 96660003
          Y E S        A L  L
(129668) E8C5E200 0003C1D3 D3000000 80160012
                    N O      A L L
(129678) 967A0002 D5D60003 C1D3D300 00008016
                         Y E  S        A L L
(129688) 0012968C 0003E8C5 E2000003 C1D3D300
              Y E  S        N O      A L L
(129698) 0003E8C5 E2000002 D5D60003 C1D3D300
              C M  D        A L L
(1296A8) 0003C3D4 C4000003 C1D3D300 00000000

*** STRUCTURED DESCRIPTION ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
INTERNAL NAME OF THE COMMAND : ADDCMD
VERSION OF THE TRANSFER AREA : x     } (nur beim neuen Format des Über-
VERSION OF THE CMD/STMT      : x     } gabebereiches (ab SDF V4.1))
MAXIMUM NUMBER OF OPERANDS   :   70
OP( 1) :
 - TYPE   : STRUCT_NAME
 - LENGTH :   10
 - VALUE  :
      M Y - C  O M M A  N D
     D4E860C3 D6D4D4C1 D5C4
OP( 3) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
      N O
     D5D6
OP( 4) :
 - TYPE   : KEYW
 - LENGTH :    4
 - VALUE  :
      N A M E
     D5C1D4C5
OP( 5) :
```

```
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
      N O
     D5D6
OP( 6) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
      N O
     D5D6
OP( 8) :
 - TYPE   : ALPHA_NAME
 - LENGTH :    5
 - VALUE  :
      M Y C O  M
     D4E8C3D6 D4
OP(12) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
      N O
     D5D6
OP(13) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      Y E S
     E8C5E2
OP(14) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      Y E S
     E8C5E2
OP(15) :
 - TYPE   : STRUCTURE
       VALUE INTRODUCING THE STRUCTURE :
      - TYPE   : KEYW
      - LENGTH :    3
      - VALUE  :
           Y E S
          E8C5E2
OP(16) :
 - TYPE   : STRUCTURE
       VALUE INTRODUCING THE STRUCTURE :
      - TYPE   : KEYW
      - LENGTH :    3
```

```
      - VALUE  :
           Y E S
          E8C5E2
OP(17) :
 - TYPE   : STRUCTURE
       VALUE INTRODUCING THE STRUCTURE :
      - TYPE   : KEYW
      - LENGTH :    3
      - VALUE  :
           Y E S
          E8C5E2
OP(18) :
 - TYPE   : STRUCTURE
       VALUE INTRODUCING THE STRUCTURE :
      - TYPE   : KEYW
      - LENGTH :    2
      - VALUE  :
           N O
          D5D6
OP(21) :
 - TYPE   : C_STR
 - LENGTH :    7
 - VALUE  :
     M Y - P  R O C
    D4E860D7 D9D6C3
OP(22) :
 - TYPE   : KEYW
 - LENGTH :    4
 - VALUE  :
     S A M E
    E2C1D4C5
OP(23) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
     N O
    D5D6
OP(24) :
 - TYPE   : STRUCTURE
       VALUE INTRODUCING THE STRUCTURE :
      - TYPE   : KEYW
      - LENGTH :    3
      - VALUE  :
           Y E S
          E8C5E2
OP(25) :
 - TYPE   : KEYW
 - LENGTH :    9
```

```
 - VALUE  :
      P R O C  E D U R  E
     D7D9D6C3 C5C4E4D9 C5
OP(32) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
      N O
     D5D6
OP(33) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      C M D
     C3D4C4
OP(40) :
 - TYPE   : KEYW
 - LENGTH :   14
 - VALUE  :
      C A L L  - P R O  C E D U  R E
     C3C1D3D3 60D7D9D6 C3C5C4E4 D9C5
OP(41) :
 - TYPE   : KEYW
 - LENGTH :    4
 - VALUE  :
      N O N E
     D5D6D5C5
OP(53) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(55) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(56) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(57) :
 - TYPE   : KEYW
 - LENGTH :    3
```

```
 - VALUE  :
      A L L
     C1D3D3
OP(58) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(59) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(60) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
//
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

8 Meldungen

8.1 SDF-A-Meldungen

SDA0001 '(&01)' VERSION '(&00)' STARTED
SDA0001 '(&01)' VERSION '(&00)' GESTARTET

SDA0003 COMMENT DOES NOT END WITH A DOUBLE QUOTE (")
SDA0003 KOMMENTAR ENDET NICHT MIT ANFUEHRUNGS-ZEICHEN (")

Bedeutung
Der Kommentar muss mit einem Anfuehrungszeichen enden.

SDA0004 CHARACTER STRING '(&00)' DOES NOT END WITH A SINGLE QUOTE (')
SDA0004 ZEICHEN-FOLGE '(&00)' ENDET NICHT MIT HOCHKOMMA (')

Bedeutung
Die Zeichenfolge muss mit einem Hochkomma enden.

SDA0030 OPERAND NAME '(&00)' IS LONGER THAN 20 CHARACTERS
SDA0030 OPERANDEN-NAME '(&00)' IST LAENGER ALS 20 ZEICHEN

Bedeutung
Maximal zulaessige Laenge: 20 Zeichen.

SDA0031 SYNTAX ERROR IN OPERAND NAME '(&00)'
SDA0031 SYNTAX-FEHLER IM OPERANDEN-NAMEN '(&00)'

SDA0033 INVALID LIST SPECIFICATION AFTER OPERAND NAME
SDA0033 LISTEN-ANGABE NACH OPERANDEN-NAMEN FEHLERHAFT

SDA0034 OPERAND NAME BEFORE EQUALS SIGN '=' MISSING
SDA0034 OPERANDEN-NAME VOR GLEICHHEITS-ZEICHEN '=' NICHT VORHANDEN

Bedeutung
Operandennamen vor Gleichheitszeichen angeben.

SDA0035 OPERAND '(&00)' NOT PERMITTED IN PRESENT MODE
SDA0035 OPERAND '(&00)' IM AKTUELLEN MODUS UNZULAESSIG

Bedeutung
Der Operand (&00) ist im aktuellen Eingabemodus unzulaessig (siehe xxx-ALLOWED bzw. die Privilegienangaben in der Syntaxdatei).
Naehere Information kann dem BS2000-Handbuch 'SDF' (oder 'SDF-A') entnommen werden.

SDA0037 '(&00)' CANNOT BE ASSIGNED TO ANY OPERAND. REASON FOR ERROR: '(&01)'. INPUT IGNORED
SDA0037 '(&00)' KANN KEINEM OPERANDEN ZUGEORDNET WERDEN. FEHLER-URSACHE: '(&01)'. EINGABE IGNORIERT

SDA0038 ERROR IN COMMAND FORMAT: '(&00)' IGNORED
SDA0038 FEHLER IM KOMMANDO-AUFBAU: '(&00)' IGNORIERT

SDA0039 SEVERAL HELP TEXTS SPECIFIED FOR SAME LANGUAGE. LAST SPECIFICATION IS USED
SDA0039 FUER GLEICHE SPRACHE MEHRERE HILFE-TEXTE ANGEGEBEN. LETZTE EINGABE WIRD VERWENDET

SDA0043 LIST ELEMENT '(&00)' EXCEEDS MAXIMUM PERMITTED NUMBER
SDA0043 LISTEN-ELEMENT '(&00)' UEBERSCHREITET MAXIMAL ZULAESSIGE ANZAHL

SDA0044 CLOSING PARENTHESIS ')' IN LIST '(&00)' MISSING
SDA0044 ABSCHLIESSENDE KLAMMER ')' IN LISTE '(&00)' NICHT VORHANDEN

SDA0045 SPECIFICATION OF OPERAND VALUE '(&00)' IN LIST NOT PERMITTED
SDA0045 ANGABE DES OPERANDEN-WERTES '(&00)' IN LISTE UNZULAESSIG

SDA0046 NO APPROPRIATE STRUCTURE ACTIVATED. VALUE '(&01)' FOR OPERAND '(&00)' COULD NOT BE INTERPRETED
SDA0046 KEINE GEEIGNETE STRUKTUR AKTIVIERT. WERT '(&01)' FUER OPERAND '(&00)' NICHT AUSWERTBAR

SDA0047 CLOSING PARENTHESIS ')' IN STRUCTURE '(&00)' MISSING
SDA0047 ABSCHLIESSENDE KLAMMER ')' IN STRUKTUR '(&00)' NICHT VORHANDEN

SDA0048 STRUCTURE SPECIFICATION '(&00)' INVALID AND NOT PERMITTED FOR VALUE '(&01)'
SDA0048 STRUKTUR-ANGABE '(&00)' FEHLERHAFT UND FUER WERT '(&01)' UNZULAESSIG

SDA0050 '(&00)' CANNOT BE ASSIGNED TO ANY VALUE. INPUT IGNORED
SDA0050 '(&00)' KANN KEINEM WERT ZUGEORDNET WERDEN. EINGABE IGNORIERT

SDA0052 MODIFICATION OF ADDRESSED OPERAND VALUE OF LIST ELEMENT NOT POSSIBLE BY SPECIFYING '(&00)' IN SEMANTICS ERROR DIALOG
SDA0052 AENDERN DES ANGESPROCHENEN OPERANDEN-WERTES DES LISTEN-ELEMENTS DURCH EINGABE '(&00)' IN SEMANTIK-FEHLER-DIALOG NICHT MOEGLICH

SDA0053 VALUE '(&00)' NOT PERMITTED IN PRESENT MODE
SDA0053 WERT '(&00)' IM AKTUELLEN MODUS UNZULAESSIG

Bedeutung
Der Wert (&00) ist im aktuellen Eingabemodus unzulaessig (siehe xxx-ALLOWED bzw. die Privilegienangaben in der Syntaxdatei).
Naehere Information kann dem BS2000-Handbuch 'SDF' (oder 'SDF-A') entnommen werden.

SDA0055 DELETION OF LIST ELEMENT '(&00)' BY ENTERING A SHORTENED LIST NOT PERMITTED
SDA0055 LOESCHEN DES LISTEN-ELEMENTS '(&00)' DURCH EINGABE EINER VERKUERZTEN LISTE UNZULAESSIG

Bedeutung
Im Semantik-Fehlerdialog kann ein Listenelement (&00) nicht durch die Eingabe einer verkuerzten Liste geloescht werden.

SDA0057 INVALID VALUE '(&00)' NOT CORRECTED YET
SDA0057 FEHLERHAFTER WERT '(&00)' NOCH NICHT KORRIGIERT

SDA0058 OPERAND VALUE '(&00)' NOT PERMITTED IN CURRENT MODE
SDA0058 OPERANDEN-WERT '(&00)' IM AKTUELLEN MODUS UNZULAESSIG

SDA0061 OPERAND VALUE '(&00)' NOT IN PERMISSIBLE RANGE '(&01)'
SDA0061 OPERANDEN-WERT '(&00)' NICHT IM ZULAESSIGEN BEREICH '(&01)'

SDA0062 LENGTH OF OPERAND VALUE '(&00)' NOT IN PERMISSIBLE RANGE FOR DATA TYPE '(&01)'
SDA0062 LAENGE DES OPERANDEN-WERTES '(&00)' FUER DATENTYP '(&01)' NICHT IM ZULAESSIGEN BEREICH

SDA0063 THE OPERAND VALUE '(&00)' IS NOT MEMBER OF THE SINGLE VALUE LIST OF THE SCOPE '(&01)'
SDA0063 OPERANDEN-WERT '(&00)' IST NICHT IN EINZEL-WERT-LISTE DES BEREICHS '(&01)' ENTHALTEN

SDA0064 OPERAND VALUE '(&00)' DOES NOT MATCH DATA TYPE '(&01)'
SDA0064 OPERANDEN-WERT '(&00)' MIT DATENTYP '(&01)' NICHT VEREINBAR

SDA0065 SPECIFICATION '(&00)' CANNOT BE ASSIGNED TO ANY OF THE VALUES '(&01)' (VALUES-OVERLAPPING=YES)
SDA0065 EINGABE '(&00)' KANN KEINEM DER WERTE '(&01)' ZUGEORDNET WERDEN (VALUES-OVERLAPPING=YES)

SDA0067 LENGTH OF OPERAND VALUE '(&00)' MUST BE EVEN
SDA0067 LAENGE DES OPERANDEN-WERTES '(&00)' MUSS GERADE SEIN

Bedeutung
Der Operandenwert X_TEXT ist mit ODD-POSSIBLE=NO in der Syntaxdatei beschrieben. Deshalb muss der fuer den Operanden VALUE angegebene Wert gerade sein.

Maßnahme
Operandenwert korrigieren.

```
SDA0071   NO OPERAND SPECIFIED FOR COMMAND OR STATEMENT
SDA0071   KEIN OPERAND FUER KOMMANDO BZW. ANWEISUNG ANGEGEBEN

SDA0072   ATTRIBUTE SPECIFIED IN FILE NAME '(&00)' NOT PERMITTED
SDA0072   IM DATEI-NAMEN '(&00)' ANGEGEBENES ATTRIBUT UNZULAESSIG

SDA0073   INVALID GENERATION OR VERSION SPECIFICATION IN FILE NAME '(&00)'
SDA0073   IM DATEI-NAMEN '(&00)' ANGEGEBENE GENERATION BZW. VERSION FEHLERHAFT

SDA0074   INVALID USER IDENTIFICATION SPECIFIED IN FILE NAME '(&00)'
SDA0074   IM DATEI-NAMEN '(&00)' ANGEGEBENE BENUTZER-KENNUNG FEHLERHAFT

SDA0075   FILE NAME '(&00)' INVALID
SDA0075   DATEI-NAME '(&00)' FEHLERHAFT
```

Bedeutung
Der Dateiname (&00) enthaelt unzulaessige Zeichen.

```
SDA0076   INVALID GENERATION OR VERSION SPECIFICATION IN FILE NAME
SDA0076   IM DATEI-NAMEN ANGEGEBENE GENERATION BZW. VERSION FEHLERHAFT

SDA0077   SPECIFICATION OF POSITIONAL OPERANDS NOT PERMITTED AFTER OPERAND NAME WITH VALUE
SDA0077   ANGABE VON STELLUNGS-OPERANDEN NACH OPERANDEN-NAMEN MIT WERT UNZULAESSIG

SDA0078   INVALID CATALOG IDENTIFICATION SPECIFIED IN FILE NAME '(&00)'
SDA0078   IM DATEI-NAMEN '(&00)' ANGEGEBENE KATALOG-KENNUNG FEHLERHAFT

SDA0079   SPECIFIED CHARACTERS IN TEXT '(&00)' NOT PERMITTED
SDA0079   IM TEXT '(&00)' ANGEGEBENE ZEICHEN UNZULAESSIG
```

Bedeutung
Unzulaessige Zeichen:
- Gleichheitszeichen
- Leerzeichen
- Semikolon
- Klammer.

```
SDA008A   'PRODUCT-NAME' MUST BE SPECIFIED WHEN ADDING CORRECTION-INFORMATION TO AN
          INSTALLATION SYNTAX FILE
SDA008A   'PRODUKT-NAME' MUSS ANGEGEBEN WERDEN, WENN KORREKTUR-INFORMATION IN
          INSTALLATIONS-SYNTAX-DATEI EINGETRAGEN WIRD

SDA008B   PRODUCT-NAME '(&00)' DIFFERS FROM REGISTERED ONE '(&01)'
SDA008B   PRODUKT-NAME '(&00)' UNTERSCHEIDET SICH VOM REGISTRIERTEN NAMEN '(&01)'

SDA008C   PRODUCT-VERSION '(&00)' DIFFERS FROM REGISTERED ONE '(&01)'
SDA008C   PRODUKT-VERSION '(&00)' UNTERSCHEIDET SICH VON REGISTRIERTER VERSION '(&01)'

SDA008D   'SOURCE' CORRECTIONS CAN ONLY BE REGISTERED IN COMPONENT SYNTAX FILES
SDA008D   'SOURCE'-KORREKTUREN KOENNEN NUR IN KOMPONENTEN-SYNTAX-DATEI EINGETRAGEN WERDEN

SDA008E   'OBJECT' CORRECTIONS CAN ONLY BE REGISTERED IN KPSD, SESD, INSD SYNTAX FILES
SDA008E   'OBJEKT'-KORREKTUREN KOENNEN NUR IN KPSD, SESD ODER INSD EINGETRAGEN WERDEN
```

```
SDA008F  THE CORRECTION-INFORMATION CANNOT BE MODIFIED BY THE CURRENT PROGRAM VERSION
SDA008F  KORREKTUR-INFORMATION KANN NICHT MIT DIESER PROGRAMM-VERSION GEAENDERT WERDEN
```

Bedeutung
Das aktuelle Programm ist zu alt.

Maßnahme
Bitte verwenden Sie eine neue Programmversion.

```
SDA0080  NUMBER (&00) OUTSIDE PERMITTED RANGE
SDA0080  ZAHL (&00) AUSSERHALB DES ZULAESSIGEN BEREICHS

SDA0081  VALUE '(&00)' NOT CONTAINED IN KEYWORD LIST OF VALUE RANGE '(&01)'
SDA0081  WERT '(&00)' NICHT IN SCHLUESSELWORT-LISTE DES WERTE-BEREICHS '(&01)' ENTHALTEN

SDA0082  ABBREVIATION '(&00)' AMBIGUOUS WITH REGARD TO '(&01)'
SDA0082  ABKUERZUNG '(&00)' BEZUEGLICH '(&01)' MEHRDEUTIG
```

Maßnahme
Eindeutige Abkuerzung verwenden.

```
SDA0083  NAME '(&00)' UNKNOWN
SDA0083  NAME '(&00)' NICHT BEKANNT

SDA0084  '(&00)' EQUIVALENT TO TPR COMMAND(S) '(&01)'. COMMAND REJECTED
SDA0084  '(&00)' ENTSPRICHT TPR-KOMMANDO(S) '(&01)'. KOMMANDO ABGEWIESEN
```

Bedeutung
Moegliche Ursachen:
- Das angegebene Kommando ist bereits vorhanden.
- Das angegebene Kommando entspricht der Abkuerzung eines bereits vorhandenen Kommandos bzw. bereits definierten Kommandos.

(&00): Eingegebenes Kommando
(&01): Bereits vorhandenes Kommando bzw. Liste der bereits definierten Kommandos.

```
SDA0085  V-RECORD OF COMPONENT SYNTAX FILE IS FULL
SDA0085  V-OBJEKT DER KOMPONENTEN-SYNTAX-DATEI IST VOLL
```

Bedeutung
Die 255 Positionen der Korrektur-Informationstabelle sind belegt.

Maßnahme
Nicht benoetigte Eintraege aus der Tabelle loeschen.

SDA0086 NO MORE ENTRIES POSSIBLE IN V-RECORD OF COMPONENT SYNTAX FILE
SDA0086 KEINE WEITEREN EINTRAEGE IN V-OBJEKT DER KOMPONENTEN-SYNTAX-DATEI MOEGLICH

Bedeutung
Nur einige der angegebenen PM-Operandenwerte wurden in die Komponenten-Syntaxdatei geschrieben.

Maßnahme
Nicht benoetigte Eintraege aus der Tabelle loeschen.

SDA0087 KERNEL OR COMPONENT SYNTAX FILE CANNOT BE OF 'USER' TYPE
SDA0087 KERNEL- BZW. KOMPONENTEN-SYNTAX-DATEI KANN NICHT VOM TYP 'USER' SEIN

SDA0088 CORRECTION NUMBER '(&00)' NOT CONTAINED IN SYNTAX FILE
SDA0088 KORREKTUR-NUMMER '(&00)' NICHT IN SYNTAX-DATEI ENTHALTEN

Bedeutung
Die Korrekturinformation ist im V-Objekt nicht enthalten.

SDA0089 THERE ARE NO SIGNIFICANT DIGITS OR MORE THAN 10 SIGNIFICANT DIGITS HAVE BEEN FOUND
SDA0089 ANZAHL BEDEUTSAMER ZIFFERN IST NULL ODER GROESSER ALS 10

Bedeutung
Fuer den SDF-Datentyp FIXED koennen hoechstens 10 Zeichen angegeben werden. Es muss jedoch wenigstens ein Zeichen angegeben werden.

SDA009A CHECK ERROR: THE (&00) '(&01)' DOESN'T MATCH
SDA009A PRUEF-FEHLER: (&00) '(&01)' STIMMT NICHT

Bedeutung
Der V-Satz enthaelt eine Angabe zum SOFTWARE-UNIT-NAME, zur VERSION und zur COMPONENT-VERSION. Diese Werte koennen als Pruef-Werte benutzt werden, bevor der V-Satz modifiziert wird.

(&00) identifiziert die Angabe.

(&01) identifiziert den Wert.

SDA009B THE CORRECTION-INFORMATION CANNOT BE MODIFIED: DATA ORGANIZATION NOT RECOGNIZED
SDA009B KORREKTUR-INFORMATION KANN NICHT GEAENDERT WERDEN: DATEN ORGANISATION NICHT ERKANNT. EINGABE IGNORIERT

Bedeutung
Die V-Satz-Struktur ist falsch.

SDA009C THE CORRECTION-INFORMATION CANNOT BE MODIFIED: THE V-RECORD IS FULL
SDA009C KORREKTUR-INFORMATION KANN NICHT GEAENDERT WERDEN: DER V-SATZ IST VOLL

Bedeutung
Der V-Satz ist voll.

Maßnahme
Nicht benoetigte Korrektur-Informationen loeschen.

SDA0090 OPERAND VALUE '(&00)' VIOLATES PERMITTED LOGICAL LENGTH FOR DATA TYPE '(&01)'
SDA0090 LAENGE DES OPERANDEN-WERTES '(&00)' NICHT IM ZULAESSIGEN LOGISCHEN BEREICH FUER DATENTYP '(&01)'

SDA0099 MANDATORY OPERAND INVALID OR MISSING
SDA0099 GEFORDERTER OPERAND UNGUELTIG BZW. NICHT VORHANDEN

Bedeutung
Fuer den geforderten Operanden wurde ein fehlerhafter Wert angegeben.

SDA0159 LANGUAGE '(&00)' NOT DEFINED IN GLOBALS OF CURRENT SYNTAX FILE HIERARCHY. LANGUAGE '(&01)' IS USED
SDA0159 SPRACHE '(&00)' IN GLOBAL-INFORMATION DER AKTUELLEN SYNTAX-DATEI-HIERARCHIE NICHT DEFINIERT. SPRACHE '(&01)' WIRD VERWENDET

SDA0300 DMS ERROR '(&01)' WHEN ACCESSING FILE '(&00)'. IN SYSTEM MODE: /HELP-MSG DMS(&01)
SDA0300 DVS-FEHLER '(&01)' BEIM ZUGRIFF AUF DATEI '(&00)'. IM SYSTEM-MODUS: /HELP-MSG DMS(&01)

Bedeutung
Naehere Information ueber den DVS-Fehlerschluessel kann ueber /HELP-MSG im Systemmodus erfragt bzw. dem BS2000-Handbuch 'Systemmeldungen' entnommen werden.

SDA0301 ERROR DURING OUTPUT TO SYSLST
SDA0301 FEHLER WAEHREND AUSGABE NACH SYSLST

SDA0302 INVALID VERSION OF SYNTAX FILE '(&00)' SELECTED
SDA0302 GEWAEHLTE VERSION DER SYNTAX-DATEI '(&00)' FEHLERHAFT

SDA0303 JVS ERROR '(&01)' WHEN ACCESSING JV '(&00)'. IN SYSTEM MODE: /HELP-MSG JVS(&01)
SDA0303 JVS-FEHLER '(&01)' BEIM ZUGRIFF AUF JV '(&00)'. IM SYSTEM-MODUS: /HELP-MSG JVS(&01)

Bedeutung
Naehere Information ueber den JVS-Fehlerschluessel kann ueber /HELP-MSG im Systemmodus erfragt bzw. dem Handbuch 'JV (BS2000)' entnommen werden.

JVS: Job Variable Service

SDA0304 DMS ERROR '(&00)' WHEN COPYING KERNEL SYNTAX FILE TO '(&01)'. IN SYSTEM MODE: /HELP-MSG DMS(&00)
SDA0304 DVS-FEHLER '(&00)' BEIM KOPIEREN DER KERNEL-SYNTAX-DATEI AUF '(&01)'. IM SYSTEM-MODUS: /HELP-MSG DMS(&00)

Bedeutung
Naehere Information ueber den DVS-Fehlerschluessel kann ueber /HELP-MSG im Systemmodus erfragt bzw. dem BS2000-Handbuch 'Systemmeldungen' entnommen werden.

SDA0305 VERSION DOES NOT MATCH DOD-FORMAT CONVENTION. INPUT IGNORED
SDA0305 VERSION STIMMT NICHT MIT DOD-FORMAT-KONVENTION UEBEREIN. EINGABE IGNORIERT

Bedeutung
Die angegebene Version entspricht nicht den Syntaxregeln.

SDA0306 KERNEL SYNTAX FILE MUST BE OF 'SYSTEM' TYPE. INPUT IGNORED
SDA0306 KERNEL-SYNTAX-DATEI MUSS TYP 'SYSTEM' ENTSPRECHEN. EINGABE IGNORIERT

SDA0307 KERNEL SYNTAX FILE CANNOT BE UPDATED. INPUT IGNORED
SDA0307 KERNEL-SYNTAX-DATEI KANN NICHT AKTUALISIERT WERDEN. EINGABE IGNORIERT

SDA0308 ERROR DURING OPEN OF (&00). THIS MAY LEAD TO SOME PROBLEMS
SDA0308 FEHLER BEIM OEFFNEN VON (&00). DAS KANN ZU PROBLEMEN FUEHREN

SDA0316 KEYWORD '(&00)' IN INPUT ALREADY EXISTS
SDA0316 SCHLUESSELWORT '(&00)' IN EINGABE BEREITS VORHANDEN

SDA0335 OPERAND NAME '(&00)' ALREADY EXISTS
SDA0335 OPERANDEN-NAME '(&00)' BEREITS VORHANDEN

SDA0350 INTERNAL ERROR:'(&00)'
SDA0350 INTERNER FEHLER:'(&00)'

SDA0372 AUTHORIZATION RANGE SPECIFIED IN USER SYNTAX FILE FOR COMMAND '(&00)' WIDER THAN THAT SPECIFIED IN SYSTEM AND GROUP SYNTAX FILES
SDA0372 ANGEGEBENER BERECHTIGUNGS-BEREICH FUER KOMMANDO '(&00)' IN BENUTZER-SYNTAX-DATEI GROESSER ALS IN SYSTEM- UND GRUPPEN-SYNTAX-DATEI

SDA0373 DESCRIPTION OF SYSTEM INTERFACE FOR COMMAND '(&00)' MODIFIED IN SYNTAX FILE
SDA0373 BESCHREIBUNG DER SYSTEM-SCHNITTSTELLE FUER KOMMANDO '(&00)' IN SYNTAX-DATEI GEAENDERT

Bedeutung
Moegliche Ursachen:
- Ein Attribut des Kommandos wurde in der Benutzersyntaxdatei geaendert.
- Das Kommando wird von einem in der Benutzersyntaxdatei definierten Kommando ueberschrieben.

SDA0376 SPECIFICATION OF POSITIONAL OPERANDS AT POSITION '(&00)' NOT PERMITTED
SDA0376 ANGABE VON STELLUNGS-OPERANDEN BEI POSITION '(&00)' UNZULAESSIG

SDA0377 NUMBER OF DATA TYPES PER OPERAND TOO LARGE
SDA0377 ANZAHL DER DATENTYPEN PRO OPERAND ZU GROSS

```
SDA0378   STRUCTURE DEPTH GREATER THAN 5
SDA0378   STRUKTURTIEFE UEBERSCHREITET 5

SDA0379   IN COMMAND '(&00)' FOR OPERAND '(&01)', VALUES WHICH ARE NOT COMPATIBLE WITH
          FILE HIERARCHY ARE ENTERED IN USER OR GROUP SYNTAX FILE
SDA0379   IM KOMMANDO '(&00)' FUER OPERANDEN '(&01)' EINGETRAGENE WERTE IN BENUTZER- BZW.
          GRUPPEN-SYNTAX-DATEI NICHT MIT DATEI-HIERACHIE VEREINBAR

SDA0380   ERROR WHEN STARTING '$.SDF-A-V1'. PROGRAM TERMINATED ABNORMALLY
SDA0380   FEHLER BEIM STARTEN VON '$.SDF-A-V1'. PROGRAMM ABNORMAL BEENDET
```

Bedeutung
Das Programm $.SDF-A-V1 wird gestartet, wenn der Schalter 15 gesetzt ist, um mit SDF-A V1.0D erstellte V1-Format-Syntaxdateien zu aendern.

Maßnahme
Schalter 15 zuruecksetzen bzw. den Systemverwalter verstaendigen.

```
SDA0381   ERROR WHEN READING A STATEMENT. PROGRAM TERMINATED ABNORMALLY
SDA0381   LESE-FEHLER BEI EINER ANWEISUNG. PROGRAMM ABNORMAL BEENDET

SDA0382   ERROR WHEN READING //OPEN-SYNTAX-FILE STATEMENT. PROGRAM TERMINATED ABNORMALLY
SDA0382   LESE-FEHLER BEI ANWEISUNG //OPEN-SYNTAX-FILE. PROGRAMM ABNORMAL BEENDET

SDA0383   FORMAT OF SYNTAX FILE HAS BEEN CHANGED
SDA0383   FORMAT DER SYNTAX-DATEI WECHSELTE
```

Bedeutung
Die Syntaxdatei wurde mit einer vorherigen Version (V1.0D oder kleiner) erzeugt. Die Struktur der Syntaxdatei ist nun veraendert und die Syntaxdatei kann nicht mehr mit einer vorherigen Version geoeffnet werden.

```
SDA0384   NO VALUES SPECIFIED FOR OPERAND '(&00)'. OPERAND WILL BE DELETED
SDA0384   FUER OPERAND '(&00)' KEINE WERTE ANGEGEBEN. OPERAND WIRD GELOESCHT

SDA0385   NO STRUCTURE OPERANDS ASSIGNED TO VALUE '(&00)'. STRUCTURE WILL BE DELETED
SDA0385   KEINE STRUKTUR-OPERANDEN DEM WERT '(&00)' ZUGEORDNET. STRUKTUR WIRD GELOESCHT
```
(C) Routing code: * Weight: 99

```
SDA0386   ERROR WHEN STARTING '$.SDF-A-V3'. PROGRAM TERMINATED ABNORMALLY
SDA0386   FEHLER BEIM STARTEN VON '$.SDF-A-V3'. PROGRAMM ABNORMAL BEENDET
```

Bedeutung
Das Programm $.SDF-A-V3 wird gestartet, wenn 'DEFINE-ENVIRONMENT *V3' zum Erzeugen oder Aendern einer Syntaxdatei im V3-Format angegeben wurde.

Maßnahme
Den Systemverwalter verstaendigen.

```
SDA0388   WARNING: DEFAULT VALUE OF OPERAND '(&00)' LIES OUTSIDE PERMISSIBLE RANGE BECAUSE
          OF '(&01)'
SDA0388   WARNUNG: STANDARD-WERT DES OPERANDEN '(&00)' ENTSPRICHT WEGEN '(&01)' NICHT
          ZULAESSIGEM WERTE-BEREICH
```

SDA0389 OBJECT '(&00)' ALREADY DEFINED IN SYNTAX FILE
SDA0389 OBJEKT '(&00)' IN SYNTAX-DATEI BEREITS DEFINIERT

SDA0390 COPY DESTINATION NOT DEFINED OR ITS TYPE IS NOT COMPATIBLE
SDA0390 KOPIER-ZIEL NICHT DEFINIERT BZW. TYP DES KOPIER-ZIELS FEHLERHAFT

Bedeutung
Moegliche Ursachen:
- Vor der Anweisung //COPY fehlt eine //ADD- bzw. //EDIT-Anweisung.
- Der Typ des Kopierziels ist unvereinbar mit dem des kopierten Objekts.

SDA0391 INVALID SYNTAX FILE OBJECT. INPUT IGNORED
SDA0391 SYNTAX-DATEI-OBJEKT FEHLERHAFT. EINGABE IGNORIERT

SDA0392 INTERNAL OPERAND OR VALUE '(&00)' ALREADY ASSIGNED
SDA0392 INTERNER OPERAND ODER WERT '(&00)' BEREITS ZUGEORDNET

SDA0393 CURRENT COMMAND OR CURRENT STATEMENT NOT DEFINED
SDA0393 AKTUELLES KOMMANDO BZW. AKTUELLE ANWEISUNG NICHT DEFINIERT

SDA0395 CURRENT OPERAND OR VALUE NOT DEFINED
SDA0395 AKTUELLER OPERAND BZW. WERT NICHT DEFINIERT

SDA0396 SPECIFIED DATA TYPE CANNOT BE FOUND
SDA0396 ANGEGEBENER DATENTYP NICHT VORHANDEN

SDA0397 SELECTED OPERAND OR VALUE NOT AVAILABLE AT CURRENT HIERARCHY LEVEL OF SYNTAX FILE
SDA0397 GEWAEHLTER OPERAND BZW. WERT AUF AKTUELLER HIERARCHIE-STUFE DER SYNTAX-DATEI NICHT VERFUEGBAR

SDA0401 WARNING: STANDARD NAME '(&00)' OF CONTROL FILE ADDED TO '(&01)'
SDA0401 WARNUNG: STANDARD-NAME '(&00)' AUS KONTROLL-DATEI WURDE '(&01)' HINZUGEFUEGT

Bedeutung
Um Inkonsistenz zu vermeiden, kann der Standardname der Kontrolldatei nicht durch die Syntaxdatei unterdrueckt werden.

SDA0402 WARNING: STANDARD NAME '(&00)' OF CONTROL FILE ADDED TO OPERAND '(&01)'
SDA0402 WARNUNG: STANDARD-NAME '(&00)' AUS KONTROLL-DATEI WURDE DEM OPERANDEN '(&01)' HINZUGEFUEGT

Bedeutung
Um Inkonsistenz zu vermeiden, kann der Standardname der Kontrolldatei nicht durch die Syntaxdatei unterdrueckt werden.

SDA0403 WARNING: STANDARD NAME '(&00)' OF CONTROL FILE ADDED TO VALUE '(&01)'
SDA0403 WARNUNG: STANDARD-NAME '(&00)' AUS KONTROLL-DATEI WURDE DEM WERT '(&01)' HINZUGEFUEGT

Bedeutung
Um Inkonsistenz zu vermeiden, kann der Standardname der Kontrolldatei nicht durch die Syntaxdatei unterdrueckt werden.

SDA0404 VALUE TYPE '(&00)' MUST BE SYNTACTICALLY SEPARATED FROM VALUE TYPE '(&01)'
SDA0404 WERTTYP '(&00)' MUSS VON WERTTYP '(&01)' SYNTAKTISCH GETRENNT WERDEN

SDA0405 ERROR CAN LEAD TO PROBLEMS OF PRIVILEGING IF SAME HIERARCHY OF SYNTAX FILES IS ACTIVATED
SDA0405 WIRD GLEICHE SYNTAX-DATEIEN-HIERARCHIE AKTIVIERT, KANN FEHLER ZU PRIVILEGIERUNGS-PROBLEMEN FUEHREN

SDA0406 VALUE '(&00)' DOES NOT MATCH CORRESPONDING SYNTAX TYPE BECAUSE OF '(&01)'
SDA0406 WERT '(&00)' ENTSPRICHT WEGEN '(&01)' NICHT ZUGEHOERIGEM SYNTAXTYP

SDA0407 CORRECTION REJECTED OR NOT POSSIBLE. STATEMENT IGNORED
SDA0407 KORREKTUR ABGEWIESEN BZW. NICHT MOEGLICH. ANWEISUNG IGNORIERT

SDA0408 ABBREVIATION '(&00)' INVALID AS AN ABBREVIATION OF MAIN VALUE '(&01)'
SDA0408 ABKUERZUNG '(&00)' ALS ABKUERZUNG DES HAUPT-WERTES '(&01)' UNZULAESSIG

Maßnahme
Eindeutige Abkuerzung verwenden.

SDA0409 KEYWORD '(&00)' TOO LONG
SDA0409 SCHLUESSELWORT '(&00)' ZU LANG

Bedeutung
Maximal zulaessige Laenge: 30 Zeichen.

SDA0410 A SINGLE VALUE MUST BE SPECIFIED FOR 'KEYWORD' DATA TYPE
SDA0410 FUER DATENTYP 'KEYWORD' ANGABE EINES EINZEL-WERTES ERFORDERLICH

SDA0411 EXTERNAL COMMAND OR STATEMENT NAME '(&00)' ALREADY ASSIGNED
SDA0411 EXTERNER KOMMANDO- BZW. ANWEISUNGSNAME '(&00)' BEREITS VERGEBEN

Bedeutung
Moegliche Ursachen:
- Das Kommando bzw. die Anweisung ist vorhanden.
- Das Kommando bzw. die Anweisung wurde auf der Benutzerebene entfernt, ist aber in der Gruppen- bzw. Systemebene bekannt.

SDA0412 WARNING: RESULT OPERAND LEVEL ALLOCATED TO OPERAND '(&00)' IN '(&01)' IS TOO LARGE AND WILL BE CORRECTED
SDA0412 WARNUNG: DEM OPERANDEN '(&00)' IN '(&01)' ZUGEWIESENER 'RESULT-OPERAND-LEVEL' IST ZU GROSS UND WIRD KORRIGIERT

SDA0413 FLAT STRUCTURE SPECIFIED FOR VALUE '(&00)' ALTHOUGH STRUCTURE ALLOWED WITHIN A LIST
SDA0413 FLACHE STRUKTUR FUER WERT '(&00)' ANGEGEBEN, OBWOHL STRUKTUR IN EINER LISTE ZUGELASSEN IST

SDA0414 DELETION OF COMMAND OR STATEMENT '(&00)' NOT PERMITTED
SDA0414 LOESCHEN VON KOMMANDO BZW. ANWEISUNG '(&00)' UNZULAESSIG

SDA0415 DELETION OF OPERAND '(&00)' NOT PERMITTED
SDA0415 LOESCHEN DES OPERANDEN '(&00)' UNZULAESSIG

```
SDA0416   DELETION OF VALUE '(&00)' NOT PERMITTED
SDA0416   LOESCHEN DES WERTES '(&00)' UNZULAESSIG

SDA0417   INTERNAL NAME '(&00)' ALREADY ASSIGNED
SDA0417   INTERNER NAME '(&00)' BEREITS VERGEBEN

SDA0418   '(&00)' OPERAND IGNORED. PROCESSING CONTINUES
SDA0418   OPERAND '(&00)' IGNORIERT. BEARBEITUNG WIRD FORTGESETZT
```

Bedeutung

– INPUT-FORM:
  TPR-Kommandos koennen nur dann in INVARIANT- oder STANDARD-Eingabeform generiert werden, wenn sie eine ISL-Schnittstelle groesser als Version 1 haben.

– SOFTWARE-UNIT-NAME:
  Der Operand hat keine Wirkung fuer die angegebene Syntaxdatei.

– VERSION:
  Der Operand kann nur ausgewertet werden, wenn der SOFTWARE-UNIT-NAME angegeben wird.

```
SDA0419   'COMMAND-REST' DATA TYPE ONLY PERMITTED AT COMMAND LEVEL
SDA0419   DATENTYP 'COMMAND-REST' NUR AUF KOMMANDO-EBENE ZULAESSIG

SDA0420   WARNING: SOME OBJECTS NOT COPIED. STATEMENT PARTIALLY EXECUTED
SDA0420   WARNUNG: EINIGE OBJEKTE NICHT KOPIERT. ANWEISUNG TEILWEISE AUSGEFUEHRT
```

Bedeutung
Ein Problem trat waehrend der Bearbeitung eines Objektes auf.
Siehe vorangegangene Meldungen.

```
SDA0421   VALUE SPECIFIED FOR 'COMMAND-REST' DATA TYPE NOT PERMITTED
SDA0421   ANGEBENER WERT FUER DATEN-TYP 'COMMAND-REST' UNZULAESSIG

SDA0422   FOR A VALUE INTRODUCED BY AN OPERAND DECLARED 'LIST-POSSIBLE=NO', LIST
          SPECIFICATION NOT POSSIBLE
SDA0422   FUER DURCH OPERANDEN 'LIST-POSSIBLE=NO' EINGEFUEHRTEN WERT KEINE LISTEN-ANGABE
          MOEGLICH

SDA0423   OPERATION ADD-OP NOT PERMITTED AT THE MOMENT
SDA0423   OPERATION ADD-OP ZUR ZEIT NICHT ERLAUBT
```

Bedeutung
Definition vom Datentyp *command-rest ist nur fuer den letzten Operanden erlaubt.

```
SDA0424   FILE NAME WITH GENERATION OR VERSION SPECIFICATION MUST NOT BE FOLLOWED BY A
          STRUCTURE
SDA0424   DATEI-NAMEN MIT GENERATIONS- BZW. VERSIONS-ANGABE DARF KEINE STRUKTUR
          HINZUGEFUEGT WERDEN

SDA0425   VALID OUTPUT POSITION SPECIFICATION FOR THIS OPERAND MANDATORY
SDA0425   GUELTIGE ANGABE DER AUSGABE-POSITION FUER DIESEN OPERANDEN ERFORDERLICH
```

SDA0426 SELECTED OPERAND OR VALUE DOES NOT MATCH REQUIRED TYPE OF OPERAND OR VALUE
SDA0426 GEWAEHLTER OPERAND BZW. WERT ENTSPRICHT NICHT ERFORDERLICHEM OPERANDEN- BZW. WERTETYP

SDA0427 NO VALUE CAN BE SPECIFIED FOR 'DEVICE' DATA TYPE
SDA0427 KEIN WERT KANN ANGEGEBEN WERDEN FUER DATENTYP 'DEVICE'

SDA0428 WARNING: OPERAND DECLARED 'PRESENCE=INTERNAL-ONLY' WHICH INTRODUCES A STRUCTURE CANNOT BE DEFINED IN LIST. 'PRESENCE' WAS RESET TO 'NORMAL'
SDA0428 WARNUNG: ALS 'PRESENCE=INTERNAL-ONLY' ERKLAERTER OPERAND, DER STRUKTUR EINLEITET, KANN NICHT IN LISTE DEFINIERT WERDEN. 'PRESENCE' WURDE AUF 'NORMAL' ZURUECKGESETZT

SDA0430 VALUE '(&00)' VIOLATES IMPLEMENTATION BOUNDARIES
SDA0430 WERT '(&00)' VERLETZT IMPLEMENTIERUNGS-GRENZEN

Bedeutung
Fuer jeden Datentyp gibt es Grenzen, die im gefuehrten Dialog ausgegeben werden (z.B. c-string 1..1800).

SDA0431 INVALID DEFAULT VALUES. STATEMENT IGNORED
SDA0431 STANDARD-WERTE FEHLERHAFT. ANWEISUNG IGNORIERT

SDA0432 TRANSFER AREA TOO SMALL. STATEMENT IGNORED
SDA0432 UEBERGABE-BEREICH ZU KLEIN. ANWEISUNG IGNORIERT

SDA0433 OPERAND DECLARED 'PRESENCE=INTERNAL-ONLY' MUST HAVE DEFAULT VALUE
SDA0433 ALS 'PRESENCE=INTERNAL-ONLY' DEFINIERTER OPERAND BENOETIGT STANDARD-WERT

SDA0434 IF DATA TYPE IS CHANGED TO 'INTEGER' OR 'TIME', REPRESENTATION 'OUT-FORM=' IS MANDATORY
SDA0434 BEI AENDERUNG EINES DATENTYPS AUF 'INTEGER' ODER 'TIME' IST DARSTELLUNG 'OUT-FORM=' ERFORDERLICH

SDA0435 STRUCTURE MUST NOT FOLLOW 'TEXT' OR 'COMMAND-REST' DATA TYPE
SDA0435 AUF DATENTYP 'TEXT' BZW. 'COMMAND-REST' DARF KEINE STRUKTUR FOLGEN

SDA0436 LOWEST LIMIT GREATER THAN HIGHEST LIMIT
SDA0436 NIEDRIGSTE GRENZE GROESSER ALS HOECHSTE GRENZE

Bedeutung
Moegliche Ursachen:
- SHORTEST-LENGTH ist groesser als LONGEST-LENGTH (z. B.: 10..2 ist falsch, 2..10 ist richtig).
- LOWEST-VALUE ist groesser als HIGHEST-VALUE (z. B.: (2,-2) ist falsch, (-2,2) ist richtig).

SDA0437 OPERAND '(&00)' IN COMMAND '(&01)' HAS INVALID RESULT OPERAND NAME. CORRECTION TO VALUE '*POS(1)'
SDA0437 OPERAND '(&00)' IM KOMMANDO '(&01)' BESITZT UNGUELTIGEN 'RESULT-OPERAND-NAME'. KORREKTUR AUF WERT '*POS(1)'

SDA0438 START OF PROCESSING OF OBJECT '(&00)'
SDA0438 BEGINN DER BEARBEITUNG VON OBJEKT '(&00)'

SDA0439 'OUTPUT=DROP-OPERAND' NOT PERMITTED FOR VALUES WITH 'LIST-ALLOWED=YES'
SDA0439 'OUTPUT=DROP-OPERAND' FUER WERTE MIT 'LIST-ALLOWED=YES' UNZULAESSIG

SDA044A WARNING: THE INTERNAL NAME OF THE COMMAND SHOULD START WITH '(&00)' OR '(&01)'
SDA044A WARNUNG: INTERNER KOMMANDO-NAME SOLLTE MIT '(&00)' ODER '(&01)' BEGINNEN

Bedeutung
(&01) = Kommandoklasse, die bei der Erzeugung der KPSD angegeben wurde.

SDA0440 NO GLOBAL INFORMATION AVAILABLE IN CURRENT SYNTAX FILE. TASK TERMINATED ABNORMALLY
SDA0440 KEINE GLOBAL-INFORMATION IN AKTUELLER SYNTAX-DATEI VORHANDEN. PROZESS ABNORMAL BEENDET

SDA0441 SPECIFICATION OF STRUCTURE FOR OPERAND '(&00)' AMBIGUOUS
SDA0441 STRUKTUR-ANGABE FUER OPERANDEN '(&00)' MEHRDEUTIG

SDA0442 RECURSIVE DEFINITION OF VALUES OR OPERANDS NOT PERMITTED
SDA0442 REKURSIVE DEFINITION VON OPERANDEN UNZULAESSIG

Bedeutung
Um die Werte 'w2' bzw. Operanden 'op2' in der Struktur 'op1=w1(op2=w2)' zu definieren, duerfen die Werte 'w1' bzw. Operanden 'op1' nicht mit dem Operanden 'ATT-INFO=YES' kopiert werden.

Maßnahme
SDF-A Anweisung //ADD-VALUE bzw. //ADD-OPERAND eingeben, um den Wert 'v2' bzw. den Operanden 'op2' zu definieren.

SDA0443 /SEND-MESSAGE (/INTR, /INFORM-PROGRAM FROM OSD-V3) FOLLOWS K2 INTERRUPT : STATEMENT CANCELLED
SDA0443 NACH EINER K2-UNTERBRECHUNG UND EINEM KOMMANDO /SEND-MESSAGE (/INTR, /INFORM-PROGRAM AB OSD-V3) WIRD DIE AKTUELLE ANWEISUNG ABGEBROCHEN

SDA0444 INTERNAL NAME OF APPLICATION DOMAIN '(&00)' FOR '(&01)' DOES NOT EXIST
SDA0444 INTERNER NAME DES ANWENDUNGS-BEREICHS '(&00)' BEI '(&01)' NICHT VORHANDEN

SDA0445 WARNING: LIST SPECIFICATION FOR OPERAND '(&00)' DELETED BECAUSE NO CORRESPONDING VALUE PERMITTED IN LIST
SDA0445 WARNUNG: LISTEN-ANGABE FUER OPERAND '(&00)' GELOESCHT DA KEIN ENTSPRECHENDER WERT IN LISTE ZULAESSIG

SDA0446 WARNING: CURRENT SYNTAX FILE NOT CLOSED PROPERLY WHEN LAST PROCESSED
SDA0446 WARNUNG: AKTUELLE SYNTAX-DATEI BEI LETZTER BEARBEITUNG NICHT ORDNUNGSGEMAESS GESCHLOSSEN

Bedeutung
Bei der letzten Bearbeitung wurde der Programmlauf abgebrochen. Die zuletzt durchgefuehrten Aenderungen sollten geprueft werden. Eine weitere Dateibearbeitung ist moeglich. Der Fehlerindikator wird bei fehlerfreier Beendigung des Programmes durch die Anweisung //END zurueckgesetzt.

SDA0447 FILE NAME '(&00)' SPECIFIED IN //OPEN-SYNTAX-FILE STATEMENT ALREADY EXISTS
SDA0447 IN ANWEISUNG //OPEN-SYNTAX-FILE ANGEGEBENER DATEI-NAME '(&00)' BEREITS VORHANDEN

Maßnahme
Der in der Anweisung angegebene Dateiname (&00) wurde bereits angegeben.

SDA0448 LIST OF SINGLE VALUES NOT PERMITTED FOR 'KEYWORD' DATA TYPE
SDA0448 EINZEL-WERT-LISTE FUER DATENTYP 'KEYWORD' UNZULAESSIG

SDA0449 COPYING OF OBJECTS WITHIN SAME SYNTAX FILE NOT POSSIBLE. STATEMENT REJECTED
SDA0449 KOPIEREN VON OBJEKTEN INNERHALB EINER SYNTAX-DATEI NICHT MOEGLICH. ANWEISUNG ABGEWIESEN

Bedeutung
In der Anweisung //COPY muss eine Syntaxdatei eingegeben werden, die sich von der aktuellen Syntaxdatei unterscheidet.

Maßnahme
Zwei verschiedene Dateinamen als Syntaxdateien angeben.

SDA0450 WARNING: INCORRECT 'LONGEST-LENGTH' FOR FULL-FILENAME OR PARTIAL-FILENAME SPECIFIED
SDA0450 WARNUNG: FEHLERHAFTE 'LONGEST-LENGTH' FUER VOLLQUALIFIZIERTEN DATEINAMEN BZW. TEILDATEINAMEN ANGEGEBEN

Bedeutung
Die LONGEST-LENGTH fuer einen vollqualifizierten Dateinamen ist wie folgt definiert:
- mit CAT-ID=YES und USER-ID=YES, LONGEST-LENGTH=54
- mit CAT-ID=NO und USER-ID=YES, LONGEST-LENGST=48
- mit CAT-ID=YES und USER-ID=NO, LONGEST-LENGTH=44
- mit CAT-ID=NO und USER-ID=NO, LONGEST-LENGTH=38.

Die LONGEST-LENGTH fuer einen Teildateinamen ist wie folgt definiert:
- mit CAT-ID=YES und USER-ID=YES, LONGEST-LENGTH=53
- mit CAT-ID=NO und USER-ID=YES, LONGEST-LENGTH=47
- mit CAT-ID=YES und USER-ID=NO, LONGEST-LENGTH=43
- mit CAT-ID=NO und USER-ID=NO, LONGEST-LENGTH=37.

Maßnahme
Automatische Korrektur, die einen richtigen Wert fuer den Operanden LONGEST-LENGTH einsetzt.

```
SDA0451   WARNING: ALIAS NAME SUPPRESSED BECAUSE DUPLICATE OF MAIN OR STANDARD NAME
SDA0451   WARNUNG: ALIAS-NAME UNTERDRUECKT DA DUPLIKAT VON HAUPT- BZW. STANDARD-NAMEN

SDA0452   MODIFICATION OF 'NON KEYWORD' VALUE INTO 'KEYWORD' NOT PERMITTED
SDA0452   AENDERUNG VON 'NON KEYWORD' IN 'KEYWORD' UNZULAESSIG

SDA0453   DEFINITION OF SDF GLOBALS NOT PERMITTED
SDA0453   ANGABE IN 'SDF-GLOBALS' UNZULAESSIG
```

Bedeutung
Duplikate in Feldnamen (CONTINUE, TEST, EXECUTE,...) sind unzulaessig.

```
SDA0454   OPTION 'OVERWRITE-POSSIBLE=YES' ONLY PERMITTED FOR OPERANDS WITH DEFAULT VALUE
SDA0454   ANGABE 'OVERWRITE-POSSIBLE=YES' NUR ZULAESSIG FUER OPERANDEN MIT STANDARD-WERT

SDA0455   WARNING: OUTPUT AREA TOO SHORT TO CONTAIN POSSIBLE INPUTS
SDA0455   WARNUNG: AUSGABE-FELD FUER MOEGLICHE EINGABEN ZU KLEIN

SDA0456   OUTPUT AREA TOO SHORT TO CONTAIN INPUT VALUE
SDA0456   AUSGABE-FELD FUER EINGEGEBENEN WERT ZU KLEIN

SDA0457   FOR ONE OPERAND ONLY ONE VALUE WITH 'NULL ABBREVIATION' IS PERMITTED
SDA0457   FUER EINEN OPERANDEN IST NUR EIN MIT 'NULL-ABBREVIATION' DEFINIERTER WERT
          ZULAESSIG

SDA0458   NO CURRENT OBJECT PRESENTLY DEFINED
SDA0458   KEIN AKTUELLES OBJEKT ZUR ZEIT DEFINIERT

SDA0459   ERROR IN //COPY STATEMENT
SDA0459   FEHLER IN ANWEISUNG //COPY

SDA0460   VALUE '(&00)' IS DEFAULT VALUE FOR OPERAND '(&01)'. DELETION NOT PERMITTED
SDA0460   WERT '(&00)' IST STANDARD-WERT DES OPERANDEN '(&01)'. LOESCHEN UNZULAESSIG

SDA0461   WARNING: THE HASHING FUNCTION IS ONLY PERMITTED FOR C-STRING VALUES
SDA0461   WARNUNG: HASH-FUNKTION NUR FUER WERTE VOM TYP 'C-STRING' ERLAUBT

SDA0463   LONGEST-LOGICAL-LENGTH INCORRECT
SDA0463   LONGEST-LOGICAL-LENGTH FEHLERHAFT
```

Bedeutung
Die laengste logische Eingabelaenge darf nicht kleiner sein als die Mindestlaenge und nicht groesser als die Maximallaenge.

```
SDA0464   WARNING: RECOVERY OF PARTIAL-FILENAME BY FILENAME WITH WILDCARDS
SDA0464   WARNUNG: ERSETZUNG VON PARTIAL-FILENAME DURCH FILENAME MIT WILDCARDS
```

Bedeutung
Der Datentyp <partial-filename> ist nicht mehr notwendig, da er vollstaendig im Datentyp <filename with-wild> enthalten ist.

```
SDA0469   ITEM CURRENTLY NOT DEFINED IN REFERENCE SYNTAX FILES
SDA0469   EINGABE ZUR ZEIT IN REFERENZ-SYNTAX-DATEIEN NICHT DEFINIERT
```

Bedeutung
Die Eingabe, die aus einem Kommando, einer Anweisung, einem Operanden bzw. einem Wert besteht, ist in den Referenzsyntaxdateien nicht definiert.

```
SDA0470   INTERNAL PROGRAM NAME '(&OO)' DOES NOT EXIST IN CURRENT SYNTAX FILE
SDA0470   INTERNER PROGRAMM-NAME '(&OO)' IN AKTUELLER SYNTAX-DATEI NICHT VORHANDEN

SDA0471   HELP TEXT DOES NOT EXIST FOR SPECIFIED LANGUAGE
SDA0471   HILFE-TEXT FUER ANGEGEBENE SPRACHE NICHT VORHANDEN

SDA0472   DEFINITION OF SYSTEM COMMAND WITH 'IMPLEMENTOR=TPR' NOT PERMITTED IN USER SYNTAX
          FILE
SDA0472   DEFINITION EINES SYSTEM-KOMMANDOS MIT 'IMPLEMENTOR=TPR' IN BENUTZER-SYNTAX-
          DATEI UNZULAESSIG

SDA0474   //EDIT STATEMENT NOT PERMITTED FOR THIS OBJECT
SDA0474   ANWEISUNG //EDIT FUER DIESES OBJEKT UNZULAESSIG
```

Bedeutung
Diese Funktion ist dem Systemverwalter vorbehalten.

```
SDA0475   //COPY STATEMENT NOT PERMITTED FOR THIS OBJECT
SDA0475   ANWEISUNG //COPY FUER DIESES OBJEKT UNZULAESSIG
```

Bedeutung
Ein altes Kommando darf nicht in eine Benutzersyntaxdatei kopiert werden. Eine SDF-Standard-Anweisung darf nicht in eine System- oder Gruppensyntaxdatei kopiert werden.

```
SDA0476   WARNING: FOR GLOBAL INFORMATION, PROGRAMS AND DOMAINS, 'ATTACHED-INFO=ONLY'
          VALUE IS INTERPRETED AS 'ATTACHED-INFO=YES'
SDA0476   WARNUNG: FUER GLOBAL-INFORMATION, PROGRAMME UND ANWENDUNG-BEREICHE WIRD WERT
          'ATTACHED-INFO=ONLY' ALS 'ATTACHED-INFO=YES' INTERPRETIERT

SDA0477   NO VALUE PRESENTLY DEFINED FOR PROCEDURE FILE NAME
SDA0477   ZUR ZEIT KEIN WERT FUER PROZEDUR-DATEINAME DEFINIERT
```

Bedeutung
Der Name der kommandoimplementierenden Prozedur ist nicht definiert:
entweder wurde der Dateiname nicht angegeben oder es fehlen noch einige IMON-Informationen.

```
SDA0478   OBJECT '(&OO)' IS NOT FLAGGED 'REMOVED'. //RESTORE STATEMENT REJECTED
SDA0478   OBJEKT '(&OO)' NICHT ALS 'REMOVED' ANGEZEIGT. ANWEISUNG //RESTORE ABGEWIESEN
```

Bedeutung
Moegliche Ursachen:
- das angegebene Objekt ist nicht als REMOVED angezeigt.
- das angegebene Objekt stimmt mit der Abkuerzung mehrerer Objekte ueberein, die nicht als REMOVED angezeigt sind.

```
SDA0479   THE NAME OF THE INPUT-FILE IS THE SAME AS THE NAME OF THE COMPARE-FILE
SDA0479   GLEICHER NAME BEI INPUT-FILE UND COMPARE-FILE

SDA0480   VALUE '(&00)' DOES NOT MATCH CORRESPONDING SYNTAX TYPE BECAUSE INPUT OF BLANKS
          IS NOT PERMITTED
SDA0480   WERT '(&00)' ENTSPRICHT NICHT ZUGEHOERIGEM SYNTAX-TYP, DA EINGABE VON
          LEERZEICHEN UNZULAESSIG IST

SDA0481   WARNING: 'STRUCTURE-IMPLICIT=YES' NOT PERMITTED FOR OPERAND '(&00)'.
          'STRUCTURE-IMPLICIT' RESET TO 'NO'
SDA0481   WARNUNG: 'STRUCTURE-IMPLICIT=YES' FUER OPERAND '(&00)' UNZULAESSIG. 'STRUCTURE-
          IMPLICIT' AUF 'NO' ZURUECKGESETZT
```

Bedeutung
Die Angabe STRUCTURE-IMPLICIT=YES ist nur bei Strukturoperanden moeglich, die durch ein Schluesselwort bzw. eine Schluesselwortnummer eingefuehrt werden.

```
SDA0482   WARNING: 'ANALYSE-DEFAULT' WAS RESET TO 'NO' FOR OPERAND '(&00)'
SDA0482   WARNUNG: 'ANALYSE-DEFAULT' ZURUECKGESETZT AUF 'NO' FUER OPERAND '(&00)'
```

Bedeutung
- Wegen eines vorangegangenen Fehlers
- Fuer Datentyp 'device'
- Fuer geheimen Wert
- Fuer Standardwert mit Listen -bzw. Struktur-Angabe.

```
SDA0483   'OVERWRITE-POSSIBLE=YES' NOT PERMITTED FOR OPERANDS AND VALUES
SDA0483   'OVERWRITE-POSSIBLE=YES' FUER OPERANDEN UND WERTE UNZULAESSIG

SDA0485   'EXTERNAL-ATTRIBUTES' VALUE ONLY VALID FOR PROGRAMS OR DOMAINS
SDA0485   WERT 'EXTERNAL-ATTRIBUTES' NUR GUELTIG FUER PROGRAMME BZW. BEREICHE

SDA0486   EXTERNAL DOMAIN OR PROGRAM NAME '(&00)' ALREADY ASSIGNED
SDA0486   EXTERNER BEREICHS- BZW. PROGRAMM-NAME '(&00)' BEREITS ZUGEORDNET

SDA0487   //RESTORE STATEMENT IN SYSTEM SYNTAX FILE NOT POSSIBLE
SDA0487   ANWEISUNG //RESTORE IN SYSTEM-SYNTAX-DATEI NICHT MOEGLICH
```

Bedeutung
Ein aus der Systemsyntaxdatei entferntes Objekt ist wirklich geloescht, ein Wiederherstellen ist nicht moeglich.

```
SDA0488   OBJECT '(&00)' NOT REMOVED. //RESTORE STATEMENT REJECTED
SDA0488   OBJEKT '(&00)' NICHT ENTFERNT. ANWEISUNG //RESTORE ABGEWIESEN
```

Bedeutung
Das angegebene Objekt ist nicht entfernt. Es entspricht der Abkuerzung bereits vorhandener Objekte, die nicht entfernt sind.

SDA0489 OBJECT '(&OO)' CANNOT BE RESTORED IN USER SYNTAX FILE
SDA0489 OBJEKT '(&OO)' KANN IN BENUTZER-SYNTAX-DATEI NICHT WIEDERHERGESTELLT WERDEN

Bedeutung
Das in der Systemsyntaxdatei definierte Objekt wurde nach der Eroeffnung der Gruppensyntaxdatei entfernt.

Maßnahme
Die Gruppensyntaxdatei eroeffnen, um dieses Objekt wiederherzustellen.

SDA0490 SYNTAX FILE '(&OO)' WAS NOT CLOSED CORRECTLY IN PREVIOUS PROCESSING. FILE IS CORRUPTED AND CAN NO LONGER BE USED
SDA0490 SYNTAX-DATEI '(&OO)' WURDE NICHT ORDNUNGSGEMAESS GESCHLOSSEN. DATEI IST ZERSTOERT UND NICHT MEHR ZU VERWENDEN

SDA0491 OVERWRITING OF AN OBJECT DEFINED IN HIGHER LEVEL OF HIERARCHY NOT POSSIBLE
SDA0491 UEBERSCHREIBEN DES OBJEKTS IST NICHT ERLAUBT, WENN DAS OBJEKT IN EINER SYNTAXDATEI HOEHERER HIERARCHIE DEFINIERT IST

SDA0492 WARNING: ERROR ON 'STXIT' INITIALISATION. /INTR COMMAND REJECTED
SDA0492 WARNUNG: FEHLER IN 'STXIT'-VORBEREITUNG. KOMMANDO /INTR ABGEWIESEN

SDA0493 ERROR ON STXIT INITIALISATION : ABEND/K2 CAN NOT BE USED. DO NOT INTERRUPT THE PROGRAM AT THIS POINT
SDA0493 FEHLER IN STXIT VORBEREITUNG : ABEND/K2 DARF NICHT VERWENDET WERDEN. PROGRAMMUNTERBRECHUNG AN DIESEM PUNKT NICHT MOEGLICH

SDA0494 THE OVERWRITE OF COMMON STATEMENTS IS ONLY POSSIBLE FROM SDF-U PROGRAM
SDA0494 UEBERSCHREIBEN VON SDF-STANDARD-ANWEISUNGEN NUR MIT SDF-U MOEGLICH

SDA0495 THE TPR-COMMAND '(&OO)' DOESN'T EXIST OR IS NOT A TPR-COMMAND
SDA0495 TPR-KOMMANDO '(&OO)' EXISTIERT NICHT ODER IST KEIN TPR-KOMMANDO

SDA0496 WARNING: THE RESULT-INTERNAL-NAME MAY NOT BE DEFINED WITH IMPLEMENTOR=*BY-TPR
SDA0496 WARNUNG: RESULT-INTERNAL-NAME KANN NICHT ZUSAMMEN MIT IMPLEMENTOR=*BY-TPR DEFINIERT WERDEN

SDA0497 NO VALUE PRESENTLY DEFINED FOR TPR COMMAND NAME
SDA0497 NOCH KEIN WERT FUER TPR-KOMMANDO-NAME DEFINIERT

SDA0499 WARNING: THERE IS NO OBJECT CORRESPONDING TO THE SPECIFIED INPUT
SDA0499 WARNUNG: KEIN OBJEKT PASST ZUR EINGABE

SDA0507 FILE '(&OO)' IS NOT A SYNTAX FILE OR FILE TYPE INVALID
SDA0507 DATEI '(&OO)' IST KEINE SYNTAX-DATEI BZW. DATEITYP FEHLERHAFT

SDA0508 SYNTAX FILE '(&OO)' NOT CLOSED CORRECTLY
SDA0508 SYNTAX-DATEI '(&OO)' NICHT ORDNUNGSGEMAESS GESCHLOSSEN

SDA0509 '(&OO)' IS NOT A NEW KERNEL SYNTAX FILE. INPUT IGNORED
SDA0509 '(&OO)' IST KEINE NEUE KERNEL-SYNTAX-DATEI. EINGABE IGNORIERT

SDA0510 VERSION VALUE '(&00)' IGNORED BECAUSE '(&01)' IS NOT A COMPONENT SYNTAX FILE. PROCESSING CONTINUES
SDA0510 VERSIONS-WERT '(&00)' IGNORIERT DA '(&01)' KEINE KOMPONENTEN-SYNTAX-DATEI IST. BEARBEITUNG FORTGESETZT

Bedeutung
Die in der Anweisung //OPEN-SYNTAX-FILE angegebene Version wurde nicht gefunden. Die angegebene Datei ist keine Komponenten-Syntaxdatei.

SDA0511 TERMINATE THE PROGRAM BY '//END' ELSE THE LAST OBJECT MAY BE INCOMPLETE IN THE SYNTAX FILE
SDA0511 PROGRAM MIT '//END' BEENDEN, SONST IST LETZTES OBJEKT IN SYNTAX-DATEI UNVOLLSTAENDIG.

Bedeutung
Programm wieder aufrufen und zuletzt bearbeitetes Kommando bzw. Anweisung mit //CLOSE-CMD-OR-STMT abschliessen oder Program mit //END beenden. Andernfalls weist SDF die Syntaxdatei ab.

SDA0512 THE PROCESSED SYNTAX FILE IS IN AN INCONSISTENT STATE
SDA0512 INKONSISTENTER ZUSTAND DER SYNTAXDATEI

Bedeutung
Die Syntaxdatei ist inkonsistent und wird von SDF abgewiesen. Syntaxdatei erneut im UP-DATE-Modus oeffnen und zuletzt bearbeitete Objekte korrigieren.

SDA0571 TABLE OF COMMAND NAMES DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0571 TABELLE DER KOMMANDO-NAMEN IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0572 TABLE OF NAMES OF APPLICATION DOMAINS DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0572 TABELLE DER NAMEN DES ANWENDUNGS-BEREICHS IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0573 TABLE OF PROGRAM NAMES DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0573 TABELLE DER PROGRAMM-NAMEN IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0574 TABLE OF ALL PRIVILEGES DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0574 TABELLE ALLER PRIVILEGIEN IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0575 STATEMENT FOR PROGRAM '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0575 ANWEISUNG FUER PROGRAMM '(&00)' IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0576 MCLP ERROR '(&00)'. COMMAND NOT PROCESSED
SDA0576 MCLP-FEHLER '(&00)'. KOMMANDO NICHT AUSGEFUEHRT

Bedeutung
Naehere Information ueber den MCLP-Fehler kann der Beschreibung des Makros CMD im BS2000-Handbuch 'Makroaufrufe an den Ablaufteil' entnommen werden.
MCLP: Macro Command Language Processor.

SDA0580 NAME OF APPLICATION DOMAIN '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0580 NAME DES ANWENDUNGS-BEREICHS '(&00)' IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

```
SDA0581   GLOBAL INFORMATION DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0581   GLOBAL-INFORMATION IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0582   COMMAND '(&OO)' DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0582   KOMMANDO '(&OO)' IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0583   STATEMENT '(&OO)' DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0583   ANWEISUNG '(&OO)' IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0584   V-RECORD DOES NOT EXIST IN SYNTAX FILE
SDA0584   V-SATZ IN SYNTAX-DATEI NICHT VORHANDEN

SDA0585   DATA TYPE '(&OO)' DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0585   DATENTYP '(&OO)' IN AKTIVER SYNTAX-DATEI NICHT VORHANDEN

SDA0586   PROGRAM '(&OO)' DOES NOT EXIST IN ACTIVE SYNTAX FILE
SDA0586   PROGRAMM '(&OO)' IN AKTIVER SYNTAXDATEI NICHT VORHANDEN

SDA0600   INTERNAL NUMBER '(&OO)' ALREADY ASSIGNED TO ANOTHER PRIVILEGE
SDA0600   INTERNE NUMMER '(&OO)' BEREITS ANDEREM PRIVILEG ZUGEORDNET
```

Bedeutung
Jedes Privileg wird intern identifiziert durch die Stellung der entsprechenden Nummer in der 64-Bit-Kette, welche die Privilegien repraesentiert. Daher muss die Stellung der Nummer fuer jedes Privileg unterschiedlich sein.

Maßnahme
Dem Produkt zugeordnete, interne Nummer aendern.

```
SDA0601   INCONSISTENCY IN PRIVILEGE DEFINITION
SDA0601   UNVEREINBARKEIT IN PRIVILEG-DEFINITION
```

Bedeutung
In der Definition eines Kommandos- bzw. einer Anweisung ist ein Knoten (Operand oder Wert) mit mehr Privilegien definiert als sein 'Vaterknoten'. Die Privilegien fuer die Operanden und Werte der niedrigsten Stufe koennen nicht mehr weiter geaendert werden, da einer dieser Operanden bzw. Werte kein Privileg mehr hat.

Maßnahme
Die jedem Kommandoknoten (Kommando, Operand oder Wert) zugeordneten Privilegien ueberpruefen und Inkonsistenzen beseitigen.

```
SDA0602   WARNING: INCONSISTENCY IN PRIVILEGE DEFINITION FOR OPERAND '(&OO)'
SDA0602   WARNUNG: INKONSISTENZ IN PRIVILEG-DEFINITION FUER OPERAND '(&OO)'
```

Bedeutung
Es muss fuer jeden Prozess die Moeglichkeit bestehen, einen Wert fuer den Operanden einzusetzen, unabhaengig von dem (den) ihm zugeordneten Privileg(ien). Deshalb muessen Standardwert und Operandenwert das (die) gleiche(n) Privileg(ien) besitzen.
Hat der Operand keinen Standardwert, muss er die gleichen Privilegien wie sein 'Vaterknoten' (Kommando- bzw. Wertknoten) besitzen. Ausserdem muss fuer jedes dem Operanden zugeordnete Privileg mindestens ein Wert mit diesem Privileg vorhanden sein.

SDA0603 //REMOVE STATEMENT NOT PERMITTED FOR THIS OBJECT
SDA0603 ANWEISUNG //REMOVE FUER DIESES OBJEKT UNZULAESSIG

Bedeutung
Entfernen eines Privilegs ist nur in der Systemkerndatei zulaessig. Diese Funktion ist dem Privilegienverwalter vorbehalten.

SDA0605 MINIMAL-ABBREVIATION '(&00)' NOT ABBREVIATION OF MAIN NAME '(&01)'
SDA0605 MINIMAL-ABKUERZUNG '(&00)' KEINE ABKUERZUNG DES HAUPTNAMENS '(&01)'

Bedeutung
Die Minimal-Abkuerzung (&00) muss eine Abkuerzung des Hauptnamens (&01) sein, der durch den Operanden NAME definiert ist.

Maßnahme
Eindeutige Abkuerzung verwenden.

SDA0606 WARNING: NAME '(&00)' SHORTER THAN REQUIRED MINIMAL-ABBREVIATION '(&01)'. INPUT IGNORED
SDA0606 WARNUNG: NAME '(&00)' KUERZER ALS ERFORDERLICHE MINIMAL-ABKUERZUNG '(&01)'. EINGABE IGNORIERT

Bedeutung
Eine kuerzere Eingabe als die Minimalabkuerzung kann eindeutig sein, wird aber aus Sicherheitsgruenden nicht ausgefuehrt.

Maßnahme
Eindeutige Abkuerzung verwenden.

SDA0607 MINIMAL-ABBREVIATION '(&00)' AMBIGUOUS WITH REGARD TO AN OPERAND NAME OR VALUE
SDA0607 MINIMAL-ABKUERZUNG '(&00)' MEHRDEUTIG BEZUEGLICH EINES OPERANDEN-NAMENS BZW. WERTES

Bedeutung
Die Minimalabkuerzung ist mehrdeutig bezueglich des Haupt-, Alias- bzw. Standardnamens eines anderen Operanden bzw. Wertes der gleichen Stufe.

Maßnahme
Eindeutige Abkuerzung verwenden.

SDA0608 WARNING: 'NULL-ABBREVIATION' RESET TO 'NO' BECAUSE 'MINIMAL-ABBREVIATION' IS GIVEN
SDA0608 WARNUNG: WEGEN ANGABE VON 'MINIMAL-ABBREVIATION' WURDE 'NULL-ABBREVIATION' AUF 'NO' ZURUECKGESETZT

Bedeutung
NULL-ABBREVIATION=YES kann nicht verwendet werden, wenn vorher eine Minimal-Abkuerzung angegeben wurde.

SDA0609 WARNING: 'PRIVILEGE DEFINITION' RESET BECAUSE PRIVILEGES ASSOCIATED WITH OBJECT TO BE COPIED DO NOT EXIST IN HIERARCHY
SDA0609 WARNUNG: 'PRIVILEG-DEFINITION' ZURUECKGESETZT, DA PRIVILEGIEN DES ZU KOPIERENDEN OBJEKTES IN HIERARCHIE NICHT VORHANDEN

Bedeutung
Bei Kommando- oder Anweisungsdefinition muessen Operanden und Werte dieselben Privilegien wie der Kommando- oder Anweisungsknoten haben.

SDA0610 WARNING: 'PRIVILEGE DEFINITION' RESET BECAUSE THE OBJECT TO BE COPIED IS DEFINED WITH MORE PRIVILEGES THAN ITS 'FATHER' NODE
SDA0610 WARNUNG: 'PRIVILEG-DEFINITION' ZURUECKGESETZT, DA ZU KOPIERENDES OBJEKT MIT MEHR PRIVILEGIEN DEFINIERT IST ALS SEIN 'VATERKNOTEN'

Bedeutung
Die Privilegiendefinition des Objektes und der Operanden und Werte der untersten Ebene wurde auf die Privilegien des 'Vaterknotens' zurueckgesetzt.

8.2 SDF-SIM-Meldungen

```
SDS0001    SDF-SIM VERSION '(&00)' STARTED
SDS0001    SDF-SIM VERSION '(&00)' GESTARTET

SDS0002    SDF-SIM TERMINATED NORMALLY
SDS0002    SDF-SIM NORMAL BEENDET

SDS0003    SDF-SIM TERMINATED ABNORMALLY
SDS0003    SDF-SIM ABNORMAL BEENDET

SDS0004    INVALID SYNTAX FILE FOR OPERAND '(&00)'
SDS0004    UNGUELTIGE SYNTAX-DATEI FUER OPERAND '(&00)'

SDS0005    '(&00)' SYNTAX FILE '(&01)' ACTIVATED
SDS0005    '(&00)' SYNTAX DATEI '(&01)' AKTIVIERT

SDS0006    PROGRAM '(&00)' IS NOT DEFINED IN THE ACTIVATED SYNTAX FILES
SDS0006    PROGRAM '(&00)' NICHT DEFINIERT IN AKTIVIERTEN SYNTAX-DATEIEN

SDS0007    ERROR '(&00)' RETURNED BY '(&01)' CALL
SDS0007    FEHLER '(&00)' ZURUECKGEGEBEN VON '(&01)' AUFRUF

SDS0008    DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
SDS0008    SOLL DER UEBERGABE-BEREICH ANGEZEIGT WERDEN? (NO/LONG/SHORT)

SDS0009    ENTER THE NAME OF THE OML TO USE FOR ENTRY '(&00)' OR *NO
SDS0009    BITTE NAME DER OML MIT DEM ENTRY '(&00)' ODER *NO ANGEBEN
```

Bedeutung
Der Benutzer hat EXECUTION=*YES angeben (in DEFINE-ENVIRONMENT) und SDF-SIM hat nicht den richtigen ENTRY fuer ein Kommando gefunden. SDF-SIM benoetigt den Namen der OML mit dem richtigen ENTRY, um den ENTRY zum Simulations-Programm zu binden.

```
SDS0010    INVALID OML NAME
SDS0010    UNGUELTIGER OML-NAME

SDS0011    SDF INTERNAL ERROR: ERROR '(&00)', RETURNED BY MACRO '(&01)', 'SERSLOG' LEVEL
SDS0011    INTERNER SDF-FEHLER: FEHLER '(&00)' VON SCHNITTSTELLE NUMMER '(&01)'
           ZURUECKGEGEBEN, 'SERSLOG' LEVEL

SDS0012    DO YOU WANT TO DISPLAY STACK? REPLY: YES/NO
SDS0012    SOLL DER STACK ANGEZEIGT WERDEN? (YES/NO)
```

Bedeutung
Ein SDF-interner Fehler ist aufgetreten. Das Anzeigen der Stacks (ABNORM-Routine) kann die Fehlerdiagnose erleichtern.

```
SDS0013    SDF INTERNAL ERROR: ERROR '(&00)', RETURNED BY INTERFACE NUMBER '(&01)',
           'TERMINATE' LEVEL
SDS0013    INTERNER SDF-FEHLER: FEHLER '(&00)' VON SCHNITTSTELLE NUMMER '(&01)'
           ZURUECKGEGEBEN, 'TERMINATE' LEVEL
```

9 Anhang

9.1 Änderungen der SDF-Programmschnittstelle

Mit SDF V4.1 wurde das Layout des normierten Übergabebereiches geändert. Aus diesem Grund wurde der Makro CMDSTRUC zur Generierung des Übergabebereiches durch den Makro CMDTA ersetzt. Außerdem wurden die Makros RDSTMT, TRSTMT und CORSTMT zur Behandlung von Anweisungen durch die neuen Makros CMDRST, CMDTST und CMDCST ersetzt.

Das bis SDF V4.0 verwendete Format des normierten Übergabebereiches sowie die Makros CMDSTRUC, RDSTMT, TRSTMT und CORSTMT werden aus Kompatibilitätsgründen weiterhin unterstützt. Sie sollten jedoch in neuen Programmen nicht mehr verwendet werden.

9.1.1 Format des normierten Übergabebereiches bis SDF V4.0

Der normierte Übergabebereich beginnt auf Wortgrenze. In Byte 0 bis 19 liegt das Kopffeld. Es enthält u.a. den internen Anweisungsnamen (siehe ADD-STMT ...,INTERNAL-NAME=...). Ab Byte 20 beginnt das Array für anweisungsglobale Operanden, d.h. für Operanden, die mit RESULT-OPERAND-LEVEL=1 definiert sind (siehe ADD-OPERAND). In ihm steht für jeden dieser Operanden eine sechs Byte lange Beschreibung. Die Operandenbeschreibungen sind in der Reihenfolge angeordnet, die sich aus den in der Anweisungsdefinition festgelegten Operandenpositionen ergibt (siehe ADD-OPERAND ..., RESULT-OPERAND-NAME=*POSITION(POSITION=<integer>)). Jede Operandenbeschreibung enthält u.a. die Absolutadresse, unter der der zugehörige Operandenwert bzw. die Beschreibung der zugehörigen Liste oder nichtlinearisierten Struktur im Übergabebereich abgelegt ist. In der Beschreibung einer nichtlinearisierten Struktur steht ein Operanden-Array, das die in der Struktur enthaltenen Operanden beschreibt. Es ist genauso aufgebaut, wie das Array für die anweisungsglobalen Operanden.

Im einfachsten Fall hat ein Operand lediglich einen einfachen Wert (Bild 8 ohne Standardheader, Seite 373).

Leitet der Operandenwert eine Struktur ein (Bild 9 ohne Standardheader, Seite 374), so gibt es für ihn eine Strukturbeschreibung. Diese enthält ein Operanden-Array mit Beschreibungen für alle Operanden der Struktur sowie für die Operanden aus linearisierten Unterstrukturen. Die Operandenbeschreibungen sind in der Reihenfolge angeordnet, die sich aus den in der Anweisungsdefinition festgelegten strukturrelativen Operandenpositionen ergibt (siehe ADD-OPERAND ...,RESULT-OPERAND-NAME=*POSITION(POSITION=<integer>)). Die zu den Operanden der Struktur gehörenden Operandenwerte können eine weitere Struktur einleiten und/oder aus einer Liste von Werten bestehen.

Von Operanden, die mit ADD-OPERAND...,LIST-POSSIBLE=*YES(...,FORM=*NORMAL) definiert sind, werden die Werte in der in Bild 10 (Seite 375, ohne Standardheader) gezeigten Form übergeben. An einem Listenelement kann eine Struktur hängen. In diesem Fall ist „Wert des Listenelements“ eine Strukturbeschreibung.

Kopffeld des normierten Übergabebereichs

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	**fehlerhafte Anweisung zurück an SDF**	**Default-Werte an SDF**
0 bis 1	Länge des Übergabebereichs (maximal 65536 Byte)	Programm	unverändert	Programm
2 bis 9	interner Anweisungsname	SDF	unverändert	Programm
10 bis 17	reserviert	–	–	–
18 bis 19	Anzahl der Positionen im Operanden-Array	SDF	unverändert	Programm

Der interne Anweisungsname und die Anzahl der Positionen in dem Operanden-Array sind in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT...,INTERNAL-NAME=...,MAX-STRUC-OPERAND=...).

Operandenbeschreibung

Das Operanden-Array und die in ihm stehenden Beschreibungen für die Operanden einer Struktur sind genau so aufgebaut wie bei den anweisungsglobalen Operanden.

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0 bis 1 0 1	Wertbeschreibung Zusatzinformation Typbeschreibung	siehe unten SDF	siehe unten unverändert	siehe unten Programm [1]
2 bis 5	Absolutadresse (unaligned abgelegt) des zu dem Operanden gehörenden Werts bzw. bei Strukturen oder Listen der weiteren Beschreibung	SDF	unverändert	Programm [1]

[1] Angabe nur für Operanden, zu denen umzusetzende Operandenwerte gehören, d.h. wenn Bit 0 der Zusatzinformation gesetzt ist.

Zusatzinformation

Die Zusatzinformation steht im ersten Byte der Wertbeschreibung. Die folgenden Angaben über die Zusatzinformation gelten unabghängig davon, ob die Zusatzinformation in einer Operandenbeschreibung, im Kopffeld einer Strukturbeschreibung, in der Beschreibung eines Listenelements oder in einer OR-Listenbeschreibung vorkommt.

Bit	Wert	Inhalt	Feldinhalt kommt im Fall ... von		
			analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0	0	Wert ist nicht vorhanden	SDF [1]	unverändert	Programm [2]
0	1	Wert ist vorhanden	SDF	unverändert	Programm [2]
1	0	Wert ist änderbar	–	Programm [3]	–
1	1	Wert ist nicht änderbar	–	Programm [3]	–
2	0	Wert ist nicht fehlerhaft	–	Programm	–
2	1	Wert ist fehlerhaft	–	Programm	–
3	0	Wert soll nicht als Default eingesetzt werden	–	–	Programm
3	1	Wert soll als Default eingesetzt werden	–	–	Programm
4 bis 7	–	reserviert	–	–	Programm

[1] z.B. Werte zu Operanden, die mit ADD-OPERAND...,PRESENCE=*EXTERNAL-ONLY definiert sind, oder Werte in nicht angesprochenen Strukturen.

[2] 0 für Operanden, zu denen weder umzusetzende Operandenwerte gehören noch Strukturen, die Operanden mit umzusetzenden Werten enthalten.
1 für Operanden, zu denen entweder umzusetzende Operandenwerte gehören oder Strukturen, die Operanden mit umzusetzenden Werten enthalten.

[3] Alle Listenwerte, die nach dem ersten änderbaren Listenwert kommen, betrachtet SDF unabhängig davon, wie Bit 1 gesetzt ist, als änderbar. Dadurch werden schon bearbeitete Listenelemente gegen Überschreiben geschützt.

Typbeschreibung

Die Typbeschreibung steht im zweiten Byte der Wertbeschreibung. Die Angaben über die Typbeschreibung gelten unabhängig davon, ob die Typbeschreibung in einer Operandenbeschreibung, im Kopffeld einer Strukturbeschreibung, in der Beschreibung eines Listenelements oder in einer OR-Listenbeschreibung vorkommt.

Strukturbeschreibungen können durch ein Programm eingegeben werden, um eigene Default-Werte anzugeben. Die Eingabe der Default-Werte ist möglich:

– im internen Format (wie beim Operanden OUTPUT der SDF-A-Anweisung ADD-VALUE) oder

– im externen Format als Zeichenfolge analog der Operandenbeschreibung. Dabei muss der Hilfsdatentyp <input-text> benutzt werden. Der Wert wird dann so analysiert, als ob er über die Benutzerschnittstelle eingegeben worden wäre.

Wert (dezimal)	Bedeutung
1	Kommandorest (Command-Rest)
2	Ganzzahl (Integer)
4	X-Zeichenkette (X-String)
5	C-Zeichenkette (C-String)
6	Name (Name)
7	alphanumerischer Name (Alphanumeric-Name)
8	strukturierter Name (Structured-Name)
9	Marke (Label)
11	Dateiname (Filename)
12	teilqualifizierter Dateiname (Partial-Filename)
13	Uhrzeit (Time)
14	Datum (Date)
15	zusammengesetzter Name (Composed-Name)

Fortsetzung ➠

Wert (dezimal)	Bedeutung
16	Text (Text)
17	Katalog-Kennung (Cat-Id)
18	Eingabe-Text (Input-Text)
19	Struktur
20	Liste
21	OR-Liste
22	Schlüsselwort (Keyword)
23	reserviert für internen Gebrauch
24	VSN
25	X-Text (X-Text)
26	Festpunktzahl (Fixed)
27	Gerät (Device)
28	Produkt-Version (Product-Version)
29	POSIX-Pfadname (Posix-Pathname)
35	POSIX-Dateiname (Posix-Filename)

Kopffeld einer Strukturbeschreibung

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0 bis 1	Anzahl der Positionen im Operanden-Array	SDF	unverändert	Programm
2 bis 3	Wertbeschreibung für den struktureinleitenden Operandenwert			
2	Zusatzinformation	–	siehe Seite 607	siehe Seite 607
3	Typbeschreibung	SDF	unverändert	Programm
4 bis 7	Absolutadresse (unaligned abgelegt) des struktureinleitenden Operandenwerts	SDF	unverändert	Programm

Die Anzahl der Positionen in dem Operanden-Array ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-VALUE...,STRUCTURE=YES(..,MAX-STRUC-OPERAND=...).

Das zu der Struktur gehörende Operanden-Array beginnt unmittelbar hinter dem Kopffeld. Es ist genau so aufgebaut wie das für anweisungsglobale Operanden.

Listenelement

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	**fehlerhafte Anweisung zurück an SDF**	**Default-Werte an SDF**
0 bis 1 0 1	Wertbeschreibung Zusatzinformation Typbeschreibung	 siehe oben SDF	 siehe Seite 607 unverändert	 siehe Seite 607 Programm [1)]
2 bis 5	Absolutadresse (unaligned abgelegt) des zu dem Listenelement gehörenden Werts bzw. bei Strukturen der weiteren Beschreibung	SDF	unverändert	Programm [1)]
6 bis 9	Absolutadresse (unaligned abgelegt) des nächsten Listenelements	SDF	unverändert	Programm [1)]

[1)] Angabe nur für Operanden, zu denen umzusetzende Operandenwerte gehören, d.h. wenn Bit 0 der Zusatzinformation gesetzt ist.

Im letzten Element der Liste hat die „Absolutadresse des nächsten Listenelements“ den Wert 0.

Eine OR-Liste besteht nur aus einem Element. Bei diesem Element entfällt die „Absolutadresse des nächsten Listenelements“.

Für einen Operanden, der mit LIST-POSSIBLE=*YES(FORM=*NORMAL) definiert ist, muss die Anzahl der Listenelemente begrenzt werden (LIMIT=...), damit es nicht zum Überlauf des normierten Übergabebereiches kommt. Die Größe einer Liste im normierten Übergabebereich kann nach folgender Formel berechnet werden:

$$n * (10 + 2 + l)$$

wobei: n die Anzahl der Listenelemente und

l die Länge eines einzelnen Listenelementes (aufgerundet auf ein Vielfaches von 2) ist.

Beispiel:

```
ADD-OPERAND ... LIST-POSSIBLE=*YES(LIMIT=100,FORM=*NORMAL)
  ADD-VALUE *NAME(1,8)
```

Eine so definierte Liste kann im normierten Übergabebereich bis zu 2000 Byte belegen (100 * (10 + 2 + 8) = 2000).

Ablage der Werte

Byte	Inhalt	Feldinhalt kommt im Fall ... von		
		analysierte Anweisung an Programm	fehlerhafte Anweisung zurück an SDF	Default-Werte an SDF
0 bis 1	Längenangabe	SDF	unverändert	Programm
2 bis ...	Wert	SDF	unverändert	Programm

Wie die Werte übergeben werden, hängt ab von der Definition in der Syntaxdatei (siehe ADD-VALUE...,OUTPUT=*NORMAL(...). Dabei gelten folgende Besonderheiten:

– Ein Wert, der mit ADD-VALUE TYPE=*INTEGER(...,OUT-FORM=*BINARY) definiert ist, wird als 4-Byte-String mit Vorzeichen abgelegt.

– Ein Wert, der mit ADD-VALUE TYPE=*TIME definiert ist, wird als 4-Byte-String mit 2 Bytes (binär) für Stunden und je ein Byte für Minuten und Sekunden abgelegt.

9.1.2 CMDSTRUC
Übergabebereich für eine analysierte Anweisung generieren

Der Makro CMDSTRUC generiert eine DSECT. Diese beschreibt den normierten Übergabebereich bis SDF V4.0, der für folgende drei Zwecke benötigt wird:

1. SDF übergibt an das Programm eine analysierte Anweisung (siehe CORSTMT, RDSTMT und TRSTMT).
2. Das Programm gibt eine semantisch fehlerhafte Anweisung an SDF zurück (siehe CORSTMT)
3. Das Programm übergibt an SDF Werte, die eingegebene Operandenwerte ersetzen sollen (siehe RDSTMT und TRSTMT).

Der normierte Übergabebereich ist im Detail auf den Seiten 605ff beschrieben.

Operation	Operanden
CMDSTRUC	[P = CMD / prefix]

P = CMD / prefix
bestimmt eine Zeichenfolge, die mit dem Anfang aller Namen in der DSECT verkettet wird. Sie darf maximal drei Zeichen lang sein. Standardmäßig wird die Zeichenfolge „CMD“ genommen.

9.1.3 CORSTMT
Semantikfehlerdialog anstoßen

Der Makro CORSTMT bewirkt, dass SDF mit dem Benutzer einen Dialog führt, in dem dieser Semantikfehler in einer Anweisung korrigiert. SDF hat die Anweisung unmittelbar zuvor analysiert und als syntaktisch richtig an das Programm übergeben. Verdeckt eingegebene Operandenwerte muss der Benutzer während des Korrekturprozesses noch einmal eingeben.

Voraussetzungen für den Semantikfehlerdialog sind:

- das Programm läuft in einer interaktiven Task und bei der Syntaxanalyse war ein Fehlerdialog zugelassen, d.h.:
  - temporär oder permanent geführter Dialog musste eingestellt sein
  - in Prozeduren musste die SDF-Option PROCEDURE-DIALOGUE=*YES eingestellt sein
  - wenn CORSTMT nach TRSTMT aufgerufen wird, musste bei TRSTMT DIALOG=ERROR eingestellt sein.
- es steht die gleiche Syntaxdatei wie bei der ersten Analyse der Anweisung zur Verfügung (kein zwischenzeitlicher Wechsel der Syntaxdatei).

Sind diese Voraussetzungen nicht erfüllt, lehnt SDF den Dialog ab (Fehlercode X'20').

Operation	**Operanden**
CORSTMT	INOUT = addr / (r1) ,MESSAGE = addr / (r3) [,DEFAULT = <u>*NO</u> / (addr,...)] [,INVAR = <u>*NO</u> / addr / (r)] [,CALLID = <u>*NO</u> / addr / (r7)] [,CCSNAME = <u>*NO</u> / *EXTEND / name] [,MF = { L / (E,(1)) / (E,opadr) }]

INOUT = addr / (r1)
Adresse des normierten Übergabebereichs bzw. Register, das diese Adresse enthält. Der Bereich muss auf Wortgrenze beginnen. In ihm steht das zuvor von SDF an das Programm übergebene Analyseergebnis der fehlerhaften Anweisung. Das Programm hat die von ihm als fehlerhaft erkannten Operandenwerte gekennzeichnet. Änderungen von

Eingabewerten durch das Programm übernimmt SDF nicht. Nach dem Fehlerdialog und der erneuten Analyse legt SDF das geänderte Analyseergebnis wieder in diesem Bereich ab (siehe Abschnitt „Format des normierten Übergabebereiches bis SDF V4.0“ auf Seite 605).

MESSAGE = addr / (r3)
Adresse des auszugebenden Textes für den Fehlerdialog bzw. Register, das diese Adresse enthält. Dieser wird im geführten Dialog in das Anweisungsmenü integriert. Der Text wird als Satz variabler Länge erwartet und darf maximal vier Zeilen auf der Datensichtstation belegen. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden. Dieser Bereich muss auf Halbwortgrenze ausgerichtet sein.

DEFAULT =
Bestimmt, ob SDF folgende Werte durch vom Programm dynamisch erzeugte Werte ersetzt:
– eingegebene Operandenwerte oder
– Default-Werte der Operanden

In der Syntaxdatei müssen die Operanden bzw. Operandenwerte entsprechend definiert sein (siehe ADD-OPERAND ...,OVERWRITE-POSSIBLE=*YES,... bzw. ADD-VALUE..., VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). Der vom Programm erzeugte Wert muss gültiger Operandenwert sein.
Im geführten Dialog zeigt SDF die vom Programm erzeugten Werte im Fragebogen.

Beispiel:

SDF-A ersetzt in den eingegebenen MODIFY-Anweisungen den Wert UNCHANGED durch den aktuellen Wert. Stehen die Operanden, die mit einem Default-Wert versehen werden sollen, in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten:
Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit Operanden hängt, die mit einem Default-Wert versehen werden sollen. Auf der anderen Seite gibt der Anwender ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit Operanden hängt, die mit einem Default-Wert versehen werden sollen. SDF versucht zunächst, die vom Anwender eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen. Ist über den struktureinleitenden Wert keine eindeutige Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Anwender den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

***NO**
die angegebenen Operandenwerte werden nicht durch vom Programm dynamisch erzeugte Werte ersetzt

(addr,...)
In einer oder in mehreren der möglichen Anweisungen sind angegebene Operandenwerte durch vom Programm dynamisch erzeugte Werte zu ersetzen. Unter den angegebenen Adressen (addr,...) stehen in dem Programm die Umsetzbeschreibungen für diese Anweisungen. Die Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche angegebenen Operandenwerte in welche Werte umzusetzen sind.
Die Bereiche für die Umsetzbeschreibungen, die für die Standardwerte des Programms übergeben werden, müssen auf Wortgrenze ausgerichtet sein. Dies gilt ebenso für den Ausgabebereich der Makros (OUTPUT-Operand).

INVAR =
Legt fest, ob die INVARIANT-INPUT-Form der Anweisung abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task zu dieser Zeit erlaubt sind. Die INVARIANT-INPUT-Form ist damit die größtmögliche Eingabeform für eine Anweisung, die für einen Benutzer mit bestimmten Privilegien und im gewählten Dialogmodus zulässig ist. Im Gegensatz zur Protokollierungsform LOGGING=*INVARIANT-FORM (siehe MODIFY-SDF-OPTIONS) werden Kennwörter und geheime Operanden jedoch *nicht* ausgeblendet.

***NO**
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

addr / (r)
Gibt die Adresse eines Puffers an, in den SDF die INVARIANT-INPUT-FORM der Anweisung schreibt. Der Puffer muss auf Wortgrenze ausgerichtet sein und das erste Halbwort muss die Länge des Puffers enthalten. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab. Der Puffer hat dann folgenden Inhalt:

1. HW	2. HW	3. HW (Halbwort)	
buflen	reclen	filler	invariant-input

buflen: Länge des Puffers
reclen: Länge des Satzes, den SDF schreibt
filler: Füllzeichen
invariant-input: INVARIANT-INPUT-Form der Anweisung, beginnt beim 7. Byte.

CALLID =
Bezieht sich auf einen Kontext (=Syntaxdateihierarchie), der durch einen OPNCALL-Makro eröffnet wurde. Der Name der Syntaxdateihierarchie (callid) muss den 4 Byte langen Wert haben, der von SDF an das Feld zurückgegeben wird, welches durch den Operanden CALLID im Open-Context-Makro bezeichnet wurde.
Diese Funktion bezieht sich auf den Gebrauch der Makros OPNCALL und CLSCALL.

***NO**
Die aktuelle Syntaxdateihierarchie (Kontext) der Task wird für die Analyse der Anweisung verwendet.

addr / (r7)
Adresse des Aufrufprüfungsfeldes bzw. Register, das diese Adresse enthält. Der Bereich muss auf Wortgrenze ausgerichtet sein.

CCSNAME =
Gibt den Namen des Zeichensatzes an, der für den Korrekturdialog auf 8-bit-Terminals und für die Konvertierung von Klein- in Großbuchstaben verwendet wird. Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Jeder codierte Zeichensatz wird durch seinen Namen (Coded Character Set Name, CCSN) bestimmt (siehe Handbuch „XHCS“ [11]). Die Ausgabe von Meldungen wird durch diesen Parameter nicht beeinflusst.

***NO**
Der 7-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

***EXTEND**
Der 8-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

name
Gibt den Namen eines speziellen 8-bit-Code an, der für Ein-/Ausgabeoperationen verwendet wird. Der Name muss 8 Byte lang sein.

MF =
Definiert besondere Anforderungen an die Makroauflösung (siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 605ff beschrieben.

Register 15 enthält im rechtsbündigen Byte einen Returncode. EQUATE-Anweisungen dafür können mit dem Makro CMDANALY generiert werden.

X’00’ normale Beendigung

X’04’ nicht behebbarer Systemfehler

X’08’ Operandenfehler im Makroaufruf

X’0C’ Übergabebereich zu klein

X’10’ Eingabeende (EOF) oder Anweisung fehlerhaft, danach wurde Eingabeende (EOF) erkannt

X’14’ Anweisung fehlerhaft, danach wurde Kommando erkannt

X’18’ Anweisung ist zwar in Ordnung, die vom Programm übergebenen Defaultwerte sind aber fehlerhaft

X’1C’ Anweisung fehlerhaft, danach wurde STEP erkannt

X’20’ Fehlerdialog nicht möglich

X’24’ Fehlerdialog wurde vom Anwender abgelehnt (Exit-Funktion ausgelöst)

X’2C’ END-Anweisung wurde gelesen

X’34’ Anweisung fehlerhaft, danach wurde END erkannt

X’38’ SDF ist nicht verfügbar

X’44’ Syntaxdatei nicht gefunden

X’4C’ Das Programm ist oberhalb der 16 MByte-Grenze nicht ablauffähig, weil SDF nicht geladen ist. Bitte Systembetreuung verständigen.

X’5C’ INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten

X’64’ XHCS-Fehler

9.1.4 RDSTMT
Anweisung lesen und analysieren

Der Makro RDSTMT bewirkt, dass SDF

– eine Programmanweisung von SYSSTMT einliest (Für die Systemdatei SYSSTMT gilt die gleiche Zuweisung, die für die Systemdatei SYSDTA getroffen ist. Bezüglich Folgezeilen, Fortsetzungszeichen und Angabe von Bemerkungen gelten für die Anweisungseingabe von SYSSTMT die gleichen Regeln wie für die Kommandoeingabe von SYSCMD.)

– die eingelesene Anweisung analysiert und

– das Analyseergebnis an das Programm übergibt.

Voraussetzung ist, dass eine aktivierte Syntaxdatei die Definition des Programms und seiner Anweisungen enthält. Die Eingabelänge einer über RDSTMT eingelesenen Anweisung beträgt 16364 Byte.

Operation	**Operanden**
RDSTMT	PROGRAM = name ,OUTPUT = addr / (r1) [,STMT = *ALL / (name,...) / *ADDR(addr/(r))] [,PREFER = *ALL / name / *ADDR(addr/(r))] [,DEFAULT = *NO / (addr,...)] [,MESSAGE = *NO / addr / (r3)] [,PROT = YES / NO] [,BUFFER = *NO / addr / (r)] [,INVAR = *NO / addr / (r)] [,SPIN = NO / YES] [,ERRSTMT = STEP / NEXT] [,CALLID = *NO / addr / (r7)] [,CCSNAME = *NO / *EXTEND / name] [,MF = { L / (E,(1)) / (E,opadr) }]

PROGRAM = name
Interner Name des Programms, das den Makroaufruf absetzt. In der Syntaxdatei ist dieser Name in der Programmdefinition abgelegt (siehe ADD-PROGRAM). Er ist mindestens ein und maximal acht Byte lang.

OUTPUT = addr / (r1)
Adresse des normierten Übergabebereichs bzw. Register, das diese Adresse enthält. Der Bereich muss auf Wortgrenze beginnen. Das Programm muss vor Aufruf des Makros RDSTMT dafür sorgen, dass in den ersten zwei Byte dieses Bereichs die maximal mögliche Bereichslänge steht (siehe Abschnitt „Format des normierten Übergabebereiches bis SDF V4.0“ auf Seite 605ff). SDF legt in dem Bereich das Analyseergebnis ab.

STMT =
Bestimmt, welche Anweisungen als Eingabe zulässig sind.

***ALL**
Alle Anweisungen sind zulässig.

(name,...)
Nur die Anweisungen, deren interner Anweisungsname angegeben ist, sind zulässig. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT). Er ist mindestens ein und maximal acht Byte lang. Die SDF-Standardanweisungen sind unabhängig von der hier getroffenen Festlegung immer zulässig.

***ADDR(addr/(r))**
Adresse der Liste der zulässigen Anweisungen. Diese Liste muss mit dem Makro CMDALLW generiert worden sein.

PREFER =
Ist nur für den geführten Dialog relevant und bestimmt, ob als nächste Eingabe eine bestimmte Anweisung erwartet wird.

***NO**
Es wird keine bestimmte Anweisung erwartet. SDF fragt mit dem Anweisungsmenü beim Benutzer ab, welche Anweisung er eingeben will.

name
Interner Name der Anweisung, deren Eingabe mit hoher Wahrscheinlichkeit zu erwarten ist. SDF gibt kein Anweisungsmenü aus, in dem der Benutzer die einzugebende Anweisung auswählt, sondern direkt den Operandenfragebogen für die erwartete Anweisung. Der Benutzer kann allerdings statt der erwarteten eine andere Anweisung eingeben.

Beispiel:

Nach MODIFY-OPERAND erwartet SDF-A als nächste Anweisung MODIFY-VALUE. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe ADD-STMT). Er ist mindestens ein und maximal acht Byte lang.

***ADDR(addr/(r))**

Adresse eines 8 Byte langen Bereiches, das den internen Namen der zu erwartenden Anweisung enthält. Der Name muss linksbündig ausgerichtet und mit Leerzeichen (X'40') aufgefüllt sein.

DEFAULT =

Bestimmt, ob SDF folgende Werte durch vom Programm dynamisch erzeugte Werte ersetzt.

– eingegebene Operandenwerte oder
– Default-Werte der Operanden

In der Syntaxdatei müssen die Operanden bzw. Operandenwerte entsprechend definiert sein (siehe ADD-OPERAND ...,OVERWRITE-POSSIBLE=*YES),... bzw. ADD-VALUE ..., VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). Der vom Programm erzeugte Wert mus s gültiger Operandenwert sein.
Im geführten Dialog zeigt SDF die vom Programm erzeugten Werte im Fragebogen.

Beispiel:

SDF-A ersetzt in den eingegebenen MODIFY-Anweisungen den Wert *UNCHANGED durch den aktuellen Wert.
Stehen die zu defaultierenden Operanden in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten: Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit zu defaultierenden Operanden hängt. Auf der anderen Seite gibt der Anwender ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit zu defaultierenden Operanden hängt. SDF versucht zunächst, die vom Benutzer eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen. Ist über den struktureinleitenden Wert keine eindeutige Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Benutzer den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

***NO**

SDF soll die eingegebenen Operandenwerte nicht durch vom Programm dynamisch erzeugte Werte ersetzen.

(addr,...)
In einer oder in mehreren der möglichen Anweisungen soll SDF eingegebene Operandenwerte durch vom Programm dynamisch erzeugte Werte ersetzen. Unter den angegebenen Adressen (adr21,...) stehen in dem Programm die Umsetzbeschreibungen (siehe Abschnitt „Format des normierten Übergabebereiches bis SDF V4.0“ auf Seite 605ff) für diese Anweisungen. Je Anweisung kann nur eine einzige Umsetzbeschreibung angegeben werden. Die Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche eingegebenen Operandenwerte in welche Werte umzusetzen sind. Die Bereiche für die Umsetzbeschreibungen, die für die Standardwerte des Programms übergeben werden, müssen auf Wortgrenze ausgerichtet sein. Dies gilt ebenso für den Ausgabebereich der Makros (OUTPUT-Operand).

MESSAGE =
Bestimmt, ob SDF bei der Aufforderung zur Anweisungseingabe eine Meldung ausgeben soll. Diese wird im geführten Dialog in das Anweisungsmenü integriert.

***NO**
SDF soll keine Meldung ausgeben.

addr / (r3)
Adresse des auszugebenden Meldungstextes bzw. Register, das diese Adresse enthält. Der Text wird als Satz variabler Länge erwartet, die maximale Länge beträgt 400 Zeichen. In SDF-formatierten Bildschirmen werden jedoch nur die ersten 80 Zeichen dargestellt. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden. Dieser Bereich muss auf Halbwortgrenze ausgerichtet sein.

PROT =
Bestimmt, ob SDF zu protokollierende Eingaben und Meldungen nach SYSOUT schreibt. Falls nicht nach SYSOUT geschrieben wird, sollte der Benutzer des Programms in der Programmdokumentation darüber informiert werden. Dieser Parameter ist im Gegensatz zum TRSTMT-Makro ein Flag (set=NO, reset=YES). Ein Protokollpuffer kann bei BUFFER bereitgestellt werden.

YES
SDF soll zu protokollierende Eingaben und Meldungen nach SYSOUT schreiben.

NO
SDF soll keine Protokollierung durchführen. Mit folgendem Verhalten ist zu rechnen:

Ergebnis der Analyse	PROT=YES	PROT=NO
Kein Fehler:	Eingabeanweisung	– / –
Syntaxfehler:	1. Eingabeanweisung 2. Syntaxfehlermeldung 3. Spin-off-Meldung	Spin-off-Meldung

BUFFER =
Das Protokoll der Anweisung und die Fehlermeldungen können in einen vom Benutzer bereitgestellten Bereich geschrieben werden.

***NO**
Es wird kein Puffer bereitgestellt.

addr / (r)
Adresse eines Bereichs bzw. Register, in dem das Protokoll der angegebenen Anweisung und die Meldungen, unabhängig vom Wert, der bei PROT angegeben wurde, abgelegt wird. Der Bereich muss auf Wortgrenze ausgerichtet sein. Das erste Halbwort muss die Gesamtlänge des Bereiches enthalten. SDF schreibt die tatsächliche Länge des Ausgabe-Protokolls in das zweite Halbwort. Im Anschluss daran werden die Protokollsätze als Sätze variabler Länge in den Bereich geschrieben.
Wenn der Puffer nicht leer ist, ist der erste Datensatz normalerweise das Protokoll des Eingabe-Kommandos. Die weiteren Datensätze enthalten Meldungen. Wenn kein Eingabeprotokoll zur Verfügung steht oder ausgegeben werden kann, wird ein doppelter Schrägstrich („//“) in den Ausgabebereich geschrieben.

INVAR =
Legt fest, ob die INVARIANT-INPUT-Form der Anweisung abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task zu dieser Zeit erlaubt sind. Im Gegensatz zur Protokollierungsform LOGGING=INVARIANT-FORM (siehe MODIFY-SDF-OPTIONS) werden Kennwörter und geheime Operanden jedoch *nicht* ausgeblendet.

***NO**
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

addr / (r)
Gibt die Adresse eines Puffers an, in den SDF die INVARIANT-INPUT-Form der Anweisung schreibt. Der Puffer muss auf Wortgrenze ausgerichtet sein und das erste Halbwort muss die Länge des Puffers enthalten. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab. Der Puffer hat dann folgenden Inhalt:

1. HW	2. HW	3. HW (Halbwort)	
buflen	reclen	filler	invariant-input

buflen: Länge des Puffers
reclen: Länge des Satzes, den SDF schreibt
filler: Füllzeichen
invariant-input: INVARIANT-INPUT-Form des Kommandos, beginnt beim 7. Byte.

SPIN =
Bestimmt, welche Anweisung SDF im Stapelbetrieb als nächste liest und analysiert.

NO
SDF soll die nächste Anweisung der Anweisungsfolge lesen und bearbeiten.

YES
SDF soll alle Anweisungen bis zur nächsten STEP-Anweisung bzw. bis zur END-Anweisung überlesen und ggf. die Bearbeitung mit der Anweisung fortsetzen, die der STEP-Anweisung folgt.

ERRSTMT =
Bestimmt, welche Anweisung den Spin-off-Mechanismus abbricht, wenn SDF einen Syntaxfehler für die Leseanweisung feststellt.

STEP
SDF leitet Spin-off ein bis STEP (oder END) erkannt wird.
Der Returncode ist X'1C', X'34',...

NEXT
SDF leitet keinen Spin-off ein: Die nächste Anweisung wird beim nächsten RDSTMT-Aufruf gelesen. Returncode ist dann X'50'.

CALLID =
Diese Funktion bezieht sich auf den Gebrauch der Makros OPNCALL und CLSCALL. CALLID benennt den Programmkontext (=Syntaxdateihierarchie, durch einen OPNCALL-Makro eröffnet), in dem die Anweisung gelesen und analysiert werden muss. Der Name der Syntaxdateihierarchie (CALLID) muss den 4 Byte langen Wert haben, der von SDF an das Feld zurückgegeben wird, welches durch den Operanden CALLID im OPNCALL-Makro bezeichnet wurde.

***NO**
Die aktuelle Syntaxdateihierarchie (Kontext) der Task wird für die Analyse der Anweisung verwendet. Das kann z.B. die für die Task beim LOGON eröffnetete Syntaxdateihierarchie sein.

addr / (r7)
Adresse des Aufrufprüfungsfeldes oder Register, das diese Adresse enthält. Der Bereich muss auf Wortgrenze ausgerichtet sein.

CCSNAME =
Gibt den Namen des Zeichensatzes an, der für den Korrekturdialog auf 8-bit-Terminals und für die Konvertierung von Klein- in Großbuchstaben verwendet wird. Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Jeder codierte Zeichensatz wird durch seinen Namen (Coded Character Set Name, CCSN) bestimmt (siehe Handbuch „XHCS“ [11]). Die Ausgabe von Meldungen wird durch diesen Parameter nicht beeinflusst.

***NO**
Der 7-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

***EXTEND**
Der 8-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

name
Gibt den Namen eines speziellen 8-bit-Code an, der für Ein-/Ausgabeoperationen verwendet wird. Der Name muss 8 Byte lang sein.

MF =
definiert besondere Anforderungen an die Makroauflösung (siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 605ff beschrieben.

Register 15 enthält im rechtsbündigen Byte einen Returncode und im linksbündigen Byte Angaben über die Zuweisung von SYSSTMT. EQUATE-Anweisungen dafür können mit dem Makro CMDANALY generiert werden.

X'00' normale Beendigung

X'04' nicht behebbarer Systemfehler

X'08' Operandenfehler im Makroaufruf

X'0C' Übergabebereich zu klein

X'10' Eingabeende (EOF) oder Anweisung fehlerhaft, Eingabeende (EOF) wurde erkannt

X'14' Anweisung fehlerhaft, Kommando wurde erkannt

X'18' Anweisung ist zwar in Ordnung, die vom Programm übergebenen Defaultwerte sind aber fehlerhaft

X'1C' Anweisung fehlerhaft, STEP wurde erkannt

X'28' Puffer zu klein, Protokoll abgebrochen

X'2C' END-Anweisung wurde gelesen

X'34' Anweisung fehlerhaft, die nächste zu bearbeitende Anweisung ist END

X'38' SDF ist nicht verfügbar

X'3C' Programm nicht in Syntaxdatei bekannt

X'44' Syntaxdatei nicht gefunden

X'4C' Das Programm ist oberhalb der 16 M Byte-Grenze nicht ablauffähig, weil SDF nicht geladen ist. Bitte Systembetreuung verständigen.

X'50' Anweisung fehlerhaft, Spin-off wurde nicht eingeleitet

X'5C' INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten

X'64' XHCS-Fehler

Zuweisung von SYSSTMT:

X'01' SYSSTMT=Datensichtstation

X'02' SYSSTMT=SYSCMD in Nicht-S-Prozedur oder Datei

X'03' SYSSTMT=Kartenleser

X'04' SYSSTMT=Diskette

X'05' SYSSTMT=SYSCMD in S-Prozedur

X'06' SYSSTMT=S-Variable

9.1.5 TRSTMT Anweisung analysieren

Der Makro TRSTMT bewirkt, dass SDF

– eine Programmanweisung, die im Programm selbst abgelegt ist, analysiert und

– das Analyseergebnis an das Programm übergibt.

Bei der Analyse der übergebenen Anweisung können zusätzliche Anweisungen entstehen, indem ein System-Exit die übergebene Anweisung durch mehrere Anweisungen ersetzt. Die Behandlung dieser zusätzlichen Anweisungen liegt in der Verantwortung des Programms.

Voraussetzung ist, dass eine aktivierte Syntaxdatei die Definition des Programms und seiner Anweisungen enthält.

Operation	Operanden
TRSTMT	PROGRAM = name ,INPUT = *NO / addr / (r1) ,OUTPUT = addr / (r2) [,STMT = *ALL / (name,...) / *ADDR(addr/(r))] [,DIALOG =NO / YES / ERROR] [,MESSAGE = *NO / addr / (r3)] [,PROT = *NO / *YES / addr / (r4)] [,INVAR = *NO / addr / (r)] [,DEFAULT = *NO / (addr,...)] [,ERROR = NO / YES] [,CALLID = *NO / addr / (r7)] [,EXECUTE = NO / YES] [,PROCMOD = ANY / NO / YES] [,CCSNAME = *NO / *EXTEND / name] [,INPUTSAV = *NO / YES] [,MF = { L / (E,(1)) / (E,opadr) }]

PROGRAM = name
interner Name des Programms, das durch den Makroaufruf ausgeführt wird. In der Syntaxdatei ist dieser Name in der Programmdefinition abgelegt (siehe ADD-PROGRAM). Er ist mindestens ein und maximal acht Byte lang.

INPUT =
bestimmt, welche Anweisung SDF analysieren soll.

***NO**
SDF soll keine im Programm abgelegte Anweisung analysieren, sondern eine durch System-Exit zusätzlich bereitgestellte Anweisung.

addr / (r1)
SDF soll die Anweisung analysieren, deren Adresse angegeben ist bzw. in dem angegebenen Register steht. SDF erwartet die Anweisung als Satz variabler Länge im üblichen BS2000-Format. Der Satzbereich muss auf Halbwortgrenze ausgerichtet sein.

OUTPUT = addr / (r2)
Adresse des normierten Übergabebereichs bzw. Register, das diese Adresse enthält. Der Bereich muss auf Wortgrenze beginnen. Das Programm muss vor Aufruf des Makros TRSTMT dafür sorgen, dass in den ersten zwei Byte dieses Bereichs die maximal mögliche Bereichslänge steht (siehe Abschnitt „Format des normierten Übergabebereiches bis SDF V4.0“ auf Seite 605). SDF legt in dem Bereich das Analyseergebnis ab.

STMT =
bestimmt, welche Anweisungen als Eingabe zulässig sind.

***ALL**
Alle Anweisungen sind zulässig.

(name,...)
Nur die Anweisungen, deren interner Anweisungsname angegeben ist, sind zulässig. Der interne Anweisungsname ist in der Syntaxdatei in der Anweisungsdefinition abgelegt (siehe Anweisung ADD-STMT). Er ist mindestens ein und maximal acht Byte lang. Die SDF-Standardanweisungen sind, unabhängig von der hier getroffenen Festlegung, immer zulässig.

***ADDR(addr/(r))**
Adresse der Liste der zulässigen Anweisungen. Diese Liste muss mit dem Makro CMDALLW generiert worden sein.

DIALOG =
bestimmt, ob SDF bei der Anweisungsanalyse einen Dialog führen soll. Dieser Operand ist nur relevant, wenn das Programm in einer interaktiven Task abläuft.

NO
SDF soll keinen Dialog führen.

YES
SDF soll die vom Programm übergebene Anweisung dem Benutzer im Dialog zu einer eventuellen Änderung anbieten, soweit das mit den geltenden SDF-Festlegungen für den Dialog verträglich ist (siehe MODIFY-SDF-OPTIONS und SET-GLOBALS).

ERROR
SDF soll nur beim Erkennen von Syntaxfehlern einen Dialog führen. Falls die Anweisung Semantikfehler enthält, kann das Programm mit CORSTMT einen Semantikfehlerdialog anstoßen.

MESSAGE =
bestimmt, ob SDF eine Meldung ausgeben soll, wenn dem Benutzer die Anweisung zur Überprüfung und ggf. Änderung angeboten wird (nur relevant für DIALOG ≠ *NO). SDF integriert diese Meldung in den Fragebogen.

***NO**
SDF soll keine Meldung ausgeben.

addr / (r3)
Adresse des auszugebenden Meldungstextes bzw. Register, das diese Adresse enthält. Der Text wird als Satz variabler Länge erwartet, die maximale Länge beträgt 400 Zeichen. In SDF-formatierten Bildschirmen werden jedoch nur die ersten 80 Zeichen dargestellt. Enthält der Text Bildschirmsteuerzeichen, so kann die Menü-Maske zerstört werden. Der Satzbereich muss auf Halbwortgrenze ausgerichtet sein.

PROT =
bestimmt, ob SDF zu protokollierende Eingaben und Meldungen nach SYSOUT schreibt. Falls nicht nach SYSOUT geschrieben wird, sollte der Benutzer des Programms in der Programmdokumentation darüber informiert werden. Dieser Parameter ist im Gegensatz zum RDSTMT-Makro als Ganzzahlwert implementiert (Byte-Feld). Ein Protokollpuffer kann bereitgestellt werden.

***NO**
SDF soll keine Protokollierung durchführen.

***YES**
SDF soll zu protokollierende Eingaben und Meldungen nach SYSOUT schreiben.

addr / (r4)
Adresse eines Puffers oder Register, das die Adresse enthält. SDF soll die zu protokollierenden Eingaben und Meldungen in diesen Puffer schreiben. Der Puffer muss an einer Halbwortgrenze beginnen. Die Länge des Puffers steht in den Bytes 0 und 1, die Länge des zu protokollierenden Satzes in Byte 2 und 3.
Wenn der Puffer nicht leer ist, ist der erste Datensatz normalerweise das Protokoll des Eingabe-Kommandos. Die weiteren Datensätze enthalten Meldungen aller Art. Wenn kein Eingabeprotokoll zur Verfügung steht oder ausgegeben werden kann, wird ein doppelter Schrägstrich („//“) in den Ausgabebereich geschrieben.
Der PROT-Parameter hat folgende Wirkung:

Ergebnis der Analyse	**PROT-Parameter**	
	YES (oder addr / reg)	**NO**
Kein Fehler:	Eingabeanweisung	– / –
Syntaxfehler:	1. Eingabeanweisung 2. Syntaxfehlermeldung	– / –

INVAR =
Legt fest, ob die INVARIANT-INPUT-Form der Anweisung abgespeichert wird. Das heißt, dass die Anweisung mit allen eingegebenen Operanden, allen durch Default-Werte vorbelegten Operanden und mit allen Operandenwerten abgelegt wird, die für die Task zu dieser Zeit erlaubt sind. Die INVARIANT-INPUT-Form ist damit die größtmögliche Eingabeform für eine Anweisung, die für einen Benutzer mit bestimmten Privilegien und im gewählten Dialogmodus zulässig ist. Im Gegensatz zur Protokollierungsform LOGGING= INVARIANT-FORM (siehe MODIFY-SDF-OPTIONS) werden Kennwörter und geheime Operanden jedoch *nicht* ausgeblendet.

INVAR = *NO
Die INVARIANT-INPUT-Form der Anweisung wird nicht abgespeichert.

INVAR = addr / (r)
Gibt die Adresse eines Puffers an, in den SDF die INVARIANT-INPUT-FORM der Anweisung schreibt. Der Puffer muss auf Wortgrenze ausgerichtet sein und das erste Halbwort muss die Länge des Puffers enthalten. SDF legt die INVARIANT-INPUT-Form ab dem zweiten Halbwort als Satz mit variabler Satzlänge ab. Der Puffer hat dann folgenden Inhalt:

1. HW	2. HW	3. HW (Halbwort)	
buflen	reclen	filler	invariant-input

buflen: Länge des Puffers
reclen: Länge des Satzes, den SDF schreibt
filler: Füllzeichen
invariant-input: INVARIANT-INPUT-Form der Anweisung, beginnt beim 7. Byte.

DEFAULT =
bestimmt, ob SDF folgende Werte durch vom Programm dynamisch erzeugte Werte ersetzt:
– eingegebene Operandenwerte oder
– Default-Werte der Operanden

In der Syntaxdatei müssen die Operanden bzw. Operandenwerte entsprechend definiert sein (siehe ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES,... bzw. ADD-VALUE..., VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). Der vom Programm erzeugte Wert muss gültiger Operandenwert sein. Im geführten Dialog zeigt SDF die vom Programm erzeugten Werte im Fragebogen.

Beispiel:

In den eingegebenen MODIFY-Anweisungen für SDF-A wird der Wert *UNCHANGED durch den aktuellen Wert ersetzt. Stehen die mit einem Standardwert zu versehenden Operanden in einer Struktur, deren Einleiter mit LIST-ALLOWED=*YES definiert ist (siehe ADD-VALUE), so kann folgender Fall eintreten:
Die Umsetzbeschreibung enthält mehrere Listenelemente, an denen eine Struktur mit Operanden hängt, die mit einem Standardwert zu versehen sind. Auf der anderen Seite gibt der Anwender ebenfalls mehrere Listenelemente ein, an denen eine Struktur mit eben solchen Operanden hängt.
SDF versucht zunächst, die vom Anwender eingegebenen und die in der Umsetzbeschreibung angegebenen Strukturen einander über den Wert des Struktureinleiters zuzuordnen. Ist über den struktureinleitenden Wert keine eindeutige Zuordnung möglich, weil keiner der eingegebenen Werte mit denen in der Umsetzbeschreibung übereinstimmt oder weil der Anwender den übereinstimmenden Wert mehrfach eingegeben hat, so erfolgt die Zuordnung über die Position des Struktureinleiters in der Liste.

***NO**
SDF soll die angegebenen Operandenwerte nicht durch vom Programm dynamisch erzeugte Werte ersetzen.

(addr,...)
In einer oder in mehreren der möglichen Anweisungen soll SDF angegebene Operandenwerte durch vom Programm dynamisch erzeugte Werte ersetzen. Unter den angegebenen Adressen (adr51,...) stehen in dem Programm die Umsetzbeschreibungen für diese Anweisungen (siehe Abschnitt „Format des normierten Übergabebereiches bis SDF V4.0“ auf Seite 605ff). Je Anweisung kann nur eine einzige Umsetzbeschreibung angegeben werden. Eine Umsetzbeschreibung enthält u.a. den internen Anweisungsnamen und die Information, welche angegebenen Operandenwerte in welche Werte umzusetzen sind. Die Bereiche für die Umsetzbeschreibungen, die für die Default-Werte des Programms übergeben werden, müssen auf Wortgrenze ausgerichtet sein. Dies gilt auch für den Ausgabebereich der Makros (Operand OUTPUT).

ERROR =
bestimmt, wie der beim Operanden MESSAGE angegebene Meldungstext ausgegeben wird.

NO
SDF soll den Meldungstext als Meldung ausgeben.

YES
SDF soll den Meldungstext als Fehlermeldung ausgeben.

CALLID =
bestimmt, welcher Kontext von SDF benutzt wird, um das Kommando zu analysieren.

***NO**
Die gegenwärtig aktivierte Syntaxdateihierarchie wird verwendet.

addr / (r7)
Adresse eines 4 Byte langen Feldes oder Register, das diese Adresse enthält. Der Aufrufende übermittelt die CALLID des Kontextes, den er verwenden will. Dieses Feld muss auf Wortgrenze ausgerichtet sein.

EXECUTE = NO / YES
bestimmt, ob SDF-Standardanweisungen ausgeführt werden. EXECUTE ist ohne Bedeutung, wenn der TRSTMT-Makro keinen Bezug auf eine neue Syntaxdateihierarchie nimmt (CALLID=*NO), z.B. wenn die aktuelle Syntaxdateihierarchie verwendet wird. Dann werden die SDF-Standardanweisungen immer von TRSTMT ausgeführt (vorausgesetzt, sie sind in der aktuellen Syntaxdateihierarchie vorhanden). Der Operand gehört zum multihierarchischen Merkmal, das mit dem vorangehenden Operanden CALLID eingeleitet wird. Bezieht sich TRSTMT auf eine parallel eröffnete Syntaxdateihierarchie (CALLID= adr7 / (r7)), werden die SDF-Standardanweisungen bei EXECUTE=ES ausgeführt.

PROCMOD =
bestimmt, in welcher Umgebung der Benutzer arbeitet. SDF führt eine Prüfung durch: Anweisungen, die in der angegebenen Umgebung nicht erlaubt sind, werden von SDF nicht akzeptiert. Der Operand nimmt Bezug auf die Möglichkeit, mehrere Syntaxdateihierarchien zu eröffnen. Er ist nur von Bedeutung, wenn der Makro-Aufruf sich auf eine neue Syntaxdateihierarchie bezieht, die zusätzlich zur aktuellen eröffnet wurde. Ist keine CALLID bestimmt (CALLID=*NO), hat PROCMOD keine Bedeutung; z.B. wird der Wert ANY automatisch gesetzt und auf den aktuellen Prozedurmodus Bezug genommen.

ANY
Es wird keine Überprüfung vorgenommen. Anweisungen werden immer analysiert.

YES
Anweisungen werden so behandelt, als würden sie aus einer Prozedurdatei gelesen. Sie werden z.B. analysiert, wenn sie in der Syntaxdatei mit DIALOG-PROC-ALLOWED=YES definiert sind und wenn das Programm im Dialog arbeitet, oder bei BATCH-PROC-ALLOWED=YES in Stapelaufträgen.

NO
Anweisungen werden so behandelt, als würden sie von einer Hauptebene (primary level) gelesen, z.B. von der Bildschirmeingabe oder aus einem Stapelauftrag. Sie werden analysiert, wenn sie in der Syntaxdatei mit DIALOG-ALLOWED=*YES definiert wurden und wenn das Programm im Dialog läuft, oder bei BATCH-ALLOWED=*YES in Stapelaufträgen.

CCSNAME =
Gibt den Namen des Zeichensatzes an, der für den Korrekturdialog auf 8-bit-Terminals und für die Konvertierung von Klein- in Großbuchstaben verwendet wird. Jedes Terminal arbeitet mit einem bestimmten Zeichensatz. Ein codierter Zeichensatz (CCS, Coded Character Set) ist die eindeutige Darstellung der Zeichen eines Zeichensatzes in binärer Form. Jeder codierte Zeichensatz wird durch seinen Namen (Coded Character Set Name, CCSN) bestimmt (siehe Handbuch „XHCS“ [11]). Die Ausgabe von Meldungen wird durch diesen Parameter nicht beeinflusst.

***NO**
Der 7-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

***EXTEND**
Der 8-bit-Standard-Code wird für Ein-/Ausgabeoperationen verwendet.

name
Gibt den Namen eines speziellen 8-bit-Code an, der für Ein-/Ausgabeoperationen verwendet wird. Der Name muss 8 Byte lang sein.

INPUTSAV =
gibt an, ob die letzten Eingaben in einer Liste gespeichert werden. Auf diese Liste kann mit Hilfe eines RESTORE-Mechanimus erneut zugegriffen werden (siehe Anweisungen MODIFY-SDF-OPTIONS und RESTORE-SDF-INPUT).

***NO**
Die Eingaben werden nicht gespeichert.

***YES**
Die Eingaben werden in einem Puffer gespeichert.

MF =
definiert besondere Anforderungen an die Makroauflösung (siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil“ [8]).

L
Es wird nur der Datenteil der Makroauflösung (Operandenliste) generiert. Das erfordert, dass im Makroaufruf keine Operandentypen mit ausführbarem Code auftreten. Der generierte Datenteil hat die im Namensfeld des Makroaufrufs angegebene Adresse.

(E,(1)) / (E,opadr)
Es wird nur der Befehlsteil der Makroauflösung generiert. Auf den zugehörigen Datenteil (Operandenliste) wird mit der Adresse „opadr“ verwiesen. Diese steht entweder in Register 1 oder wird direkt angegeben.

Rückinformation und Fehleranzeigen

Der Aufbau des Übergabebereichs ist auf den Seiten 605ff beschrieben.

Register 15 enthält im rechtsbündigen Byte einen Returncode, im linksbündigen Byte einen Indikator über die Existenz zusätzlicher Anweisungen. EQUATE-Anweisungen dafür können mit dem Makro CMDANALY generiert werden.

X’00’	normale Beendigung
X’04’	nicht behebbarer Systemfehler
X’08’	Operandenfehler im Makroaufruf
X’0C’	Übergabebereich zu klein
X’18’	Anweisung ist zwar in Ordnung, die vom Programm übergebenen Default-Werte sind aber fehlerhaft
X’1C’	Anweisung fehlerhaft
X’20’	Fehlerdialog nicht möglich
X’24’	Fehlerdialog wurde abgelehnt
X’28’	Fehler- oder Protokollbereich zu klein
X’2C’	END-Anweisung wurde gelesen
X’38’	SDF ist nicht verfügbar
X’3C’	Programm in Syntaxdatei nicht bekannt
X’44’	Syntaxdatei nicht gefunden
X’48’	SDF-Kommando (oder -Anweisung) ausgeführt
X’4C’	Das Programm ist oberhalb der 16 MByte-Grenze nicht ablauffähig, weil SDF nicht geladen ist. Bitte Systembetreuung verständigen.

X’5C’ INVAR-Puffer zu klein, INVARIANT-INPUT abgeschnitten

X’64’ XHCS-Fehler

Indikator für zusätzliche Anweisungen (durch System-Exit):

X’00’ es existieren keine weiteren Anweisungen

X’01’ es existieren weitere Anweisungen

9.2 Sich ausschließende Datentypen

Folgende Tabelle zeigt an, welche Kombinationen von Datentypen für einen Operanden zulässig sind.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
1: alphan.-name	()	x	x	x			x	x	x	x	x	x			x			x	x		x		x	x	
2: cat-id	x	n	x	x					x		x	x			x			x	x		x		x	x	
3: command-rest	x	x	n	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
4: comp.-name	x	x	x	()		x	x	x	x	x	x	x			x		x	x	x		x		x	x	
5: c-string			x		()												x		x					x	x
6: date			x	x		n			x										x					x	
7: device	x		x	x			x		x		x	x			x			x	x		x		x	x	
8: fixed	x		x	x				()		x		x					x		x	x	x		x	x	
9: filename	x	x	x	x		x	x		()		x	x			x		x	x	x		x		x	x	
10: integer	x		x	x				x		()		x							x	x	x		x	x	
11: keyword	x	x	x	x			x		x						x			x	x		x		x	x	
12: keyw-number	x	x	x	x			x	x	x	x					x			x	x		x		x	x	
13: *keyword			x																x					x	
14: *keyw-numb.			x																x					x	
15: name	x	x	x	x			x		x		x	x			()			x	x		x		x	x	
16: partial-filen.			x													()			x					x	
17: prod.-version			x	x	x			x	x								()		x		x			x	x
18: struct.-name	x	x	x	x			x		x		x	x			x			()	x		x		x	x	
19: text	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	()	x	x	x	x	x	x
20: time			x					x		x									x	n	x			x	
21: vsn	x	x	x	x			x	x	x	x	x	x			x		x	x	x	x	()		x	x	
22: x-string			x																x			()		x	
23: x-text	x	x	x	x			x	x	x	x	x	x			x			x	x		x		()	x	
24: posix-filen. / posix-pathn.	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	()	x
25: 'posix-filen.' / 'posix-pathn.'			x		x												x		x					x	()

kein Eintrag	Kombination der Datentypen ist uneingeschränkt möglich.
x	Kombination der Datentypen ist nur bei VALUE-OVERLAPPING=*YES erlaubt.
( )	Kombination gleicher Datentypen mit unterschiedlicher Länge bzw. unterschiedlichem Wertebereich ist nur bei VALUE-OVERLAPPING=*YES erlaubt.
n	Mehrfachangabe des gleichen Datentyps ist nicht möglich, da keine Zusatzattribute existieren, die zur Unterscheidung dienen könnten.

Fachwörter

Anwendungsbereich

Nach Benutzerkriterien zusammengestellte Menge von Kommandos, aus denen der Benutzer im geführten Dialog das gewünschte Kommando auswählen kann. Anwendungsbereiche können sich gegenseitig überlappen.

Datentyp

Vom Kommandoprozessor SDF auf syntaktische Korrektheit überprüfbarer Basis-Syntax-Typ. Die möglichen Datentypen sind der Tabelle „Datentypen“ auf Seite 11ff zu entnehmen.

Liste

Zuweisung mehrerer Operandenwerte zu einem Operanden in einem Kommando oder in einer Anweisung. Die Liste wird durch Klammerung der durch Komma getrennten Werte gebildet. Es ist eine Eigenschaft des jeweiligen Operanden, ob er eine Liste erlaubt. Listenklammern sind von Klammern für Strukturen durch den Kontext unterscheidbar.

Struktur

Syntaktische Zusammenfassung von mehreren Operanden. Diese Zusammenfassung wird syntaktisch durch eine Klammerung der Operanden ausgedrückt. Strukturklammern sind von Klammern für Listen durch den Kontext unterscheidbar. Eine Struktur kann entweder selbst ein Operandenwert sein, oder sie ist abhängig von der Spezifikation des Werts aus einer struktureinleitenden Eingabealternative.

Literatur

Wenden Sie sich zum Bestellen von Handbüchern bitte an Ihre zuständige Geschäftsstelle.

[1] **SDF V4.5A** (BS2000/OSD)
Einführung in die Dialogschnittstelle SDF
Benutzerhandbuch

Zielgruppe
BS2000/OSD-Anwender
Inhalt
Das Handbuch beschreibt die Dialog-Eingabe von Kommandos und Anweisungen im SDF-Format. Ein Schnelleinstieg mit leicht nachvollziehbaren Beispielen und weitere umfangreiche Beispiele erleichtern die Anwendung. SDF-Syntaxdateien werden erklärt.
Bestellnummer
U2339-J-Z125-8

[2] **SDF V4.5A** (BS2000/OSD)
SDF-Verwaltung
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an die Systembetreuung und an erfahrene BS2000-Benutzer.
Inhalt
Es beschreibt, wie SDF mit Hilfe von SDF-Kommandos und den Dienstprogrammen SDF-I, SDF-U und SDF-PAR installiert und verwaltet wird. Die Anweisungen von SDF-I, SDF-U und SDF-PAR sind vollständig beschrieben.
Bestellnummer
U2622-J-Z125-10

[3] **SDF-A** (BS2000/OSD)
Tabellenheft

Zielgruppe
Das Tabellenheft wendet sich an geübte BS2000-Benutzer mit Erfahrung bei der Anwendung von SDF-A.
Inhalt
Es enthält die SDF-A-Anweisungen in alphabetischer Reihenfolge. Daran anschließend sind die Makros und Funktionsaufrufe der SDF-Programmschnittstelle, die Formate des Übergabebereichs und die SDF-SIM-Anweisungen aufgeführt.
Bestellnummer
U2285-J-Z125-9

[4] **BS2000/OSD-BC V5.0**
Kommandos Band 1 - 5
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich sowohl an den nichtprivilegierten Anwender als auch an die Systembetreuung.
Inhalt
Die Bände 1 bis 5 enthalten die Kommandos ADD-... bis WRITE-... (BS2000/OSD-Grundausbau und ausgewählte Produkte) mit der Funktionalität für alle Privilegien. Die Kommando- und Operandenfunktionen werden ausführlich beschrieben; viele Beispiele unterstützen das Verständnis. Am Anfang jedes Bandes informiert eine Übersicht über alle in den Bänden 1-5 beschriebenen Kommandos.
Der Anhang von Band 1 enthält u.a. Informationen zur Kommandoeingabe, zu bedingten Jobvariablenausdrücken, Systemdateien, Auftragsschaltern, Geräte- und Volumetypen.
Der Anhang der Bände 4 und 5 enthält jeweils eine Übersicht zu den Ausgabespalten der SHOW-Kommandos der Komponente NDM. Der Anhang von Band 5 enthält zusätzlich eine Übersicht aller START-Kommandos.
In jedem Band ist ein umfangreiches Stichwortverzeichnis mit allen Stichwörtern der Bände 1-5 enthalten.
Bestellnummern
U2338-J-Z125-15 Kommandos Band 1, A – C
U41074-J-Z125-2 Kommandos Band 2, D – MOD-I
U21070-J-Z125-5 Kommandos Band 3, MOD-J – R
U41075-J-Z125-2 Kommandos Band 4, S – SH-O
U23164-J-Z125-4 Kommandos Band 5, SH-P – Z

[5] **BS2000/OSD-BC V5.0**
Kommandos Band 6, Ausgabe in S-Variablen und SDF-P-BASYS
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an Programmierer und Anwender, die Prozeduren erstellen.
Inhalt
Band 6 enthält die tabellarische Darstellung aller S-Variablen, die von den SHOW-Kommandos bei einer strukturierten Ausgabe mit Werten versorgt werden. Weitere Kapitel:
– Einführung in das Arbeiten mit S-Variablen
– SDF-P-BASYS V2.2A
Bestellnummer
U23165-J-Z125-4

[6] **BS2000/OSD-BC V5.0**
Einführung in die Systembetreuung
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an die Systembetreuung und das Operating des Betriebssystems BS2000/OSD.
Inhalt
Es sind u.a. folgende Themen zur Verwaltung und Überwachung des BS2000/OSD-Grundausbaus enthalten: Systemeinleitung, Parameterservice, Job- und Tasksteuerung, Speicher-, Geräte-, Benutzer-, Datei-, Pubset- und Systemzeit-Verwaltung, Privilegienvergabe, Accounting und Operatorfunktionen.
Bestellnummer
U2417-J-Z125-14

[7] **BS2000/OSD-BC V5.0**
Einführung in das DVS
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an den nichtprivilegierten Anwender und an die Systembetreuung.
Inhalt
Es beschreibt die Dateiverwaltung und -verarbeitung im BS2000.
Themenschwerpunkte:
– Datenträger und Dateien
– Datei- und Katalogverwaltung
– Datei- und Datenschutz
– OPEN-, CLOSE-, EOV-Verarbeitung
– DVS-Zugriffsmethoden (SAM, ISAM,...)
Wesentliche Neuheiten in der OSD-BC V5.0 sind die Einführung von Dateien ≥ 32 GB und die Möglichkeit, die TSOS-Miteigentümerschaft an Dateien einzuschränken.
Bestellnummer
U4237-J-Z125-7

[8] **BS2000/OSD-BC V5.0**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an alle BS2000/OSD-Assembler-Programmierer.
Inhalt
Das Handbuch enthält eine Zusammenstellung der Makroaufrufe an den Ablaufteil:
– Binden und Laden
– virtueller Speicher, Memory Pool, ESA
– Task- und Programmsteuerung
– ITC, Serialisierung, Eventing, DLM, Contingencys, STXIT
– Meldungswesen, Accounting, JMS, TIAM, VTSU, ...
Ausführliche Beschreibung aller Makroaufrufe in lexikalischer Reihenfolge mit Beispielen.
Allgemeiner Lernteil über ITC, Serialisierung, Eventing, DLM, Contingencies, STXIT, virtueller Speicher, Memory Pool, ESA, ...
Bestellnummer
U3291-J-Z125-9

[9] **BS2000/OSD-BC V5.0**
Systeminstallation
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an die BS2000/OSD-Systemverwaltung.
Inhalt
Beschrieben wird die Generierung der Hardware-Konfiguration mit UGEN und die Installationsdienste. Letztere beinhalten die Plattenorganisation mit MPVS, die Installation von Datenträgern mit dem Dienstprogramm SIR und das Subsystem IOCFCOPY.
Bestellnummer
U2505-J-Z125-15

[10] **SECOS V4.0** (BS2000/OSD)
Security Control System
Benutzerhandbuch

Zielgruppe
– BS2000-Systemverwalter
– BS2000-Anwender, die den erweiterten Zugriffsschutz für Dateien nutzen
Inhalt
Leistung und Anwendung der Funktionseinheiten:
– SRPM (Privilegien und Betriebsmittel verwalten)
– SRPMSSO (Single Sign On)
– GUARDS (Zugriffsbedingungsverwaltung und -auswertung für Objekte)
– GUARDDEF (Default Protection, Standardschutz)
– GUARDCOO (Co-owner Protection, Miteigentümerschutz)
– SAT (Protokollierung und Auswertung sicherheitsrelevanter Daten, Ereignisüberwachung mit Alarmfunktion).
Bestellnummer
U5605-J-Z125-6

[11] **XHCS V1.3** (BS2000/OSD)
8-bit-Code-Verarbeitung im BS2000/OSD
Benutzerhandbuch

Zielgruppe
Anwender der Zugriffsmethoden DCAM, TIAM und *open*UTM sowie Systembetreuer; Anwender, die von EHCS auf XHCS umstellen.
Inhalt
XHCS (Extended Host Code Support) ist ein Softwareprodukt des BS2000/OSD. Es ermöglicht Ihnen, erweiterte Zeichensätze bei 8-bit-Datenstationen zu nutzen. XHCS ist die zentrale Informationsquelle über die codierten Zeichensätze im BS2000/OSD. XHCS löst EHCS ab.
Bestellnummer
U9232-J-Z135-4
Erweiterungen siehe Readme-Datei zu XHCS V1.4

[12] **SDF-P V2.2A** (BS2000/OSD)
Programmieren in der Kommandosprache
Benutzerhandbuch

Zielgruppe
BS2000-Anwender und Systembetreuung.
Inhalt
SDF-P ist eine strukturierte Prozedursprache im BS2000. Nach einführenden Kapiteln zum Prozedur- und Variablenkonzept werden Kommandos, Funktionen und Makros ausführlich beschrieben.
Inhaltlicher Überblick:
– Schnelleinstieg SDF-P
– Prozedurkonzept von SDF-P
– S-Prozeduren erstellen, testen, aufrufen, steuern
– S-Variablen, S-Variablenströme, Funktionen, Ausdrücke
– Nicht-S-Prozeduren umstellen
– Makros, Builtin-Funktionen, SDF-P-Kommandos
Für den Einsatz von SDF-P V2.2A werden SDF-P-BASYS ≥ V2.1A, VAS ≥ V2.0A und SDF ≥ V4.1A vorausgesetzt.
Bestellnummer
U6442-J-Z125-5

[13] **IMON V2.5** (BS2000/OSD)
Installationsmonitor
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an die Systembetreuung des Betriebssystems BS2000/OSD.
Inhalt
Das Handbuch beschreibt die Installation und Verwaltung von BS2000-Software mit dem Installationsmonitor IMON und seinen drei Komponenten IMON-BAS, IMON-GPN und IMON-SIC.
In zwei Beispielkapiteln wird die Installation (standard und kundenspezifisch) mit der Komponente IMON-BAS für Systeme mit BS2000-OSD V2.0 und ab BS2000-OSD V3.0/V4.0 ausführlich dargestellt.
Bestellnummer
U21926-J-Z125-3

[14] **DSSM V4.0/SSCM V2.3**
Verwaltung von Subsystemen in BS2000/OSD
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an die Systembetreuung und die Softwareberatung des BS2000/OSD.
Inhalt
Es werden das Subsystemkonzept des BS2000/OSD, die Dynamische Subsystemverwaltung DSSM V4.0 und die Subsystemkatalog-Verwaltung SSCM V2.3 mit den dazugehörigen Kommandos und Anweisungen beschrieben.
DSSM bietet die Möglichkeit, benutzereigene Subsystem-Konfigurationen tasklokal zu erstellen und zu verwalten.
Bestellnummer
U23166-J-Z125-3

[15] **BS2000/OSD**
Softbooks Deutsch
CD-ROM

Zielgruppe
BS2000/OSD-Anwender
Inhalt
Auf der CD-ROM „BS2000/OSD SoftBooks Deutsch“ sind nahezu alle deutschen Handbücher und Readme-Dateien zur BS2000-Systemsoftware der aktuellsten BS2000/OSD-Version und auch von Vorgängerversionen gespeichert, incl. der aufgeführten Handbücher. Diese Softbooks finden Sie auch im Internet auf unserem Manual Server. Sie können in den Handbüchern nachschlagen oder sich vollständige Handbücher herunterladen.
Bestellnummer der CD-ROM
U26175-J8-Z125-1
Internet-Adresse
http://manuals.fujitsu-siemens.com

Stichwörter

Inhalt

SDF-A V4.1E (BS2000/OSD)

Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an erfahrene BS2000-Benutzer und an die Systembetreuung.

Inhalt
Es beschreibt, wie Syntaxdateien bearbeitet werden und erklärt die SDF-A-Funktionen anhand von Beispielen. Die SDF-A-Anweisungen sind in alphabetischer Reihenfolge aufgeführt.
Über die im Handbuch beschriebene SDF-Programmschnittstelle können auch Benutzerprogramme die SDF-Benutzeroberfläche nutzen. Mit dem Dienstprogramm SDF-SIM kann die Syntax von Kommandos und Anweisungen getestet werden.

Ausgabe: März 2002

Datei: sdf_a.pdf

Dieses Handbuch wurde erstellt von
cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Fujitsu Siemens Computers GmbH
Handbuchredaktion
81730 München

Kritik
Anregungen
Korrekturen

Fax: 0 700 / 372 00000

e-mail: manuals@fujitsu-siemens.com
http://manuals.fujitsu-siemens.com

Absender

Kommentar zu SDF-A V4.1E

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.