

Deutsch



FUJITSU Software BS2000

BINDER V2.7A

Binder in BS2000

Benutzerhandbuch

Ausgabe März 2016

Kritik... Anregungen... Korrekturen...

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

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

Zertifizierte Dokumentation nach DIN EN ISO 9001:2008

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2008 erfüllt.

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

Copyright und Handelsmarken

Copyright © 2016 Fujitsu Technology Solutions GmbH.

Alle Rechte vorbehalten.

Liefermöglichkeiten und technische Änderungen vorbehalten.

Alle verwendeten Hard- und Softwarenamen sind Handelsnamen und/oder Warenzeichen der jeweiligen Hersteller.

Inhalt

1	Einleitung	9
1.1	Zielsetzung und Zielgruppen des Handbuchs	12
1.2	Konzept des Handbuchs	12
1.3	Änderungen gegenüber dem Vorgängerhandbuch	13
1.4	Darstellungsmittel	14
2	Einführung in den BINDER	15
2.1	Bindelademodule (LLMs)	16
2.2	Logische Struktur eines LLM	17
2.3	Physische Struktur eines LLM	18
2.3.1	LLMs mit Einzel-Slice	19
2.3.2	LLMs mit nach Attributen gebildeten Slices	19
2.3.3	LLMs mit benutzerdefinierten Slices	20
2.4	Identifikation eines LLM	24
2.5	Inhalt eines LLM	30
2.6	Randbedingungen für LLMs	32
3	BINDER-Funktionen	35
3.1	Erzeugen, Ändern und Speichern eines LLM	37
3.1.1	Erzeugen eines LLM	37
3.1.2	Ändern eines LLM	40
3.1.3	Ändern der Merkmale eines LLM	43
3.1.4	Speichern eines LLM	43

3.2	Einfügen, Entfernen und Ersetzen von Modulen	47
3.2.1	Einfügen von Modulen	47
3.2.2	Entfernen von Modulen	52
3.2.3	Ersetzen von Modulen	54
3.3	Erzeugen und Ändern der logischen Struktur eines LLM	57
3.3.1	Erzeugen der logischen Struktur eines LLM	57
3.3.2	Ändern der logischen Struktur eines LLM	66
3.4	Erzeugen der physischen Struktur eines LLM	68
3.4.1	Benutzerdefinierte Slices	68
3.4.2	Nach Attributen gebildete Slices	75
3.4.3	Ändern des Typs der physischen Struktur	78
3.4.4	Verknüpfung zwischen PRIVATE- und PUBLIC-Slices	78
3.5	Befriedigen von Externverweisen	80
3.5.1	Regeln zur Befriedigung von Externverweisen	81
3.5.2	Autolink-Funktion	85
3.5.3	Behandeln unbefriedigter Externverweise	96
3.6	Behandlung von Namenskonflikten	102
3.7	Behandlung der COMMON-Bereiche	103
3.8	Behandlung der Pseudoregister	104
3.9	Relativierung der Adressen	104
3.10	Behandlung von Symbolen	105
3.10.1	Symbolnamen	106
3.10.2	Ändern von Symbolnamen	107
3.10.3	Ändern der Attribute von Symbolen	114
3.10.4	Ändern der Maskierung von Symbolen	117
3.10.5	Ändern des Typs von Symbolen	119
3.11	Mischen von Modulen	120
3.12	Festlegen von Standardwerten	126
3.13	Anzeigefunktionen	127
3.13.1	Anzeigen von Standardwerten	127
3.13.2	Anzeigen von Symbolinformationen	129
3.13.3	Anzeigen und Prüfen von Bibliothekselementen	132
3.14	Steuern der Protokollierung	134
3.15	Steuern der Fehlerbehandlung	135
3.15.1	Fehlerklassen	135
3.15.2	Meldungsbehandlung	136

4	BINDER-Ein-/Ausgabe	137
4.1	Eingaben für den BINDER	138
4.2	Ausgaben des BINDER	141
4.2.1	Gespeicherte LLMs	141
4.2.2	Listen	141
5	BINDER-Lauf	169
5.1	Aufrufen und Beenden des BINDER	169
5.2	Überwachung des BINDER-Laufs mit Jobvariablen	170
6	Unterprogrammchnittstelle	173
6.1	Makroaufruf BINDER	173
6.2	Beispiele	177
7	BINDER-Anweisungen	189
7.1	Einteilung der Anweisungen nach Funktionen	189
7.2	Hinweise zur SDF-Benutzeroberfläche	192
7.2.1	SDF-Syntaxdarstellung	193
7.3	Beschreibung der Anweisungen	209
	BEGIN-SUB-LLM-STATEMENTS	
	Beginn eines Sub-LLM festlegen	211
	END	
	Beenden des BINDER-Laufs	214
	END-SUB-LLM-STATEMENTS	
	Ende eines Sub-LLM festlegen	214
	INCLUDE-MODULES	
	Einfügen von Modulen	215
	MERGE-MODULES	
	Mischen von LLMs oder Sub-LLMs	225
	MODIFY-ERROR-PROCESSING	
	Steuern der Fehlerbehandlung	231
	MODIFY-LLM-ATTRIBUTES	
	Ändern der Merkmale eines LLM	235

MODIFY-MAP-DEFAULTS	
Ändern der Standardwerte für die Ausgabe von Listen	241
MODIFY-MODULE-ATTRIBUTES	
Ändern der Attribute von Modulen	244
MODIFY-STD-DEFAULTS	
Ändern von Standardwerten	249
MODIFY-SYMBOL-ATTRIBUTES	
Ändern der Attribute von Symbolen	255
MODIFY-SYMBOL-TYPE	
Ändern des Typs von Symbolen	259
MODIFY-SYMBOL-VISIBILITY	
Ändern der Maskierung von Symbolen	261
REMOVE-MODULES	
Entfernen von Modulen	264
RENAME-SYMBOLS	
Ändern von Symbolnamen	265
REPLACE-MODULES	
Ersetzen von Modulen	270
RESOLVE-BY-AUTOLINK	
Befriedigen von Externverweisen durch Autolink	279
SAVE-LLM	
Speichern eines LLM	287
SET-EXTERN-RESOLUTION	
Behandeln unbefriedigter Externverweise	301
SET-USER-SLICE-POSITION	
Festlegen der Lage einer Slice	303
SHOW-DEFAULTS	
Anzeigen von Standardwerten	305
SHOW-LIBRARY-ELEMENTS	
Anzeigen und Prüfen von Bibliothekselementen	307
SHOW-MAP	
Ausgeben von Listen	314
SHOW-SYMBOL-INFORMATION	
Ausgeben von Symbolinformationen	323
START-LLM-CREATION	
Erzeugen eines LLM	325
START-LLM-UPDATE	
Ändern eines LLM	331
START-STATEMENT-RECORDING	
BINDER-Anweisungen protokollieren	335
STOP-STATEMENT-RECORDING	
Protokollierung von BINDER-Anweisungen beenden	336

8	Nutzungsmodelle für Bindelademodule (LLMs)	337
8.1	Programm	338
8.2	Modul	340
8.3	Programmbibliothek	342
8.4	Modulbibliothek	344
8.5	Generierung der unterschiedlichen Programmtypen	346
8.5.1	Generierung als Programm	346
8.5.1.1	Nicht gemeinsam benutzbares Programm	346
8.5.1.2	Teilweise gemeinsam benutzbares Programm	347
8.5.1.3	Vollständig gemeinsam benutzbares Programm	350
8.5.2	Generierung als Modul	352
8.5.2.1	Nicht gemeinsam benutzbares Modul	352
8.5.2.2	Teilweise gemeinsam benutzbares Modul	354
8.5.2.3	Vollständig gemeinsam benutzbares Modul	356
8.5.3	Generierung als Programmbibliothek	358
8.5.4	Generierung als Modulbibliothek	359
8.6	Weitere Hinweise zu den Nutzungsmodellen	361
8.6.1	LLM-Format 1	361
8.6.2	Vorladen von PUBLIC-Slices über die ASHARE-Schnittstelle	361
9	Migration	363
9.1	Alte und aktuelle Begriffe	363
9.1.1	Alte Begriffe	363
9.1.2	Aktuelle Begriffe	365
9.2	Merkmale des aktuellen Binder-Lader-Systems	366
9.2.1	Bindelademodul (LLM) und BINDER	366
9.2.2	Dynamischer Bindelader DBL	367
9.2.3	DBL und BINDER	368
10	Anhang: Beschreibung der ISAM-Schlüssel	369
	Fachwörter	373

Literatur **383**

Stichwörter **385**

1 Einleitung

Ein Quellprogramm, das von einem Compiler (Assembler, C, COBOL, FORTRAN, PL1 usw.) übersetzt wurde, kann entweder Bindemodul-Format oder Bindelademodul-Format haben. Bindemodule (Object Module, OM) und Bindelademodule (Link and Load Module, LLM) sind Eingabeobjekte für das System **Binder-Lader-Starter**, das aus diesen Objekten ablauffähige Programme erzeugt.

Der **Binder** fügt ein übersetztes Quellprogramm mit anderen Bindemodulen oder Bindelademodulen zu einer ladefähigen Einheit zusammen. Dabei sucht er die zum Programmablauf erforderlichen Bindemodule oder Bindelademodule und verknüpft sie miteinander. Der Binder ergänzt dabei jedes Modul um die Adressen, die sich auf Felder außerhalb des betreffenden Moduls beziehen (Externverweise) und deshalb vom Compiler noch nicht eingetragen werden konnten. Dieser Vorgang wird Binden genannt.

Damit die beim Binden erzeugte Einheit ablaufen kann, muss sie von einem **Lader** in den Speicher geladen werden. Erst dann kann das Programm gestartet und ausgeführt werden.

Der Binder BINDER gehört zum System **Binder-Lader-Starter (BLS)**. In diesem stehen dem Benutzer außerdem noch folgende Funktionseinheiten zur Verfügung:

- das Subsystem BLSSERV mit der Funktionalität des dynamischen Bindeladers DBL,
- die optional zuschaltbare Sicherheitskomponente BLSSEC.

Binder BINDER

Der Binder BINDER bindet Module zu einer logisch und physisch strukturierten ladbaren Einheit zusammen. Diese Einheit bezeichnet man als **Bindelademodul** (Link and Load Module, **LLM**). Abgespeichert wird ein LLM vom BINDER als Bibliothekselement vom Typ L in einer Programmbibliothek oder in einer PAM-Datei.

Module, aus denen der BINDER ein LLM bindet, können sein:

- Bindemodule (OMs) und vorgebundene Bindemodule (Großmodule) aus einer Bindemodulbibliothek (OML), aus einer Programmbibliothek (Typ R) oder aus der temporären EAM-Bindemoduldatei,
- bereits gebundene oder von Compilern erzeugte LLMs aus einer Programmbibliothek (Typ L),
- bereits gebundene LLMs aus einer PAM-Datei (PAM-LLM).

BLSSERV mit dem Dynamischen Bindelader DBL

Der **dynamische Bindelader DBL** hat die Aufgabe, Module zu einer Ladeeinheit zu binden und diese zu laden.

Module, aus denen der DBL eine Ladeeinheit bindet, können sein:

- Bindelademodule (LLMs), die vom BINDER gebunden oder von Compilern erzeugt und in einer Programmbibliothek (Typ L) gespeichert wurden,
- Bindelademodule (LLMs), die vom BINDER gebunden und in einer PAM-Datei gespeichert wurden (PAM-LLMs, ab BLSSERV V2.5),
- Bindemodule (OMs), die von Compilern erzeugt und in einer Bindemodulbibliothek (OML), in einer Programmbibliothek (Typ R) oder in der temporären EAM-Bindemoduldatei gespeichert wurden.

Sicherheitskomponente BLSSEC

Wenn ein „sicheres System“ gefordert ist, kann die Sicherheitskomponente BLSSEC als Subsystem geladen werden. In diesem Fall führt das Binder-Lader-System vor dem Laden jedes Objektes eine Sicherheitsüberprüfung durch. Nur dann, wenn bei dieser Sicherheitsüberprüfung keine Probleme auftreten, wird das Objekt vom DBL geladen. Die Aktivierung des Subsystems BLSSEC führt bei allen Ladeaufrufen an das BLS zu geringerer Ladeperformance. BLSSEC sollte deshalb nach erfolgreicher Sicherheitsprüfung wieder entladen werden.

Die folgende Tabelle zeigt, welche Module von den einzelnen Funktionseinheiten verarbeitet bzw. erstellt werden. [Bild 1](#) zeigt das Zusammenwirken dieser Funktionseinheiten.

Modulart	Systembaustein		
	BINDER	DBL	BLSSEC
Bindemodul (OM)	ja	ja	ja
Bindelademodul (LLM)	ja	ja	ja
Bindelademodul in PAM-Datei (PAM-LLM)	ja	ja	ja
Vorgebundenen Modul (Großmodul)	ja	ja	ja
Programm (Lademodul)	nein	nein	ja

Der Binder BINDER ist ein Dienstprogramm. Der dynamische Bindelader DBL dagegen gehört zum Subsystem BLSSERV, das ein Teil des BS2000-Organisationsprogrammes ist. Er bietet seine Funktionalität über BS2000-Kommandos und über Programmschnittstellen an. Den Anstoß zur Ausführung eines geladenen Programmes gibt ein Starter, der Teil des Subsystems BLSSERV ist und für den Benutzer unsichtbar bleibt.

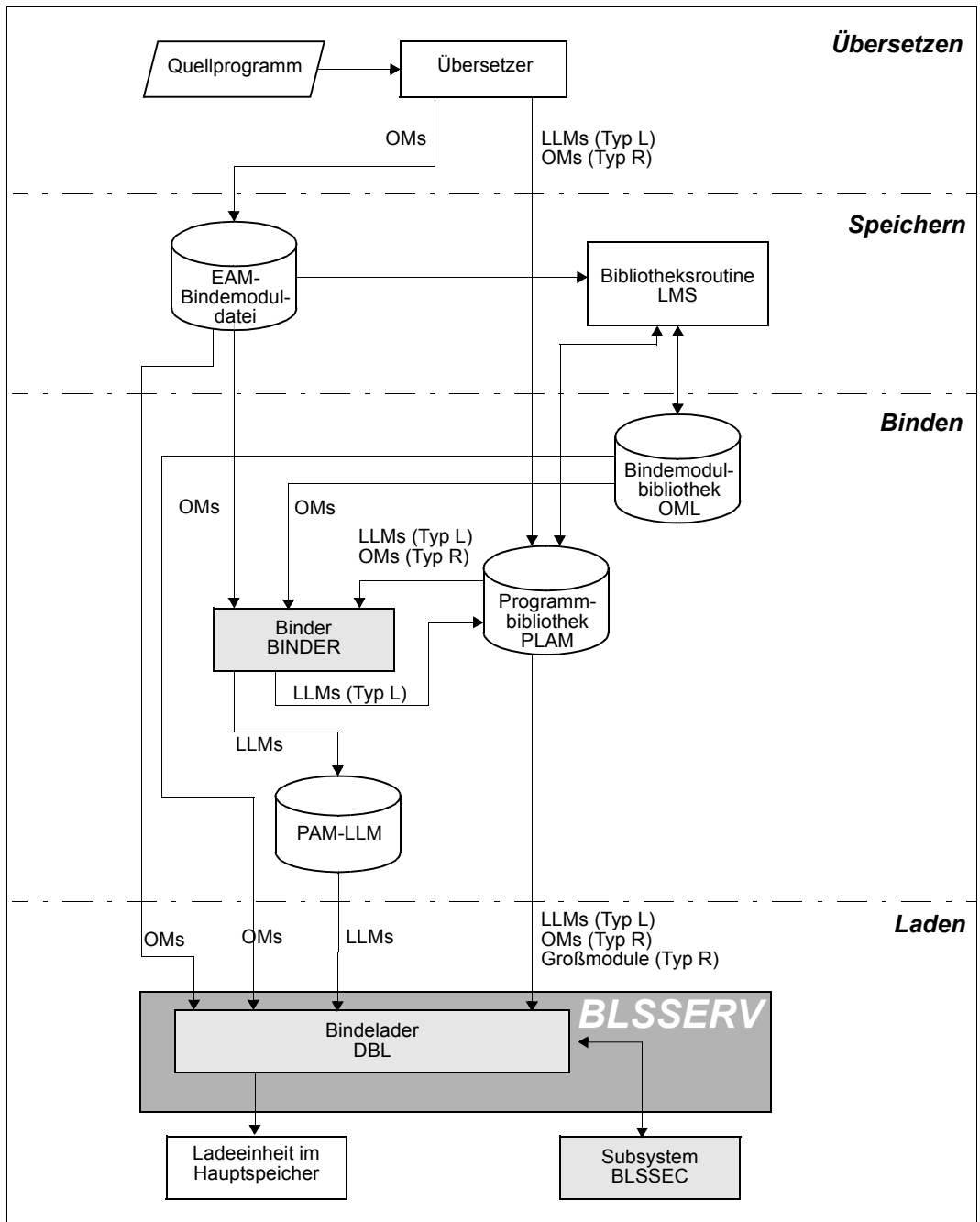


Bild 1: Zusammenwirken der Funktionseinheiten für Binden und Laden

1.1 Zielsetzung und Zielgruppen des Handbuchs

Das vorliegende Handbuch zum Binder BINDER wendet sich an den Software-Entwickler. Das Handbuch dient zum einen der Beschreibung der Leistung und Anwendung des BINDER, zum anderen dient es als Nachschlagewerk für die BINDER-Anweisungen und den Makroaufruf.

Für den Binder BINDER gibt es ein eigenes Taschenbuch. Es enthält die Formate aller BINDER-Anweisungen und den Makroaufruf BINDER. Das Tabellenheft ist als Übersicht für den mit dem BINDER vertrauten Benutzer bestimmt.

1.2 Konzept des Handbuchs

Die Beschreibung des gesamten Systems Binder-Lader-Starter (BLS) umfasst drei Benutzerhandbücher:

- Das vorliegende Handbuch beschreibt nur den Binder BINDER mit seinen Funktionen, Anweisungen und der Unterprogrammchnittstelle.
- Das Benutzerhandbuch „BLSSERV Bindelader-Starter“ [1] enthält die Beschreibung des dynamischen Bindeladers DBL.

Das vorliegende Handbuch enthält im Einzelnen:

- in den ersten vier Kapiteln die Struktur und den Inhalt von Bindelademodulen (LLMs), die Funktionen und die Ein-/Ausgabe des BINDER.
- in den nächsten drei Kapiteln den BINDER-Lauf, die Unterprogrammchnittstelle und die BINDER-Anweisungen in Form eines Nachschlageteils.
- ein Kapitel, in dem verschiedene Nutzungsmodelle für LLMs vorgestellt werden und auf welche Weise LLMs erzeugt werden müssen, um diesen Modellen zu entsprechen
- im Kapitel „Migration“ die wesentlichen Unterschiede zwischen dem alten Binder-/Lader-Konzept (bis BS2000 V9.5) und dem neuen System Binder-Lader-Starter (seit BS2000 V10.0). Es soll dem Benutzer eine Umstiegshilfe geben.
- die Meldungen des BINDER mit Bedeutung und Maßnahme sowie die wichtigsten Begriffe des Systems Binder-Lader-Starter.



Informationen zum Binder TSOSLNK und zum statischen Lader ELDE sowie zur Migration vom TSOSLNK zum BINDER finden Sie im Vorgängerhandbuch „BINDER V2.3“.

Readme-Datei

Funktionelle Änderungen der aktuellen Produktversion und Nachträge zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei.

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter <http://manuals.ts.fujitsu.com> zur Verfügung. Alternativ finden Sie Readme-Dateien auch auf der Softbook-DVD.

Informationen unter BS2000

Wenn für eine Produktversion eine Readme-Datei existiert, finden Sie im BS2000-System die folgende Datei:

```
SYSRME.<product>.<version>.<lang>
```

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D/E). Die Information können Sie am Bildschirm mit dem Kommando /SHOW-FILE oder mit einem Editor ansehen.

Das Kommando /SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product> zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

Ergänzende Produkt-Informationen

Aktuelle Informationen, Versions-, Hardware-Abhängigkeiten und Hinweise für Installation und Einsatz einer Produktversion enthält die zugehörige Freigabemittelung. Solche Freigabemittelungen finden Sie online unter <http://manuals.ts.fujitsu.com>.

1.3 Änderungen gegenüber dem Vorgängerhandbuch

Die vorliegende Ausgabe des Handbuchs „BINDER“ enthält gegenüber dem Vorgängerhandbuch („BINDER V2.3“) folgende Änderungen:

- Für PAM-LLM, in der Anweisung SAVE-LLM: Neuer Operand *BEHIND-SLICE(SLICE = <structured-name 1..32>) in der Struktur des Operanden LOAD-ADDRESS=*BY-SLICES für die Angabe der SLICE-ADDRESS.
- Anweisungen START-LLM-CREATION und MODIFY-LLM-ATTRIBUTES: Neuer Operand RELOAD-SLICE in der Struktur des Operanden BY-USER(AUTOMATIC-CONTROL) zum Nachladen eines bereits geladenen Slices.
- Unterstützung von X86-Prozessoren: Die Werte X86 und MIXX8 für HSI werden in der Liste ausgegeben.
- Anweisungen MODIFY-MAP-DEFAULTS und SHOW-MAP: Neuer Wert *X86 für den Operanden HSI-CODE zur Unterstützung von X86 im GENERATE-COPYRIGHT-MODULE

- SHOW-LIBRARY-ELEMENT-Anweisung: Wenn der SELECT-Operand auf *NAME-COLLISION gesetzt wird, sucht BINDER auch in dem gerade bearbeiteten LLM nach Duplikaten.
- Anweisung RENAME-SYMBOLS: Neuer Operand WARNING-MESSAGE in der Struktur des Operanden SYMBOL-OCCURRENCE zur Verbesserung der Diagnosefunktion
- Ausgaben des BINDER: Neues Listenbeispiel: Einfügen von Modulen mit Hilfe von einer SYSPLAMALT-Variable

Allgemeine Änderung

Die bisherige Bezeichnung BS2000/OSD-BC des BS2000-Grundausbaus ändert sich und lautet ab Version V10.0: BS2000 OSD/BC.

1.4 Darstellungsmittel

In diesem Handbuch werden folgende Darstellungsmittel verwendet:

- Literaturhinweise werden im Text in Kurztiteln und eckigen Klammern [] angegeben. Der vollständige Titel jeder Druckschrift, auf die verwiesen wird, ist im Literaturverzeichnis aufgeführt.
- In den Beispielen sind Benutzereingaben in **halbfetter Schreibmaschinenschrift** und Systemausgaben in *Schreibmaschinenschrift* wiedergegeben.



Dieses Symbol kennzeichnet wichtige Hinweise, die Sie unbedingt beachten sollten.

2 Einführung in den BINDER

Der BINDER bindet **Bindelademodule (LLMs)** aus Modulen zusammen und speichert sie als Bibliothekselemente (Elementtyp L) in einer Programmbibliothek.

Ein Bindelademodul (LLM) ist ein Objekt mit den Eigenschaften von vorgebundenen Modulen (Großmodulen), wie dynamisches Binden/Laden.

Wie ein Großmodul besteht ein LLM aus mehreren Modulen, die vom BINDER gebunden werden. Ein LLM umfasst ebenfalls die volle Funktionalität von Lademodulen, z.B. Überlagerungsstrukturen und Abbild im Speicherformat.

Im Folgenden sind die Hauptmerkmale des BINDER-Konzepts zusammengestellt:

1. Der BINDER verarbeitet LLMs nicht nur im Stapelbetrieb, sondern auch im Dialogbetrieb. Das bedeutet:
 - Jede Anweisung wird nach der Eingabe *sofort* verarbeitet.
 - Beim Erzeugen eines LLM kann sich der Benutzer zu jedem Zeitpunkt Informationen über den aktuellen Zustand des LLM ausgeben lassen. Der Benutzer kann dann entscheiden, ob in das aktuelle LLM weitere Module eingefügt werden oder ob Module entfernt werden sollen.
2. Der Benutzer kann LLMs, die in Programmbibliotheken gespeichert sind, ganz oder teilweise wieder verwenden. Das bedeutet:
 - In das aktuelle LLM kann ein vollständiges LLM oder ein „Sub-LLM“ eingefügt werden.
 - Ein LLM, das in einer Programmbibliothek gespeichert ist, kann geändert werden. Dabei können Module ausgetauscht, zusätzlich in das LLM aufgenommen oder innerhalb des LLM umgeordnet werden.
3. Während desselben BINDER-Laufs können *mehrere* LLMs erzeugt oder geändert werden. Der BINDER-Lauf wird also nicht nach dem Speichern des ersten LLM beendet. Darüberhinaus kann dasselbe LLM mehrmals gespeichert werden, z.B. mit verschiedenen Eigenschaften in derselben Programmbibliothek oder in verschiedenen Programmbibliotheken.

Der Aufbau und die Eigenschaften eines Bindelademoduls (LLM) sind im folgenden Abschnitt näher beschrieben.

2.1 Bindelademodule (LLMs)

Ein Bindelademodul (LLM) besteht aus einem oder mehreren **Modulen**. Es wird vom BINDER als Bibliothekselement vom **Elementtyp L** in einer Programmbibliothek gespeichert.

Module eines LLM können sein:

- Bindemodule (OMs), die von Compilern erzeugt werden,
- vorhandene Bindelademodule (LLMs) in einer Programmbibliothek (Elementtyp L).

Vorgebundene Module (Großmodule) haben dasselbe Format wie Bindemodule (OMs), die von Compilern erzeugt werden. Sie werden daher im Folgenden als Bindemodule (OMs) betrachtet.

Ein LLM hat folgende Merkmale:

- eine logische Struktur,
- eine physische Struktur,
- eine Identifikation,
- einen Inhalt.

2.2 Logische Struktur eines LLM

Die logische Struktur eines LLM wird durch eine Baumstruktur realisiert. Dieser Baum hat folgenden Aufbau (Bild 2):

1. Die Wurzel (root) bildet der **interne Name** (INTERNAL-NAME) des LLM. Auf den internen Namen wird in den Anweisungs- und Operandenbeschreibungen Bezug genommen.
2. Die Knoten (nodes) bilden Unterstrukturen, die als **Sub-LLMs** bezeichnet werden. Die Sub-LLMs sind hierarchisch in Stufen (levels) gegliedert.
3. Die Blätter (leaves) bilden Bindemodule (OMs) und leere Sub-LLMs, die in das LLM eingebunden werden.

Vorteile der logischen Struktur sind:

- Sub-LLMs eines LLM können einzeln eingefügt, entfernt oder ersetzt werden, da jedes Sub-LLM direkt angesprochen werden kann (siehe Seite 47ff).
- Beim Befriedigen von Externverweisen in einem LLM kann der Geltungsbereich begrenzt werden, z.B. auf das Sub-LLM der niedersten Stufe, da jedes Sub-LLM als getrennte Einheit durchsucht werden kann.

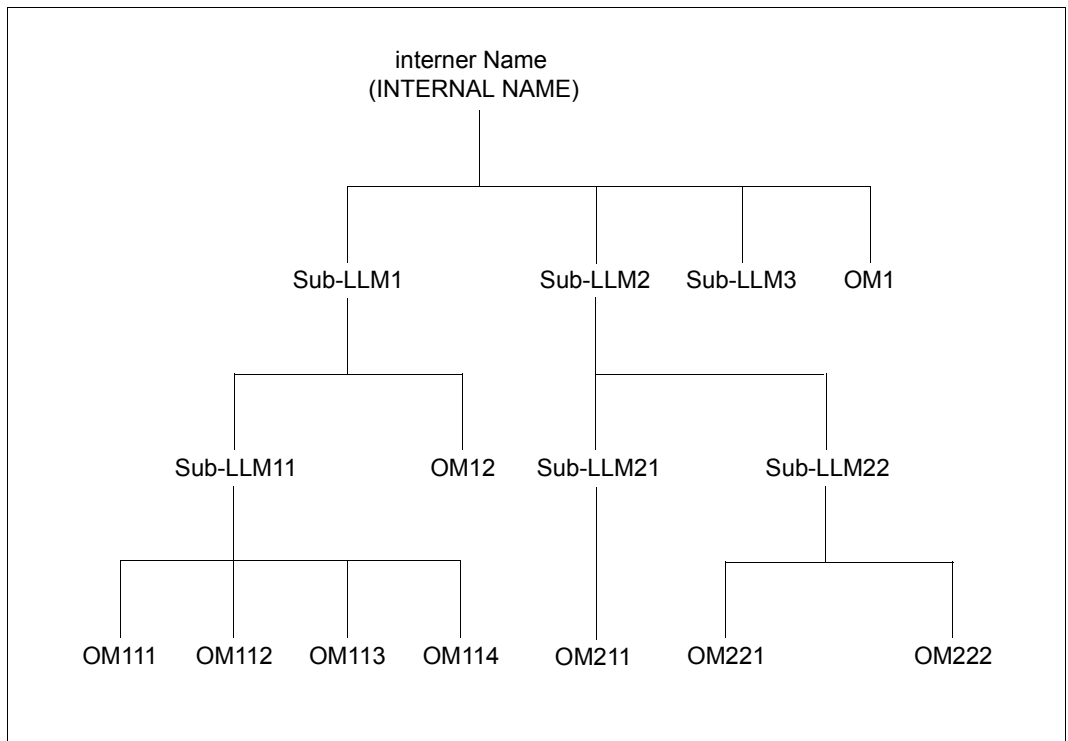


Bild 2: Beispiel für die logische Struktur eines LLM

2.3 Physische Struktur eines LLM

Die Bindemodule (OMs) eines LLM bestehen aus Programmabschnitten (CSECTs), die der dynamische Bindelader DBL unabhängig voneinander in den Hauptspeicher lädt. Bei einem LLM hat der Benutzer jedoch die Möglichkeit, die CSECTs eines oder mehrerer OMs des LLM zu einer Einheit zusammenzufassen, bei der alle CSECTs zusammenhängend in den Hauptspeicher geladen werden. Eine solche ladbare Einheit bezeichnet man als **Slice**. Die Slices bilden die physische Struktur eines LLM. Die Größe einer Slice ist nicht begrenzt. Man unterscheidet folgende drei physische Strukturen von LLMs:

- LLMs mit Einzel-Slice (Single Slice),
- LLMs mit Slices, die nach Attributen von CSECTs gebildet werden (Slices by Attributes),
- LLMs mit Slices, die vom Benutzer definiert werden (User defined Slices).

Den Typ der physischen Struktur legt der Benutzer beim Erzeugen eines LLM mit der Anweisung START-LLM-CREATION fest und er kann mit der Anweisung MODIFY-LLM-ATTRIBUTES geändert werden.

2.3.1 LLMs mit Einzel-Slice

Das LLM besteht aus einer einzigen Slice. Überlagerungsstrukturen (Overlays) sind nicht vorhanden.

2.3.2 LLMs mit nach Attributen gebildeten Slices

Den Daten und Instruktionen von CSECTs können bestimmte **Attribute** mitgegeben werden, die beim Binden und Laden ausgewertet werden.

Falls vom Benutzer verlangt, fasst der BINDER sämtliche CSECTs, die die gleichen Attribute oder die gleiche Kombination von Attributen haben, zu einer Slice zusammen. Eine einzelne CSECT kann dabei nicht auf mehrere Slices aufgespalten werden. Sie ist immer in einer Slice enthalten.

Nach folgenden Attributen bildet der BINDER Slices:

READ-ONLY

- Lesezugriff (READ-ONLY=YES)
Die CSECT kann nur gelesen werden. Dieses Attribut schützt die CSECT im Hauptspeicher vor dem Überschreiben.
- Lese- und Schreibzugriff (READ/WRITE) (READ-ONLY=NO)
Die CSECT kann gelesen und überschrieben werden.

RESIDENT

- Hauptspeicherresident (RESIDENT=YES)
Die CSECT wird in den Klasse-3-Speicher geladen und dort resident gehalten.
- Seitenwechselbar (PAGEABLE) (RESIDENT=NO)
Die CSECT ist seitenwechselbar.

Dieses Attribut ist nur für Shared Code des Systems relevant.

PUBLIC

- Gemeinsam benutzbar (PUBLIC=YES)
Die CSECT enthält Daten und Instruktionen, die gemeinsam benutzt werden können. Eine Slice, die aus CSECTs mit dem Attribut PUBLIC gebildet wurde, kann auch in einen Common Memory Pool (siehe Handbuch „BLSSERV Bindelader-Starter“ [1]) oder als unprivilegiertes Subsystem (siehe Handbuch „Einführung in die Systembetreuung“ [9]) geladen werden.
- Nicht gemeinsam benutzbar (PRIVATE, d.h. PUBLIC=NO)
Die CSECT enthält Daten und Instruktionen, die nur privat benutzt werden können.

RMODE

- Residenzmodus (RMODE=ANY)
Die CSECT kann unterhalb 16 Mbyte und oberhalb 16 Mbyte geladen werden.
- Residenzmodus (RMODE=24)
Die CSECT kann nur unterhalb 16 Mbyte geladen werden.

Diese Attribute können beliebig kombiniert werden. Maximal sind 16 Slices möglich. Die Namen der Slices werden vom BINDER festgelegt (siehe Seite 77).

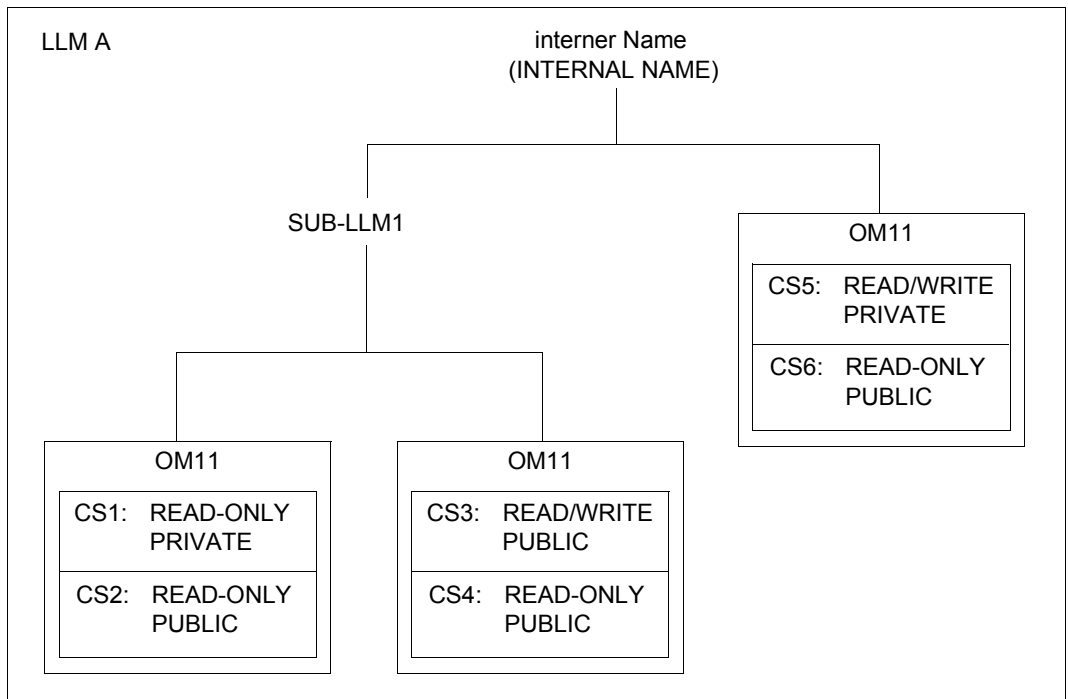
2.3.3 LLMs mit benutzerdefinierten Slices

Die physische Struktur des LLM wird vom Benutzer mit Anweisungen SET-USER-SLICE-POSITION festgelegt. Dabei können Überlagerungsstrukturen (Overlays) festgelegt werden (siehe Seite 68ff). Die CSECTs eines Bindemoduls (OM) können dabei nicht auf mehrere Slices aufgespalten werden. Sie sind immer in einer Slice enthalten.

Beispiel

Gegeben ist ein LLM A mit folgender *logischer* Struktur:

OM	CSECT	CSECT-Attribut
OM11	CS1 CS2	READ-ONLY und PRIVATE READ-ONLY und PUBLIC
OM12	CS3 CS4	READ/WRITE und PUBLIC READ-ONLY und PUBLIC
OM1	CS5 CS6	READ/WRITE und PRIVATE READ-ONLY und PUBLIC



Folgende *physische* Strukturen des LLM A können z.B. festgelegt werden (siehe [Bild 3](#)).

Einzel-Slice

Zur Einzel-Slice SLICE1 werden zusammengefasst:

- die CSECTs CS1 und CS2 von OM11,
- die CSECTs CS3 und CS4 von OM12 und
- die CSECTs CS5 und CS6 von OM1.

Die Attribute READ-ONLY und PUBLIC der CSECTs werden nicht berücksichtigt.

Nach Attributen gebildete Slices

Folgende Slices werden gebildet:

- SLICE1 aus allen CSECTs mit den Attributen READ-ONLY und PUBLIC.
Dies sind die CSECTs CS2 von OM11, CS4 von OM12 und CS6 von OM1.
- SLICE2 aus allen CSECTs mit den Attributen READ/WRITE und PUBLIC.
Dies ist die CSECT CS3 von OM12.
- SLICE3 aus allen CSECTs mit den Attributen READ-ONLY und PRIVATE.
Dies ist die CSECT CS1 von OM11.
- SLICE4 aus allen CSECTs mit den Attributen READ/WRITE und PRIVATE.
Dies ist die CSECT CS5 von OM1.

Benutzerdefinierte Slices

Folgende Überlagerungsstruktur wird festgelegt:

- Zur SLICE1 werden die CSECTs CS1 und CS2 von OM11 zusammengefasst.
SLICE1 soll die Root-Slice in der Überlagerungsstruktur sein.
- Zur SLICE2 werden die CSECTs CS3 und CS4 von OM12 zusammengefasst.
SLICE2 soll unmittelbar an SLICE1 anschließen.
- Zur SLICE3 werden die CSECTs CS5 und CS6 zusammengefasst.
SLICE3 soll SLICE2 überlagern.

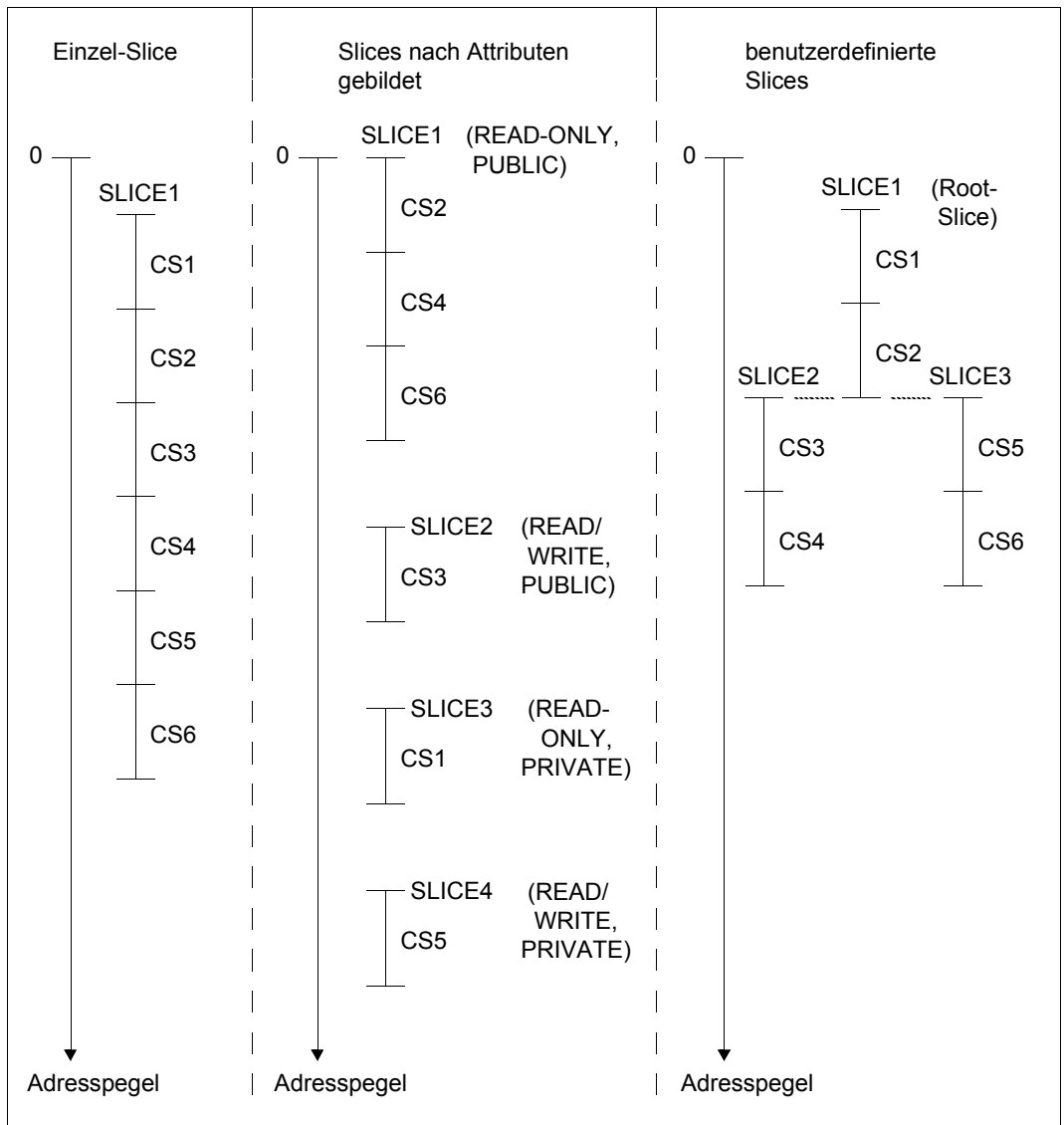


Bild 3: Beispiel für die physischen Strukturen eines LLM

2.4 Identifikation eines LLM

Jedes LLM kann angesprochen werden:

- durch einen internen Namen und eine interne Version, wenn er im Arbeitsbereich des BINDER erzeugt wird,
- durch einen Elementnamen und eine Elementversion, wenn er als Bibliothekselement in einer Programmbibliothek abgespeichert wird.

Sub-LLMs werden mit ihrem Pfadnamen angesprochen.

Interner Name und interne Version

Der interne Name kennzeichnet die Root in der Baumstruktur des LLM (siehe [Seite 17](#)). Er wird mit der Anweisung START-LLM-CREATION (Operand INTERNAL-NAME) festgelegt und kann mit der Anweisung MODIFY-LLM-ATTRIBUTES (Operand INTERNAL-NAME) geändert werden. Sollen Module im aktuellen LLM ersetzt werden (REPLACE-MODULES) oder entfernt werden (REMOVE-MODULES), müssen die internen Namen benutzt werden.

Zusätzlich zum internen Namen kann in der Anweisung START-LLM-CREATION eine interne Version (Operand INTERNAL-VERSION) angegeben werden. Sie kann mit der Anweisung MODIFY-LLM-ATTRIBUTES (Operand INTERNAL-VERSION) geändert werden.

Der interne Name und die interne Version werden beim Speichern des LLM in einer Programmbibliothek als Elementname und Elementversion übernommen, wenn in der Anweisung SAVE-LLM entsprechende Werte gesetzt sind.

Der interne Name und die interne Version werden beim Laden des LLM mit einer Meldung protokolliert.

Elementname und Elementversion

In einer Programmbibliothek wird ein Bibliothekselement gekennzeichnet durch den **Elementtyp** und die **Elementbezeichnung** (siehe Handbuch „LMS“ [3]). Für ein LLM, das als Element in einer Programmbibliothek abgespeichert wurde, ist der Elementtyp immer „L“. Die Elementbezeichnung setzt sich aus dem Elementnamen und der Elementversion des LLM zusammen.

Der Elementname und die Elementversion werden in der Anweisung SAVE-LLM festgelegt (Operanden ELEMENT und VERSION).

Pfadname

Sub-LLMs und damit auch die OMs innerhalb eines Sub-LLM werden mit ihren Pfadnamen angesprochen. Der Pfadname eines Sub-LLM oder OM besteht aus einer hierarchisch gegliederten Folge von Einzelnamen, die durch einen Punkt voneinander getrennt sind. Er hat folgendes Format:

:<path-name>: = internal-name.subLLM-level-1.subLLM-level-2. subLLM-level-n

Die Zeichenfolge ' ... ' ist dabei zu ersetzen durch:

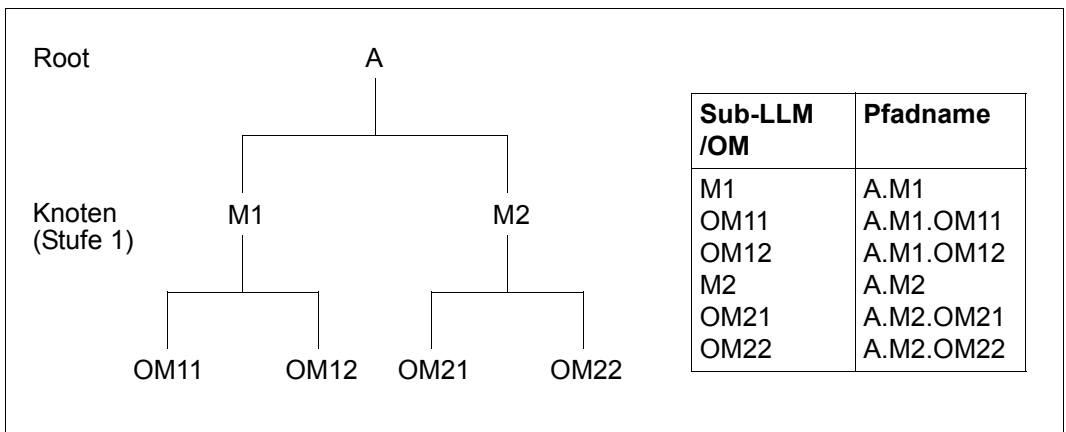
- eine Folge weiterer, durch Punkt voneinander getrennter subLLM-level oder
- eine leere Zeichenfolge (siehe „Abkürzungsmöglichkeiten“, [Seite 28f](#)).

Die Struktur des Pfadnamens ist identisch mit der logischen Struktur des LLM (siehe [Seite 17ff](#)). Dies bedeutet:

- Der erste Name „internal-name“ ist immer der interne Name (Root) des LLM-Baums.
- Der letzte Name „subLLM-level-n“ kennzeichnet den Knoten, der adressiert werden soll (Stufe n).
- Die dazwischen liegenden Namen kennzeichnen die Knoten, die zwischen Root und der letzten Stufe n liegen. Sie stellen die Verbindung zwischen dem ersten und letzten Namen her.

Beispiel

LLM mit Knoten Stufe 1



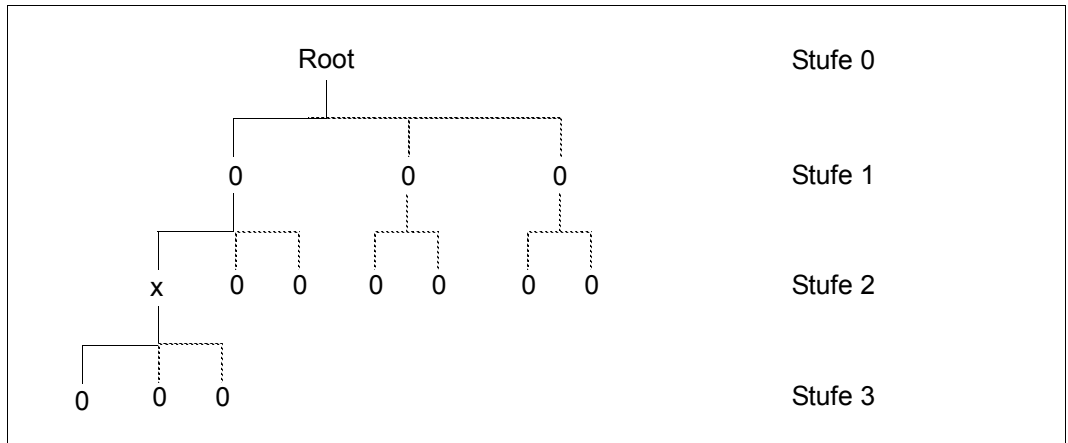
Abkürzung des Pfadnamens

Der Benutzer kann den Pfadnamen zu einem gewünschten Bindemodul (OM) abkürzen.

Beschreibung des Suchverfahrens

Um zu einem gewünschtem OM zu gelangen, benutzt der BINDER bei der Suche nach dem OM ein Suchverfahren, das als „backtracking“ bezeichnet wird.

Ausgehend von der Root wird der am weitesten links stehende Ast durchlaufen. Erreicht der BINDER eine Verzweigung, wählt er wieder den linken Ast.

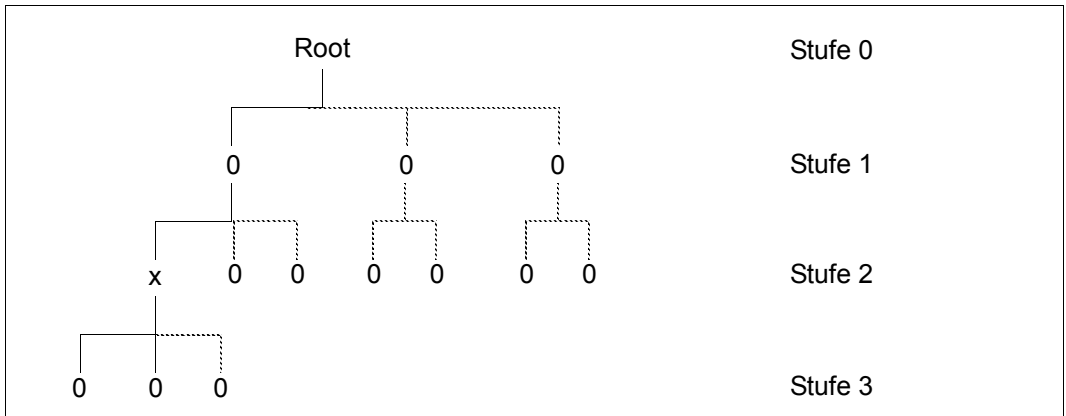


In dem Beispiel und in den folgenden bedeuten:

- 0 beliebiger Knoten
- Pfad, der bereits durchlaufen wird
- Pfad, der noch nicht betrachtet wurde
- x, y.. Knotenbezeichnung

Hat der BINDER die Stufe n (unterste Stufe im LLM, im Beispiel Stufe 3) erreicht und den gewünschten OM nicht gefunden, geht er zum Knoten auf die Stufe n-1 (im Beispiel Knoten x in Stufe 2) zurück. Dieses Zurückgehen wird als „backtracking“ bezeichnet.

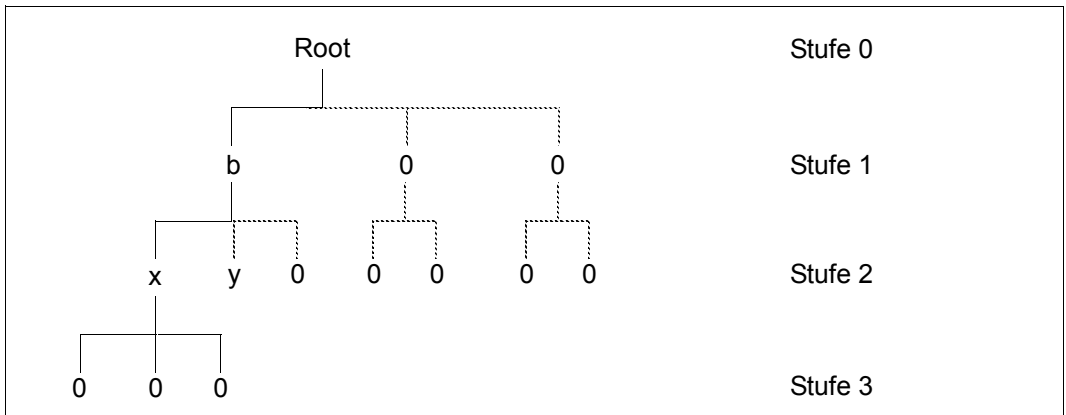
Von dort aus prüft der BINDER, ob von dem Knoten noch ein weiterer Ast bzw. weitere Äste abgehen. Wenn dies der Fall ist, setzt der BINDER die Suche in demjenigen Ast fort, der am nächsten an dem bisher durchlaufenen liegt („zweitlinkeste“ Ast).



Dieses Vorgehen wiederholt sich solange, bis entweder das gewünschte OM gefunden wird oder alle Verzweigungen eines Knotens abgearbeitet sind.

Wenn alle Verzweigungen eines Knotens abgearbeitet wurden und das OM nicht gefunden wurde, wird wieder um eine Stufe zurückgegangen und die nächste Verzweigung des auf der höheren Stufe liegenden Knotens wird untersucht.

Im folgenden Beispiel wird zu Knoten b auf Stufe 1 zurückgegangen (backtracking) und als Nächstes wird der Pfad von Knoten b zu Knoten y durchlaufen. Ist der Knoten y identisch mit dem gewünschten OM, wird die Suche erfolgreich beendet.



Ist der Knoten y nicht identisch mit dem gewünschten OM, wird der dritte Ast des Knotens b durchlaufen. Führt dieser Weg auch nicht zum Erfolg, wird zur Wurzel zurückgegangen und von dort aus das mittlere Sub-LLM nach den beschriebenen Verfahren durchlaufen.

Abkürzungsmöglichkeiten

Existiert nur ein OM gleichen Namens im gesamten LLM, so kann der Pfadname wie folgt abgekürzt werden:

```
:<path-name> = .subLLM-level-n
```

Existieren zwei oder mehrere OMs gleichen Namens im LLM, so muss zumindest ein Zwischenknoten im Pfadnamen enthalten sein, um den Weg zu dem gewünschten OM für den BINDER eindeutig zu beschreiben.

Folgende Formate sind möglich:

a)

```
:<path-name> = internal-name..subLLM-level-n
```

b)

```
:<path-name> = internal-name.subLLM-level-1..subLLM-level-n
```

Zu Format a):

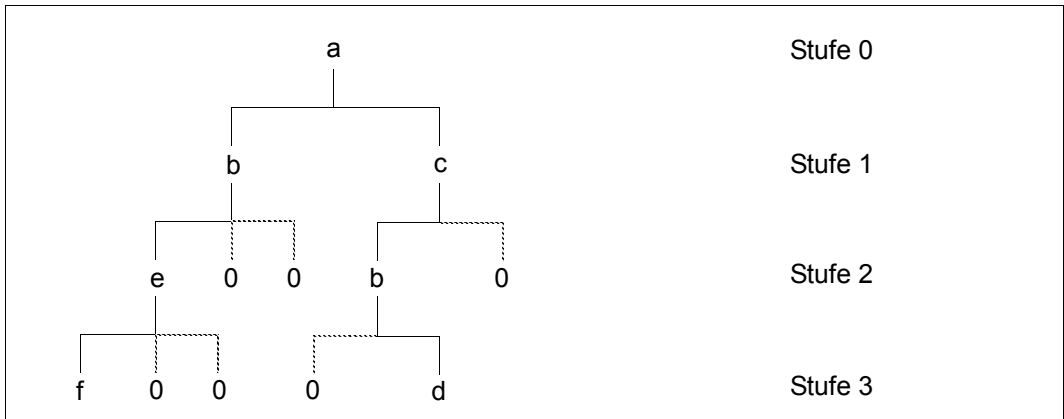
Ist in diesem Format `subLLM-level-n = subLLM-level-1`, so werden die zwei Punkte im Pfadnamen durch einen Punkt ersetzt, d.h.

`internal-name..subLLM-level-1 = internal-name.subLLM-level-1`.

Dies kann zu Fehlern führen, wenn

- ein Knoten erster Stufe und ein Knoten einer tieferen Stufe in einem anderen, weiter rechts liegenden Ast den gleichen Namen haben und
- der gewünschte Knoten, der kein Knoten erster Stufe ist, im weiter rechts liegenden Ast liegt.

Beispiel:



gewünschter Knoten	mögliche Pfadangaben		Suche erfolgreich
	abgekürzt	vollständig	
f	.f .b.f a.b.f	a.b.e.f a.b.e.f a.b.e.f	ja ja ja
d	.d a.c..d a..b.d	a.c.b.d a.c.b.d	ja ja nein

Die Angabe von „a..b.d“ würde in diesem Beispiel zu einem Fehler führen, da

- der Pfad „a..b.d“ in den Pfad „a.b.d“ umgesetzt wird,
- d kein Knoten im linken Unterbaum ist.

Die Umsetzung wird hier durchgeführt, weil der Knoten a ein direkter Nachfolger der Wurzel a im linken Unterbaum ist.

Zu Format b):

Dieses Format kann, abhängig von der Position des gewünschten Knotens, erweitert werden durch die Angabe weiterer Zwischenknoten.

2.5 Inhalt eines LLM

Ein LLM enthält mindestens die **Textinformation** und die **physische Strukturinformation**.

- **Textinformation (TXT)**

Die Textinformation besteht aus dem Code und den Daten der Module.

- **Physische Strukturinformation**

Die physische Strukturinformation enthält die Beschreibung der Slices, die zum LLM gehören.

Der Benutzer kann auswählen, ob das LLM zusätzlich folgende Informationen enthalten soll:

- **Externadressbuch (ESV)**

Das Externadressbuch (External Symbols Vector, ESV) enthält alle Programmdefinitionen und Referenzen. Ein LLM ohne Externadressbuch kann von BINDER nicht eingefügt oder geändert werden.

- **Relativierungsinformation (LRLD)**

Die Relativierungsinformation (LRLD) legt fest, wie Adressen beim Binden und Laden auf eine gemeinsame Bezugsadresse ausgerichtet (relativiert) werden. Ist die Relativierungsinformation vollständig, dann kann das LLM an eine beliebige Adresse geladen werden. Wenn die LRLD jedoch nur für unbefriedigte Externverweise existiert, muss das LLM an eine bestimmte Adresse geladen werden. Unbefriedigte Externverweise können jedoch aufgelöst werden. Fehlt die Relativierungsinformation ganz, können Adressen nicht relativiert werden und das LLM muss an eine festgelegte Adresse geladen werden. Ein LLM ohne vollständige LRLD kann von BINDER nicht eingefügt oder geändert werden.

- **Logische Strukturinformation**

Die Logische Strukturinformation enthält Angaben über die logische Struktur des LLM. Die Logische Strukturinformation eines LLM kann vollständig (LLM mit allen Sub-LLMs und Bindemodulen) oder nur teilweise (nur Bindemodule (Blätter), keine Sub-LLMs) vorhanden sein. Fehlt die Logische Strukturinformation ganz, dann kann das LLM von BINDER nicht eingefügt oder geändert werden.

- **Test- und Diagnoseinformation (LSD)**

Die Test- und Diagnoseinformation (LSD) wird von den Test- und Diagnosehilfen für das Testen auf Quellsprachenebene benötigt. Voraussetzung ist, dass beim Übersetzen des Quellprogramms entsprechende Compiler-Optionen gesetzt werden.

Das Abspeichern der LSD-Information ist nur dann möglich, wenn das Externadressbuch (ESV) auch abgespeichert wird. Fehlt die Test- und Diagnoseinformation, ist kein Testen auf Quellsprachenebene möglich.

- **Deskriptoren für Initialisierungs- und Beendigungsrouinen („Ini/Fini“-Information)**

Die Deskriptoren für Initialisierungs- und Beendigungsrouinen werden von neuen Compilern für objektorientierte Programmiersprachen benötigt. Sie sind notwendig, damit alle Initialisierungs- und Beendigungsrouinen eines LLMs vor bzw. nach dem Ablauf des eigentlichen Modulcodes in einer definierten Reihenfolge ausgeführt werden. Sie haben folgenden Inhalt:

- Typ der Routine: Initialisierung oder Beendigung
- Informationen zur Berechnung der Adresse der Routine
- Einen Identifikator, der vom Compiler generiert wird

Wenn ein LLM „Ini/Fini“-Information enthält, ist dies im logischen Root-Knoten des LLMs vermerkt.

Zum Laden von LLMs, die „Ini/Fini“-Information enthalten, ist BLSSERV erforderlich. D.h. sie können in BS2000-Versionen kleiner V3.0 nicht geladen werden, und in BS2000 V3.0 nur, wenn BLSSERV ab Version V2.0 eingesetzt wird.

Ob das LLM „Ini/Fini“-Information enthält, legt der Compiler fest.

Ob das LLM das Externadressbuch (ESV) und/oder die Relativierungsinformation (LRLD) enthalten soll, legt der Benutzer beim Speichern des LLM mit der Anweisung SAVE-LLM fest.

Logische Strukturinformationen und/oder LSD-Informationen kann der Benutzer auswählen beim:

- Erzeugen eines LLM (START-LLM-CREATION)
- Ändern eines LLM (START-LLM-UPDATE)
- Ändern der Merkmale eines LLM (MODIFY-LLM-ATTRIBUTES)
- Speichern eines LLM (SAVE-LLM)
- Einfügen von Modulen (INCLUDE-MODULES)
- Ersetzen von Modulen (REPLACE-MODULES)
- Befriedigen von Externverweisen durch Autolink (RESOLVE-BY-AUTOLINK)

Welche Zusatzinformationen ein LLM enthält, kann der Benutzer den Listen entnehmen, die mit der Anweisung SHOW-MAP oder beim Speichern des LLM mit der Anweisung SAVE-LLM ausgegeben werden (siehe Seite 141ff).

2.6 Randbedingungen für LLMs

Mit der Festlegung der Eigenschaften und des Inhalts eines LLM wird die Menge der später ausführbaren Aktionen eingeschränkt. Deshalb müssen beim Abspeichern des LLM einige Randbedingungen beachtet werden.

Im Folgenden sind die Werte einiger Operanden der Anweisung SAVE-LLM und die Folgen, die sich daraus für die spätere Verarbeitung des LLM ergeben, dargestellt.

<i>Operand</i>	REQUIRED-COMPRESSION
<i>Wert</i>	YES
<i>Folge</i>	Das Laden des LLM ist in allen BS2000-Systemen möglich.

<i>Operand</i>	LOGICAL-STRUCTURE	SYMBOL-DICTIONARY	RELOCATION-DATA
<i>Wert</i>	NONE	NO	UNRESOLVED-ONLY oder NO
<i>Folge</i>	<ol style="list-style-type: none"> 1. Ändern des LLM ist nicht mehr möglich. 2. Einfügen des LLM in ein anderes LLM ist nicht mehr möglich. 		

<i>Operand</i>	RELOCATION-DATA
<i>Wert</i>	UNRESOLVED-ONLY oder NO
<i>Folge</i>	Das LLM ist nicht verschiebbar, da die Adressen nicht relativiert werden können. Soll das LLM verschiebbar sein, muss unbedingt RELOCATION-DATA=YES angegeben werden.
<i>Wert</i>	NO
<i>Folge</i>	Unbefriedigte Externverweise können nicht aufgelöst werden. Sie sind nur dann auflösbar, wenn RELOCATION-DATA=YES oder RELOCATION-DATA=UNRESOLVED-ONLY angegeben wird.

<i>Operand</i>	TEST-SUPPORT
<i>Wert</i>	NO
<i>Folge</i>	Das Testen auf Quellsprachenebene (z.B. mit der Testhilfe AID, siehe Handbuch „AID“ [8]) ist nicht möglich.

Außerdem ist zu beachten, dass ein LLM mit benutzerdefinierten Slices *nicht* in ein anderes LLM eingefügt werden kann. Das Ändern eines LLM mit benutzerdefinierten Slices ist jedoch möglich.

LLM-Format

Folgende Tabelle zeigt die Abhängigkeiten zwischen LLM-Format, dem Wert des Operanden FOR-BS2000-VERSIONS, mit dem es ausgewählt wird, und den DBL- und BLSSERV-Versionen, mit denen es verarbeitet werden kann:

LLM-Format	Operand FOR-BS2000-VERSIONS	ladbar mit
1	FROM-V10	DBL in allen BS2000-Versionen
2	FROM-OSD-V1	DBL in allen BS2000-Versionen
3	FROM-OSD-V3	DBL ab BS2000/OSD-BC V3.0A ¹
4	FROM-OSD-V4	BLSSERV ab V2.0A (ab BS2000/OSD-BC V3.0A)

¹ Das LLM-Format 3, das X86-Code beinhaltet, kann ab BS2000/OSD-BC V9.0 geladen werden.

Außerdem existieren Abhängigkeiten zwischen dem LLM-Format und einigen Eigenschaften des LLMs. Folgende Tabelle zeigt, welches LLM-Format für bestimmte Eigenschaften eines LLMs erforderlich ist:

Eigenschaft	LLM-Format
REQUIRED-COMPRESSON=*YES	≥ 2
CONNECTION-MODE= *BY-RESOLUTION	1, 3, 4
RISC-Code vorhanden	≥ 3 ¹
RESOLUTION-SCOPE definiert	≥ 3 ²
LLM-Ausgabe in PAM-Datei	≥ 3
EEN-Namen (siehe Abschnitt „Symbolnamen“ auf Seite 106)	4 ³
Initialisierungs-/Beendigungsinformation	4 ⁴

- ¹ Wenn bei SAVE-LLM der Operand RELOCATION-DATA=*NO angegeben ist, kann das LLM auch mit Format 1 oder 2 abgespeichert werden
- ² Das LLM kann mit Format 1 oder 2 abgespeichert werden. Es kann jedoch mit BINDER nicht weiterverarbeitet werden. Die RESOLUTION-SCOPE-Angabe wird beim Speichern nicht berücksichtigt.
- ³ Das LLM kann mit Format 1 - 3 abgespeichert werden, wenn die EEN-Namen (siehe Abschnitt „Symbolnamen“ auf Seite 106) unterdrückt werden. Es kann jedoch mit BINDER nicht weiterverarbeitet werden. Das Unterdrücken der EEN-Namen kann auf eine der folgenden Arten vorgenommen werden:
Mit dem Operanden SYMBOL-DICTIONARY oder automatisch, falls alle EEN-Externverweise befriedigt sind und FOR-BS2000-VERSION ≠ FROM-OSD-V4. Die EEN-Namen (siehe Abschnitt „Symbolnamen“ auf Seite 106) werden in diesem Fall nicht abgespeichert.
- ⁴ Das LLM kann mit Format 1 - 3 abgespeichert werden. Es kann jedoch mit BINDER nicht weiterverarbeitet werden. Die „Ini/Fini“-Information wird abgespeichert, nicht jedoch die Information, dass das LLM „Ini/Fini“-Information enthält.

Falls eine der oben genannten Eigenschaften des LLMs ein Format mit einer höheren Nummer erfordert, als durch den Operanden FOR-BS2000-VERSIONS festgelegt wurde, gibt BINDER eine Fehlermeldung aus.

3 BINDER-Funktionen

Die BINDER-Funktionen sind in folgende Funktionsgruppen eingeteilt:

- Erzeugen, Ändern und Speichern eines LLM
- Einfügen, Entfernen und Ersetzen von Modulen
- Erzeugen der logischen Struktur eines LLM
- Erzeugen der physischen Struktur eines LLM
- Befriedigen von Externverweisen
- Behandlung von Symbolen
- Mischen von Modulen
- Ändern der Merkmale von LLMs und Modulen
- Anzeigefunktionen
- Steuern der Listenausgabe und Fehlerbehandlung

Bild 4 zeigt die Funktionsgruppen mit den zugehörigen BINDER-Anweisungen und ihr Zusammenwirken.

Die einzelnen Funktionsgruppen sind in den folgenden Abschnitten näher beschrieben.

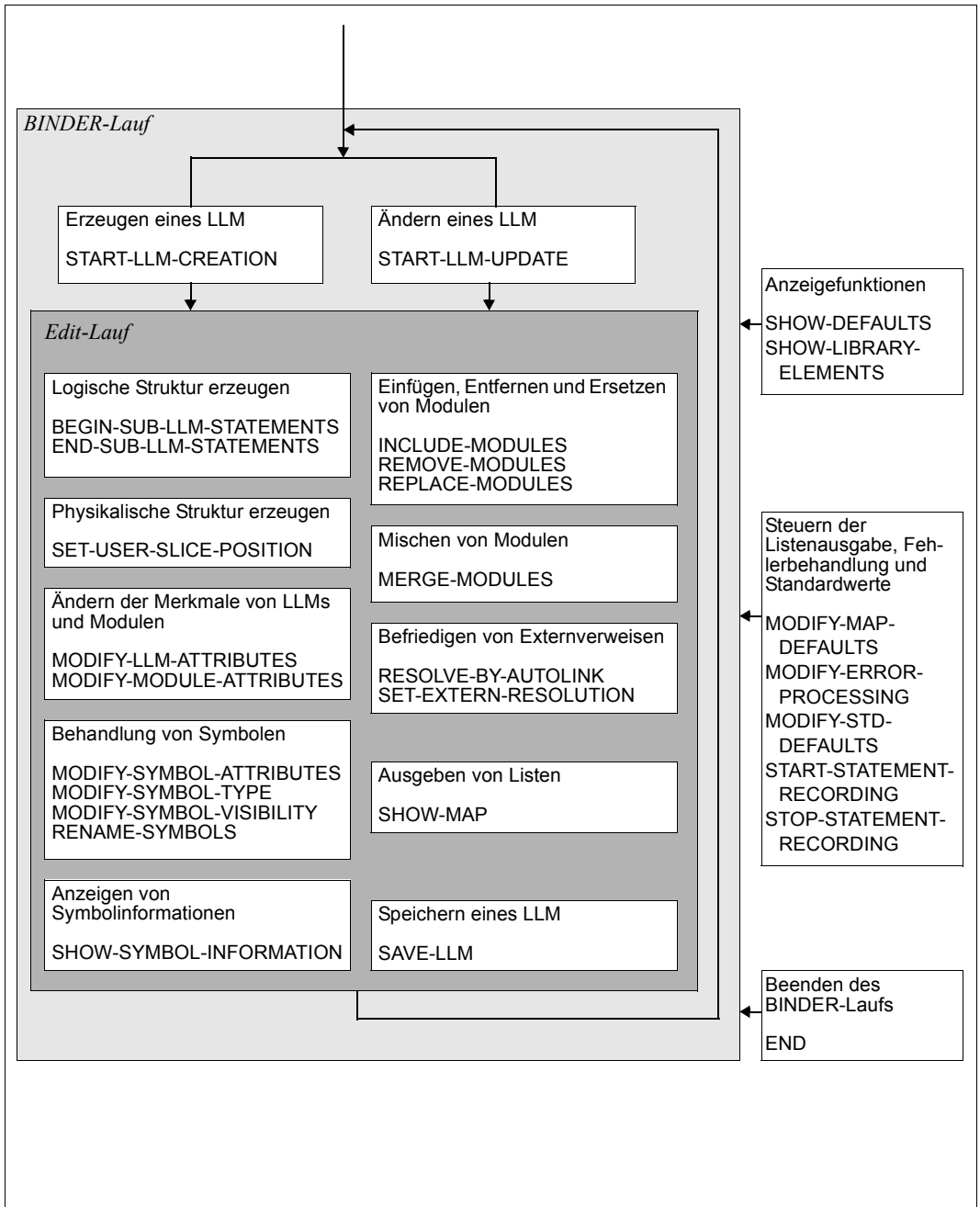


Bild 4: Übersicht über die BINDER-Funktionen

3.1 Erzeugen, Ändern und Speichern eines LLM

3.1.1 Erzeugen eines LLM

Ein LLM wird mit der Anweisung START-LLM-CREATION im **Arbeitsbereich** des BINDER erzeugt. Dabei werden seine Merkmale entsprechend den Angaben in der Anweisung START-LLM-CREATION festgelegt. Das erzeugte LLM im Arbeitsbereich des BINDER bezeichnet man als **aktuelles LLM**. Das aktuelle LLM kann anschließend im Arbeitsbereich verarbeitet werden. Z.B. können Module

- eingefügt werden (INCLUDE-MODULES),
- entfernt werden (REMOVE-MODULES),
- ersetzt werden (REPLACE-MODULES).

Die Verarbeitung des aktuellen LLM wird beendet, ohne dass es implizit abgespeichert wird. Es wird mit der Anweisung SAVE-LLM in einer Programmbibliothek als Element vom Typ L gespeichert.

Falls ein Element mit dem gleichen Elementnamen und der gleichen Elementversion bereits in der Programmbibliothek vorhanden ist und OVERWRITE=YES angegeben wurde, wird dieses Element überschrieben.

Beim Erzeugen eines LLM muss in der Anweisung START-LLM-CREATION der interne Name (INTERNAL-NAME) angegeben werden (siehe [Seite 138f](#)). Er wird als Elementname des LLM in der Programmbibliothek eingetragen, wenn beim Speichern des LLM in der Anweisung SAVE-LLM entsprechende Werte für den Elementnamen gewählt wurden (ELEMENT=*INTERNAL-NAME).

Wahlfrei können angegeben werden:

- die interne Version (INTERNAL-VERSION)
Sie wird als Elementversion des LLM in der Programmbibliothek eingetragen, wenn beim Speichern des LLM in der Anweisung SAVE-LLM entsprechende Werte für die Elementversion gewählt werden (VERSION=*INTERNAL-VERSION). Die Elementversion wird beim Laden des LLM mit einer Meldung protokolliert.
- die physische Struktur (SLICE-DEFINITION)
Es können LLMs aus Einzel-Slices, LLMs aus nach Attributen gebildeten Slices oder LLMs aus benutzerdefinierten Slices erzeugt werden (siehe [Seite 18ff](#)).
- Copyright-Information (COPYRIGHT)
Die Copyright-Information besteht aus einem Text und der Jahreszahl, die in das LLM eingetragen werden. Sie wird beim Laden des LLM mit einer Meldung protokolliert.

- Vereinbarungen über die Strukturinformation und LSD-Information (INCLUSION-DEFAULTS).



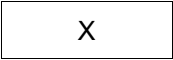
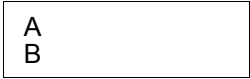
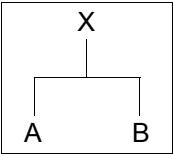

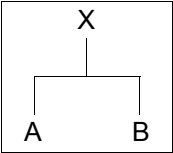
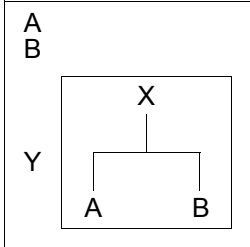
Mit dem Unteroperanden LOGICAL-STRUCTURE kann der Benutzer festlegen, ob beim Einfügen oder Ersetzen von Modulen die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird oder ob Unterstrukturen (Sub-LLMs) unberücksichtigt bleiben. Im letzteren Fall werden beim Einfügen oder Ersetzen von Modulen nur die *Bindemodule (OMs)* aus den Modulen übernommen.

Mit dem Unteroperanden TEST-SUPPORT kann der Benutzer festlegen, ob beim Einfügen, Entfernen oder Ersetzen von Modulen die Test- und Diagnoseinformation (LSD) aus den Modulen übernommen wird.

Die Werte der Operanden LOGICAL-STRUCTURE und TEST-SUPPORT gelten bis zur nächsten Anweisung START-LLM-CREATION oder START-LLM-UPDATE (siehe Edit-Lauf, [Seite 138f](#)). Sie können in allen Anweisungen INCLUDE-MODULES, RESOLVE-BY-AUTOLINK und REPLACE-MODULES als Standardwerte verwendet werden. Die Werte können mit der Anweisung MODIFY-LLM-ATTRIBUTES geändert werden.

Beispiel

Erzeugen und Speichern eines LLM

<u>Anweisungen</u>	<u>aktuelles LLM</u> (Arbeitsbereich)	<u>Programmbibliothek</u> (LIB1)
(1) START-BINDER		
(2) START-LLM-CREATION INTERNAL-NAME=X		
(3) INCLUDE-MODULES LIBRARY=LIB1, ELEMENT=(A,B)		
(4) SAVE-LLM LIBRARY=LIB1, ELEMENT=Y		
(5) END		

Bedeutung

- (1) Aufruf des BINDER. Der BINDER richtet einen Arbeitsbereich ein.
- (2) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt. Der interne Name X bildet die Wurzel in der logischen Struktur des LLM.
- (3) Die Bindemodule A und B werden aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt.
- (4) Das erzeugte LLM wird als Element mit dem Elementnamen Y in der Programmbibliothek LIB1 gespeichert.
- (5) Ende des BINDER-Laufs.

3.1.2 Ändern eines LLM

Ein LLM, das in einer Programmbibliothek als Element vom Typ L gespeichert ist, wird mit der Anweisung START-LLM-UPDATE geändert. Dabei wird das LLM aus der Programmbibliothek in den Arbeitsbereich des BINDER gelesen. Nach dem Einlesen wird das LLM zum **aktuellen LLM**, d.h. es hat den gleichen Zustand wie vor dem Speichern in die Programmbibliothek. Das aktuelle LLM kann anschließend im Arbeitsbereich verarbeitet werden. Z.B. können

- Module eingefügt werden (INCLUDE-MODULES),
- Module entfernt werden (REMOVE-MODULES),
- Module ersetzt werden (REPLACE-MODULES),
- Merkmale des LLMs geändert werden (MODIFY-LLM-ATTRIBUTES),
- Merkmale von Modulen geändert werden (MODIFY-MODULE-ATTRIBUTES).

Mit dem Unteroperanden LOGICAL-STRUCTURE kann der Benutzer festlegen, ob beim Einfügen, Entfernen oder Ersetzen von Modulen die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird oder ob Unterstrukturen (Sub-LLMs) unberücksichtigt bleiben. Im letzteren Fall werden beim Einfügen oder Ersetzen von Modulen nur die *Bin-demodule* (OMs) aus den Modulen übernommen.

Mit dem Unteroperanden TEST-SUPPORT kann der Benutzer festlegen, ob beim Einfügen oder Ersetzen von Modulen die Test- und Diagnoseinformation (LSD) aus den Modulen übernommen wird.

Als Standardwerte für die Operanden LOGICAL-STRUCTURE und TEST-SUPPORT werden die Werte angenommen, die beim Erzeugen des LLM mit der Anweisung START-LLM-CREATION festgelegt wurden.

Die Werte der Operanden LOGICAL-STRUCTURE und TEST-SUPPORT gelten bis zur nächsten Anweisung START-LLM-CREATION oder START-LLM-UPDATE (siehe Edit-Lauf, [Seite 138f](#)). Sie können in allen Anweisungen INCLUDE-MODULES, RESOLVE-BY-AUTOLINK und REPLACE-MODULES als Standardwerte verwendet werden. Die Standardwerte für LOGICAL-STRUCTURE und TEST-SUPPORT können mit der Anweisung MODIFY-LLM-ATTRIBUTES geändert werden.

Die Verarbeitung des aktuellen LLM wird beendet, ohne dass es implizit abgespeichert wird. Es wird mit der Anweisung SAVE-LLM in einer Programmbibliothek als Element vom Typ L gespeichert. Falls das neue Element den gleichen Elementnamen und die gleiche Elementversion behält und OVERWRITE=YES angegeben wurde, wird das bisherige Element in der Programmbibliothek überschrieben.

Beispiel

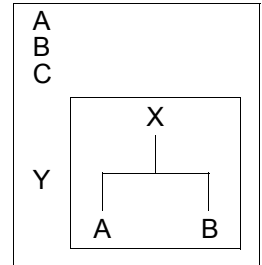
Ändern und Speichern eines LLM

Anweisungen

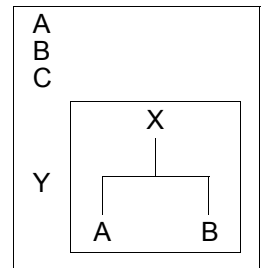
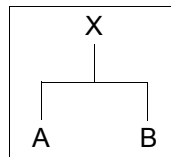
aktuelles LLM
(Arbeitsbereich)

Programmbibliothek
(LIB1)

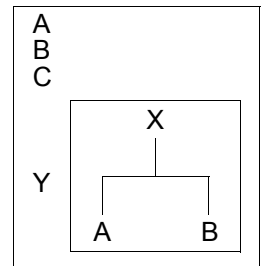
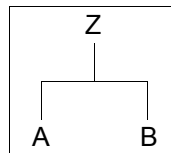
(1) START-BINDER



(2) START-LLM-UPDATE
LIBRARY=LIB1
ELEMENT=Y

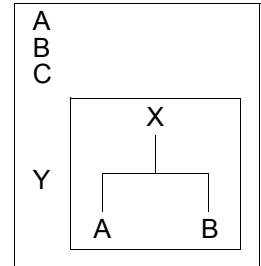
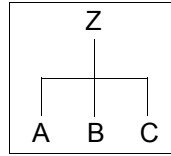


(3) MODIFY-LLM-ATTRIBUTES
INTERNAL-NAME=Z

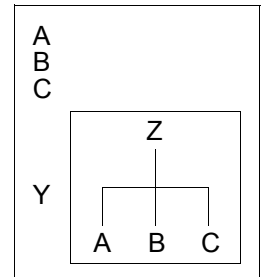
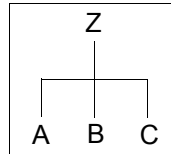


Anweisungenaktuelles LLM
(Arbeitsbereich)Programmbibliothek
(LIB1)

(4) INCLUDE-MODULES
ELEMENT=C



(5) SAVE-LLM
ELEMENT=Y,
OVERWRITE=YES



(6) END

Bedeutung

- (1) Aufruf des BINDER. Der BINDER richtet einen Arbeitsbereich ein.
- (2) Ein LLM, das als Element mit dem Elementnamen Y in der Programmbibliothek LIB1 gespeichert ist, soll geändert werden. Das LLM wird dazu in den Arbeitsbereich gelesen.
- (3) Der interne Name X des LLM wird geändert. Der neue interne Name ist Z.
- (4) Das Bindemodul C wird aus der Programmbibliothek LIB1 gelesen und in das LLM eingefügt. Der Bibliotheksname LIB1 wird aus der vorhergehenden Anweisung START-LLM-UPDATE übernommen (aktuelle Programmbibliothek).
- (5) Das geänderte LLM wird als Element mit dem *gleichen* Elementnamen Y in der Programmbibliothek LIB1 gespeichert und überschreibt das bisherige Element.
- (6) Ende des BINDER-Laufs.

3.1.3 Ändern der Merkmale eines LLM

Merkmale eines LLM, die beim Erzeugen des LLM mit der Anweisung START-LLM-CREATION festgelegt wurden, können mit der Anweisung MODIFY-LLM-ATTRIBUTES geändert werden. Die Anweisung ändert folgende Merkmale:

- den internen Namen (INTERNAL-NAME),
- die interne Version (INTERNAL-VERSION),
- den Typ der physischen Struktur des LLM (SLICE-DEFINITION),
- die Copyright-Information (COPYRIGHT),
- Vereinbarungen über die Strukturinformation (LOGICAL-STRUCTURE),
- Vereinbarungen über Test- und Diagnoseinformationen (LSD) (TEST-SUPPORT).

Folgende Änderungen des Typs der physischen Struktur sind erlaubt:

1. LLM mit nach Attributen gebildeten Slices in ein LLM mit Einzelslice
2. LLM mit Einzelslice in ein LLM mit nach Attributen gebildeten Slices
3. LLM mit nach Attributen gebildeten Slices in ein LLM mit nach *anderen* Attributen gebildeten Slices
4. LLM mit benutzerdefinierten Slices in ein LLM mit benutzerdefinierten Slices und geänderten Werten für AUTOMATIC-CONTROL und EXCLUSIVE-SLICE-CALL.

3.1.4 Speichern eines LLM

Das aktuelle LLM, das im Arbeitsbereich mit einer Anweisung START-LLM-CREATION erzeugt oder START-LLM-UPDATE geändert wurde, wird mit der Anweisung SAVE-LLM als Element vom Typ L in einer Programmbibliothek gespeichert. Ein mit der Anweisung START-LLM-UPDATE geändertes LLM überschreibt normalerweise das bisherige Element, wenn das neue Element den gleichen Elementnamen behält. Mit dem Operanden OVERWRITE kann das Überschreiben jedoch verboten werden. Der Benutzer erhält dann eine Fehlermeldung, wenn er versucht, das Element unter dem gleichen Namen und mit der gleichen Versionsnummer abzuspeichern. Wenn OVERWRITE=YES angegeben wurde und das Element noch nicht existiert hat, erhält der Nutzer dagegen keine Fehlermeldung. Für das Speichern des LLM kann man den Namen der Programmbibliothek entweder explizit angeben oder man kann die **aktuelle** Programmbibliothek auswählen. Die aktuelle Programmbibliothek ist die Bibliothek, auf die sich die letzte vorhergehende Anweisung START-LLM-UPDATE oder SAVE-LLM bezogen hat.

Elementname und Elementversion, die das LLM beim Speichern in der Programmbibliothek erhalten soll, können entweder explizit angegeben werden oder es wird der aktuelle Name und die aktuelle Version angenommen. Den aktuellen Namen und die aktuelle Version übernimmt der BINDER aus der letzten Anweisung SAVE-LLM, die nach der letzten Anweisung START-LLM-CREATION oder START-LLM-UPDATE angegeben wurde.

Wurde keine entsprechende Anweisung SAVE-LLM angegeben, übernimmt der BINDER als Elementname und Elementversion

- den Elementnamen und die Elementversion aus der letzten Anweisung START-LLM-UPDATE oder
- den *internen Namen* bzw. die *interne Version* aus der letzten Anweisung START-LLM-CREATION.

Mit dem Operanden FOR-BS2000-VERSIONS kann die BS2000-Version festgelegt werden, in der das LLM durch den DBL ladbar sein soll. Bei Angabe von FOR-BS2000-VERSIONS=*FROM-V10 wird das LLM mit *Format 1* abgespeichert. Bei Angabe von FOR-BS2000-VERSIONS=FROM-OSD-V1 wird das LLM mit *Format 2* abgespeichert, wenn der Benutzer REQUIRED-COMPRESSSION=*YES angegeben hat oder das LLM Slices hat, die nach dem Attribut PUBLIC gebildet wurden.

LLMs im Format 2 bieten den Vorteil, dass ihre Text-Information komprimiert werden kann und die Befriedigung von Externverweisen der PRIVATE-Slice durch die PUBLIC-Slice ein besseres Performance-Verhalten hat.

Hinweis

Mit dem Operanden REQUIRED-COMPRESSSION kann die Komprimierung der Text-Information (TXT) in LLMs veranlasst werden.

Wie Namenskonflikte zu behandeln sind, wird mit NAME-COLLISION gesteuert (siehe auch [Seite 102ff](#)).

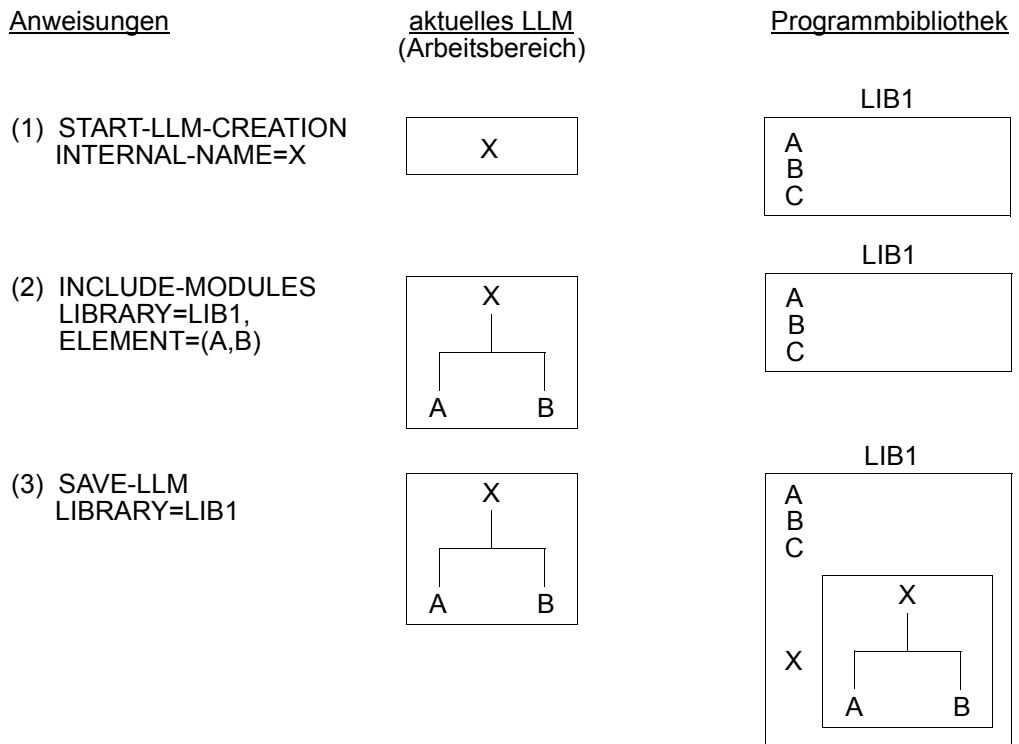
Die Standardwerte für die Operanden FOR-BS2000-VERSIONS, OVERWRITE, REQUIRED-COMPRESSSION und NAME-COLLISION können mit der Anweisung MODIFY-STD-DEFAULTS geändert werden.

Beim Speichern des LLM kann der Benutzer die Zusatzinformationen (siehe [Seite 30f](#)) festlegen, die übernommen werden. Dabei sollten aber einige Randbedingungen (siehe [Seite 32f](#)) beachtet werden. Folgende Zusatzinformationen können ausgewählt werden:

- Externadressbuch (ESV) (Operand SYMBOL-DICTIONARY)
- Relativierungsinformation (LRLD) (Operand RELOCATION-DATA)
- Logische Strukturinformation (Operand LOGICAL-STRUCTURE) und
- Test- und Diagnoseinformation (LSD) (Operand TEST-SUPPORT).

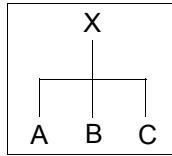
Beispiel

Speichern eines LLM in verschiedenen Programmbibliotheken

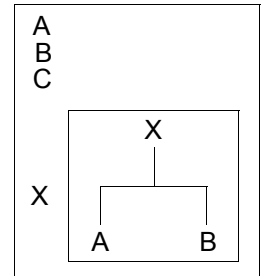


Anweisungen

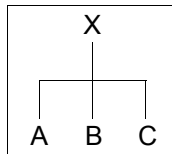
(4) INCLUDE-MODULES
ELEMENT=C

aktuelles LLM
(Arbeitsbereich)Programmbibliothek

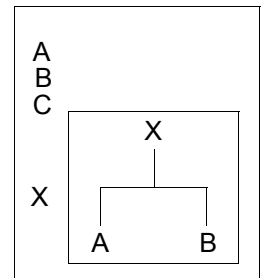
LIB1



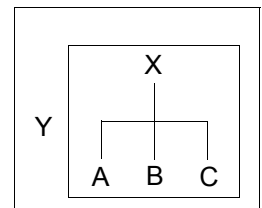
(5) SAVE-LLM
LIBRARY=LIB2,
ELEMENT=Y



LIB1



LIB2



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die Bindemodule A und B werden aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt.
- (3) Das aktuelle LLM wird in der Programmbibliothek LIB2 gespeichert. Als Elementname wird der interne Name X aus der letzten Anweisung START-LLM-CREATION übernommen.

- (4) Das Bindemodul C wird aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt. Der Bibliotheksname LIB1 wird aus der letzten vorhergehenden Anweisung INCLUDE-MODULES übernommen.
- (5) Das aktuelle LLM wird in der Programmbibliothek LIB2 als Element mit dem Elementnamen Y gespeichert.

3.2 Einfügen, Entfernen und Ersetzen von Modulen

3.2.1 Einfügen von Modulen

Mit der Anweisung INCLUDE-MODULES fügt der BINDER Module in das aktuelle LLM im Arbeitsbereich ein. Als Module können sowohl Bindemodule (OMs) als auch LLMs eingefügt werden. Es ist aber nicht möglich, LLMs mit benutzerdefinierten Slices bzw. LLMs ohne Relativierungsinformation, ohne Logische Strukturinformation oder ohne Externadressbuch einzufügen. Wurde die gesamte Strukturinformation beim Speichern des LLM übernommen (Operand LOGICAL-STRUCTURE = WHOLE-LLM), dann können entweder LLMs vollständig eingefügt oder einzelne Sub-LLMs ausgewählt werden. Sub-LLMs werden über ihren Pfadnamen ausgewählt (Operand ELEMENT=...(...,SUB-LLM=...)...). Die Slice-Struktur des Eingabe-LLMs hat keinen Einfluss auf die Slice-Bildung des gerade bearbeiteten LLMs.

Die Eingabequelle kann sein:

- für Bindemodule eine Programmbibliothek (Elementtyp R), eine Bindemodulbibliothek (OML) oder die EAM-Bindemoduldatei,
- für LLMs bzw. Sub-LLMs eine Programmbibliothek (Elementtyp L).

Die Eingabequelle kann entweder explizit angegeben werden oder es kann die aktuelle Eingabequelle übernommen werden. Die aktuelle Eingabequelle ist die Bibliothek bzw. EAM-Bindemoduldatei, aus der das letzte Modul geholt wurde (mit einer Anweisung START-LLM-UPDATE, INCLUDE-MODULES oder REPLACE-MODULES).

Aus der gewählten Eingabequelle können alle Module oder einzelne explizit angegebene Module in das LLM eingefügt werden.

Für die Module kann festgelegt werden, ob nur LLMs, nur OMs oder beide Typen gesucht werden sollen. Werden LLMs *und* OMs in einer Programmbibliothek gesucht, kann bei Namensgleichheit die Priorität über den Operanden TYPE festgelegt werden. Standardmäßig hat ein LLM eine höhere Priorität als ein OM.

Module in einer Programmbibliothek werden nach ihrer Elementversion ausgewählt. Wenn keine Elementversion explizit angegeben wurde, wird das Element mit der *höchsten* Elementversion angenommen (siehe Handbuch „LMS“ [3]). Der Name des logischen Knotens

(Operand NAME), der durch das Einfügen erzeugt wird, kann entweder der interne Name des Moduls (LLM/OM), der externe Name des Moduls oder ein vom Benutzer vergebener Name sein.

Mit dem Operanden RUN-TIME-VISIBILITY kann festgelegt werden, ob ein Modul als Laufzeitmodul betrachtet werden soll oder nicht. Bei Angabe von RUN-TIME-VISIBILITY=YES werden alle Symbole dieses Moduls beim Abspeichern des LLM maskiert, bereits befriedigte Externverweise bleiben jedoch befriedigt.

Zur Befriedigung von Externverweisen beim Einfügen oder Ändern dieses LLM werden diese Symbole während des BINDER-Laufs wieder sichtbar gemacht.

Beim Einfügen von Modulen hat der Benutzer auch die Möglichkeit, die Behandlung von Namenskonflikten zu steuern.

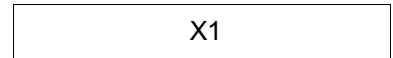
Beispiel 1

Im Arbeitsbereich werden nacheinander verschiedene LLMs erzeugt. In das aktuelle LLM werden LLMs und OMs eingefügt, die aus der Programmbibliothek LIB1 geholt werden. Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert.

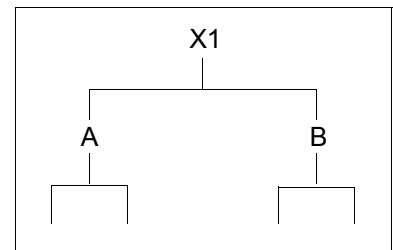
Die Programmbibliothek LIB1 mit dem Dateikettungsnamen EXLINK enthält die OMs A und B und die LLMs A ,B, Y und Z.

Anweisungenaktuelles LLM
(Arbeitsbereich)

(1) START-LLM-CREATION
INTERNAL-NAME=X1

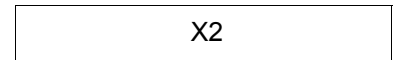


(2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B)

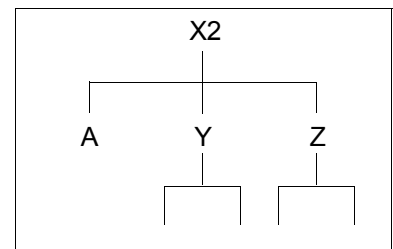


(3) SAVE-LLM LIBRARY=LIB1

(4) START-LLM-CREATION
INTERNAL-NAME=X2

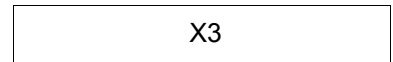


(5) INCLUDE-MODULES
LIBRARY=*LINK(EXLINK),
ELEMENT=(A,Y,Z),
TYPE=(R,L)

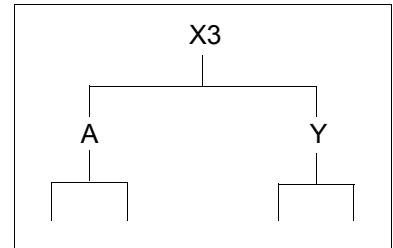


(6) SAVE-LLM LIBRARY=LIB1

(7) START-LLM-CREATION
INTERNAL-NAME=X3

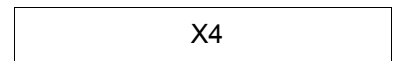


(8) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,Y),
TYPE=L

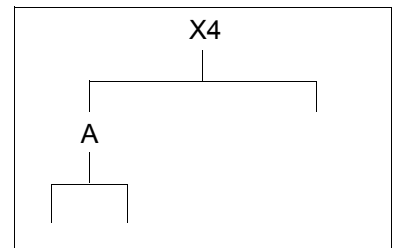


(9) SAVE-LLM LIBRARY=LIB1

(10) START-LLM-CREATION
INTERNAL-NAME=X4

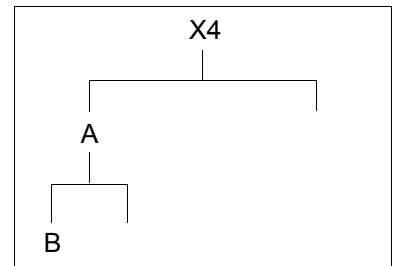


(11) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=A,TYPE=L



(12) INCLUDE-MODULES ELEMENT=B,
TYPE=R, PATH-NAME=X4.A

(13) SAVE-LLM LIBRARY=LIB1



Bedeutung

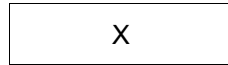
- (1) Ein LLM mit dem internen Namen X1 wird im Arbeitsbereich erzeugt.
- (2) Die LLMs A und B werden aus der Programmbibliothek LIB1 in das aktuelle LLM eingefügt, da standardmäßig bei Elementen mit gleichen Namen eine höhere Priorität für LLMs festgelegt ist.
- (3) Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert. Als Elementname wird der interne Name X1 aus der letzten Anweisung START-LLM-CREATION übernommen.
- (4) Ein LLM mit dem internen Namen X2 wird im Arbeitsbereich erzeugt.
- (5) Aus der Programmbibliothek LIB1, die mit dem Dateikettungsnamen EXLINK angesprochen wird, werden folgende Module in das aktuelle LLM eingefügt:
 1. Das Bindemodul A, da der Operand TYPE=(R,L) bei Elementen mit dem gleichen Namen eine höhere Priorität für Bindemodule festlegt.
 2. Die LLMs Y und Z.
- (6) Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert. Als Elementname wird der interne Name X2 aus der letzten Anweisung START-LLM-CREATION übernommen.
- (7) Ein LLM mit dem internen Namen X3 wird im Arbeitsbereich erzeugt.
- (8) Aus der Programmbibliothek LIB1 werden die LLMs A und Y in das aktuelle LLM eingefügt, da über den Operanden TYPE=L nur LLMs ausgewählt werden.
- (9) Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert. Als Elementname wird der interne Name X3 aus der letzten Anweisung START-LLM-CREATION übernommen.
- (10) Ein LLM mit dem internen Namen X4 wird im Arbeitsbereich erzeugt.
- (11) Aus der Programmbibliothek LIB1 wird das LLM A eingefügt, da über den Operanden TYPE=L nur LLMs ausgewählt werden.
- (12) Das Bindemodul B wird als Sub-LLM am Knoten mit dem Pfadnamen X4.A eingefügt, da über den Operand TYPE=R nur Bindemodule ausgewählt werden. Die Programmbibliothek LIB1 ist durch die vorhergehende Anweisung INCLUDE-MODULES festgelegt (aktuelle Programmbibliothek).
- (13) Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert. Als Elementname wird der interne Name X4 aus der letzten Anweisung START-LLM-CREATION übernommen.

Beispiel 2

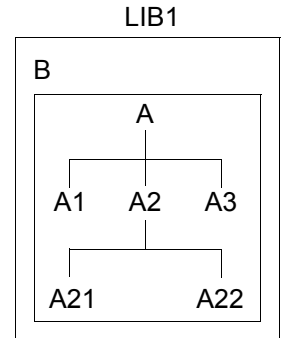
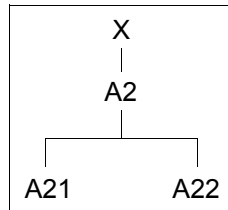
Einfügen eines Sub-LLM

Anweisungen

(1) START-LLM-CREATION
INTERNAL-NAME=X

aktuelles LLM
(Arbeitsbereich)Programmbibliothek

(2) INCLUDE-MODULES
LIBRARY=LIB1,
ELEMENT=B
(SUB-LLM=A.A2)



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Ein Sub-LLM A2 eines LLM A, das als Element B in der Programmbibliothek LIB1 gespeichert ist, wird in das aktuelle LLM eingefügt. Das Sub-LLM A2 wird im LLM A mit dem Pfadnamen A.A2 angesprochen.

3.2.2 Entfernen von Modulen

Module im aktuellen LLM entfernt der BINDER mit der Anweisung REMOVE-MODULES. Entfernt werden Bindemodule und Sub-LLMs. Die Sub-LLMs werden mit ihren Pfadnamen festgelegt.

Nicht ausgefügt werden:

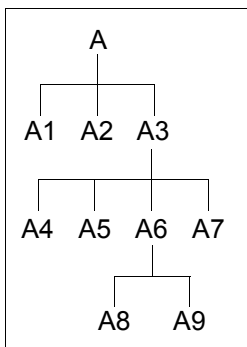
- das aktuelle Sub-LLM (siehe [Seite 57ff](#)),
- ein Sub-LLM, dessen Beginn zwar mit der Anweisung BEGIN-SUB-LLM-STATEMENTS definiert, dessen Ende jedoch noch nicht mit der Anweisung END-SUB-LLM-STATEMENTS festgelegt wurde, sowie jedes Sub-LLM, der ein solches Sub-LLM enthält (siehe [Seite 57ff](#)).

Beispiel

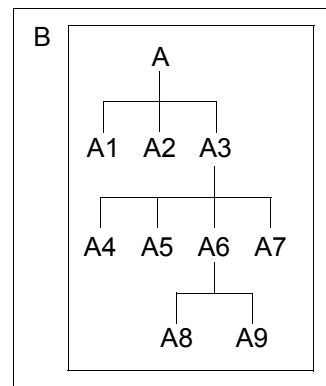
Entfernen eines Bindemoduls und eines Sub-LLM

Anweisungen

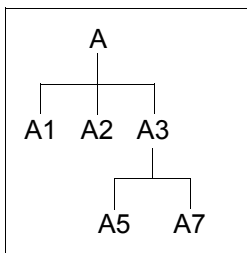
- (1) START-LLM-UPDATE
LIBRARY=LIB1,
ELEMENT=B

aktuelles LLM
(Arbeitsbereich)Programmbibliothek

LIB1



- (2) REMOVE-MODULES
NAME=(A4,A6),
PATH-NAME=A.A3



Bedeutung

- (1) Ein LLM, das als Element mit dem Namen B in der Programmbibliothek LIB1 gespeichert ist, soll geändert werden. Das LLM wird dazu in den Arbeitsbereich gelesen.
- (2) Das Bindemodul A4 und das Sub-LLM A6, die im Sub-LLM A3 des aktuellen LLM enthalten sind, werden entfernt. Der Pfadname PATH-NAME=A.A3 verzweigt zum Sub-LLM A3.

3.2.3 Ersetzen von Modulen

Die Anweisung REPLACE-MODULES ersetzt Module im aktuellen LLM. Die Module im aktuellen LLM können durch Bindemodule, LLMs oder durch beide ersetzt werden. Da das Ersetzen von Modulen ein implizites Einfügen anderer Module beinhaltet, gelten beim Ersetzen die gleichen Einschränkungen wie beim Einfügen von Modulen (siehe [Seite 47](#)).

Die Eingabequelle kann sein:

- für Bindemodule eine Programmbibliothek (Elementtyp R), eine Bindemodulbibliothek (OML) oder die EAM-Bindemoduldatei,
- für LLMs bzw. Sub-LLMs eine Programmbibliothek (Elementtyp L).

Die Eingabequelle kann entweder explizit angegeben werden oder es kann die aktuelle Eingabequelle übernommen werden. Die aktuelle Eingabequelle ist die Bibliothek bzw. EAM-Bindemoduldatei, aus der das letzte Modul geholt wurde (mit einer Anweisung START-LLM-UPDATE, INCLUDE-MODULES oder REPLACE-MODULES).

Aus der gewählten Eingabequelle können alle Module oder einzelne explizit angegebene Module geholt werden. Für die Module kann festgelegt werden, dass nur LLMs, nur OMs oder beide Typen gesucht werden. Werden LLMs *und* OMs in einer Programmbibliothek gesucht, kann bei Namensgleichheit die Priorität über den Operanden TYPE festgelegt werden. Standardmäßig hat ein LLM eine höhere Priorität als ein OM.

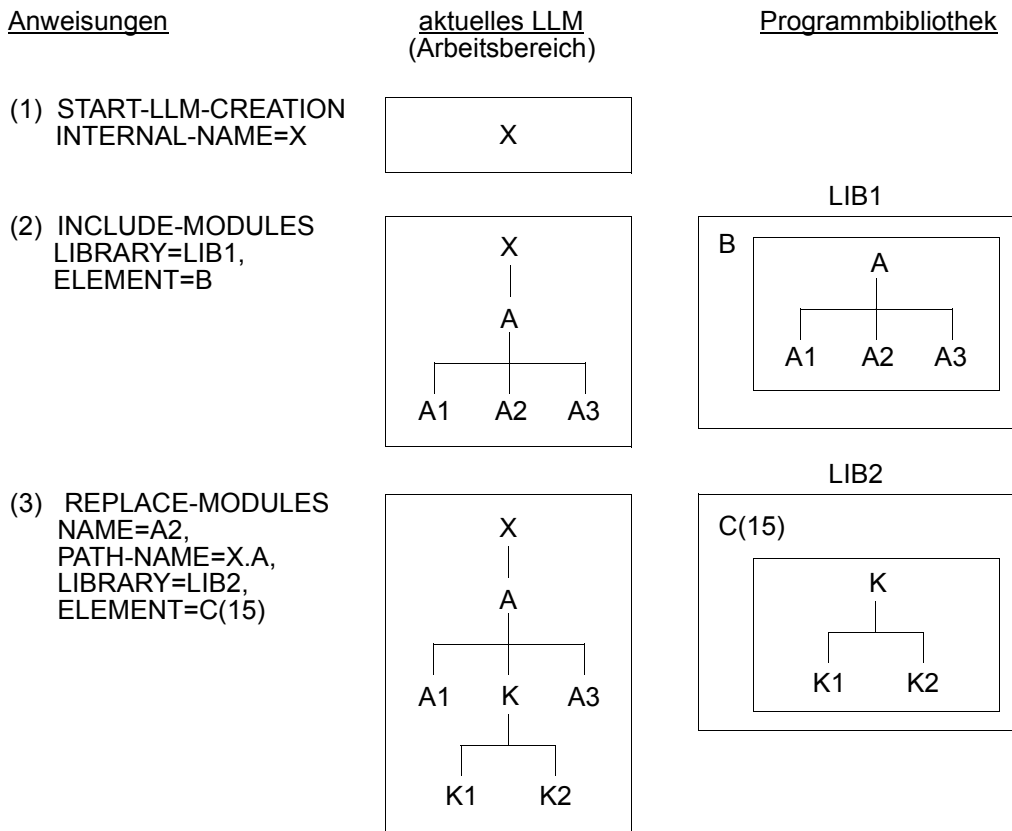
Der Name des neuen logischen Knotens (Operand NAME) kann entweder der interne Name des Moduls (LLM/OM), der externe Name des Moduls oder ein vom Benutzer vergeben Name sein. Module in einer Programmbibliothek werden nach ihrer Elementversion ausgewählt. Wenn keine Elementversion explizit angegeben wurde, wird das Element mit der *höchsten* Elementversion angenommen (siehe Handbuch „LMS“ [3]).

Mit dem Operanden RUN-TIME-VISIBILITY kann festgelegt werden, ob ein Modul als Laufzeitmodul betrachtet werden soll oder nicht. Bei Angabe von RUN-TIME-VISIBILITY=YES werden alle Symbole dieses Moduls beim Abspeichern des LLM maskiert, bereits befriedigte Externverweise bleiben jedoch befriedigt. Zur Befriedigung von Externverweisen beim Einfügen oder Ändern dieses LLM werden diese Symbole während des BINDER-Laufs wieder sichtbar gemacht.

Beim Ersetzen von Modulen hat der Benutzer auch die Möglichkeit, die Behandlung von Namenskonflikten zu steuern.

Beispiel 1

Ersetzen eines Bindemoduls durch ein LLM



Bedeutung

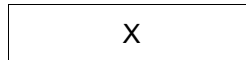
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Das LLM mit dem Elementnamen B wird aus der Programmbibliothek LIB1 eingefügt.
- (3) Das LLM mit dem Elementnamen C und der Elementversion 15 wird aus der Programmbibliothek LIB2 gelesen und ersetzt im aktuellen LLM als Sub-LLM das Bindemodul A2, zu dem über den Pfadnamen X.A verzweigt wird.

Beispiel 2

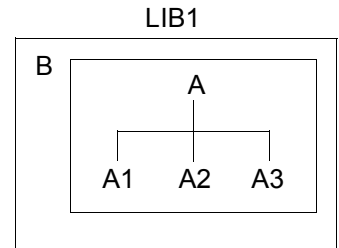
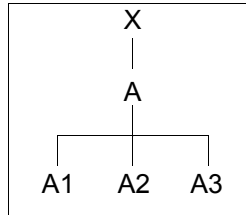
Ersetzen eines Bindemoduls durch ein LLM

Anweisungenaktuelles LLM
(Arbeitsbereich)Programmbibliothek

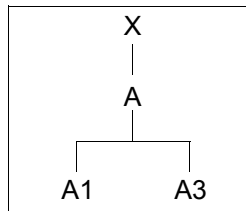
- (1) START-LLM-CREATION
INTERNAL-NAME=X



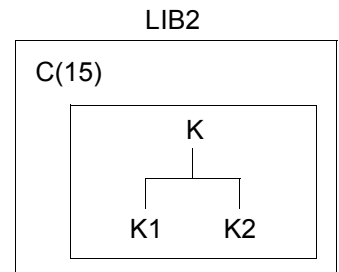
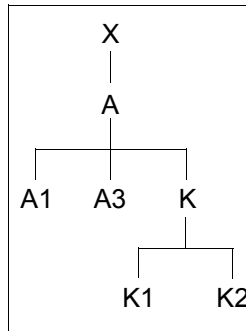
- (2) INCLUDE-MODULES
LIBRARY=LIB1,
ELEMENT=B



- (3) REMOVE-MODULES
NAME=A2,
PATH-NAME=X.A,



- (4) INCLUDE-MODULES
LIBRARY=LIB2,
ELEMENT=C(15),
PATH-NAME=X.A,

**Bedeutung**

(1)/(2) wie im Beispiel 1.

(3)/(4) Die aufeinander folgenden Anweisungen REMOVE-MODULES und INCLUDE-MODULES haben die gleiche Wirkung wie eine Anweisung REPLACE-MODULES (siehe Beispiel 1). Die Struktur des LLM ist jedoch verschieden von der Struktur in Beispiel 1.

3.3 Erzeugen und Ändern der logischen Struktur eines LLM

3.3.1 Erzeugen der logischen Struktur eines LLM

Die logische Struktur eines LLM wird durch geschachtelte Anweisungen BEGIN-SUB-LLM-STATEMENTS und END-LLM-STATEMENTS beschrieben. Die Anweisung BEGIN-SUB-LLM-STATEMENTS legt den Beginn eines Sub-LLM, die Anweisung END-SUB-LLM-STATEMENTS das Ende eines Sub-LLM in der entsprechenden Schachtelungsstufe fest. In jeder Schachtelungsstufe können z.B. Anweisungen INCLUDE-MODULES, REMOVE-MODULES und RESOLVE-BY-AUTOLINK angegeben werden (siehe [Bild 5](#)).

Der Knoten in der logischen Struktur, an dem das Sub-LLM beginnen soll, kann entweder durch den Pfadnamen festgelegt werden (siehe [Seite 24ff](#)) oder es kann das aktuelle Sub-LLM übernommen werden.

Das aktuelle Sub-LLM ist wie folgt festgelegt:

- Die Anweisungen START-LLM-CREATION und START-LLM-UPDATE legen die Root des LLM-Strukturbaumes als aktuelles Sub-LLM fest.
- Jede folgende Anweisung BEGIN-SUB-LLM-STATEMENTS setzt das aktuelle Sub-LLM um eine Stufe weiter.
- Jede Anweisung END-SUB-LLM-STATEMENTS setzt das aktuelle Sub-LLM auf die Stufe zurück, die vor der zugehörigen Anweisung BEGIN-SUB-LLM-STATEMENTS das aktuelle Sub-LLM enthalten hat.

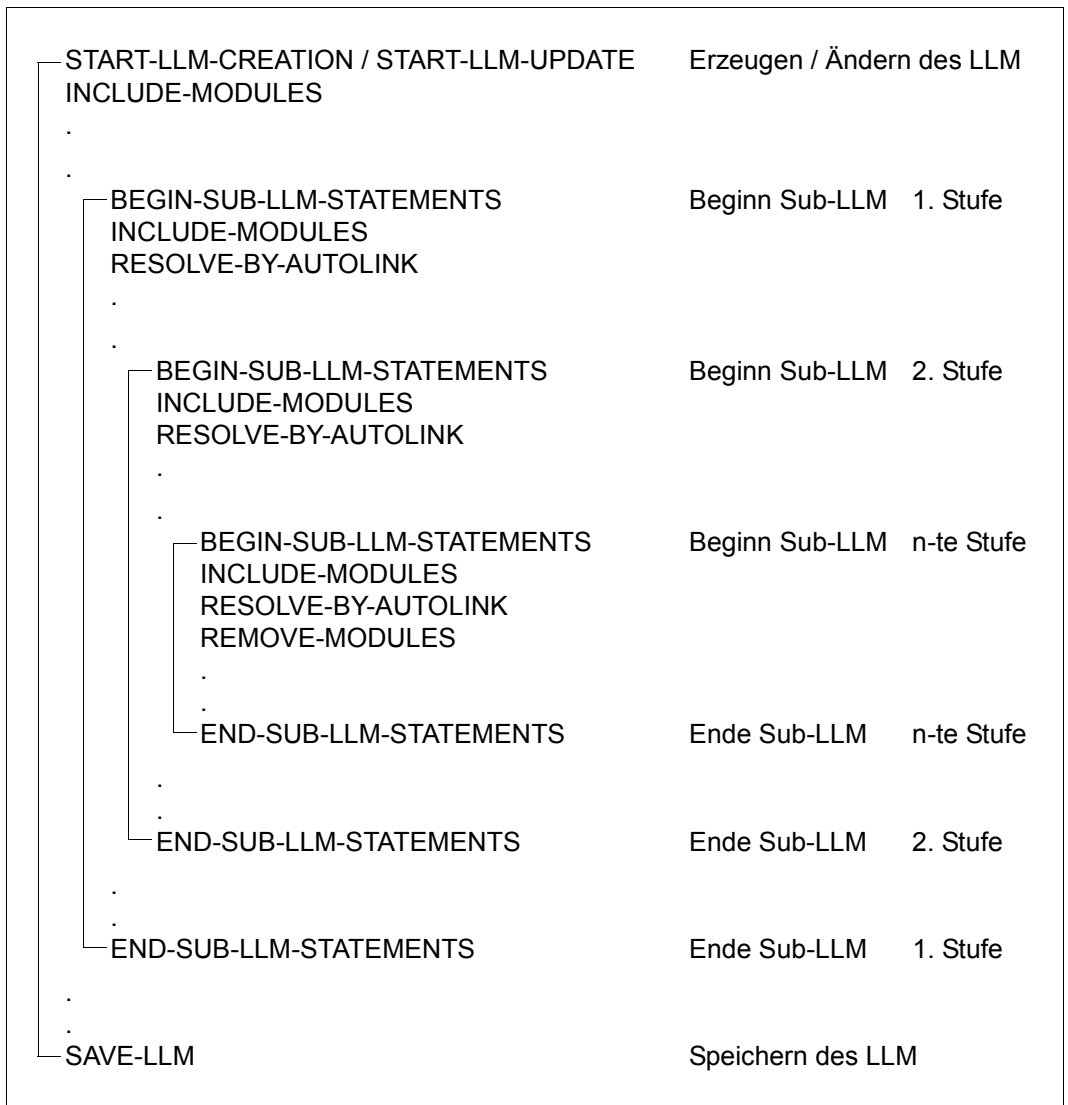


Bild 5: Schachtelung bei Sub-LLMs

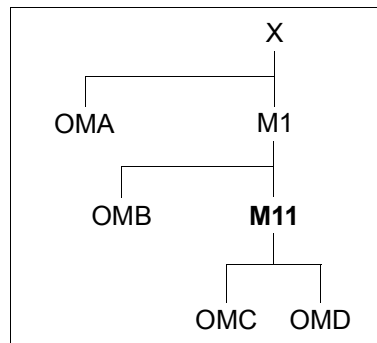
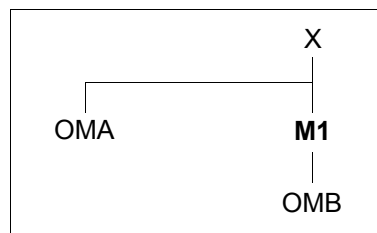
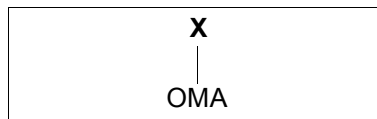
Die folgenden Beispiele erläutern die Wirkungsweise der Anweisungen BEGIN-SUB-LLM-STATEMENTS und END-SUB-LLM-STATEMENTS.

Beispiel 1

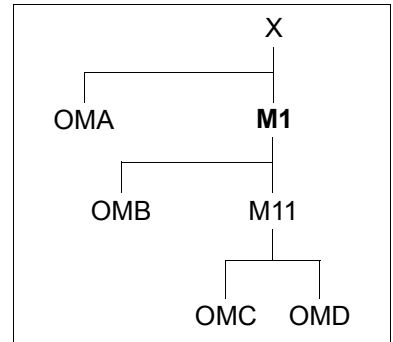
Dieses Beispiel zeigt die einfachste Anwendung der Anweisungen BEGIN-SUB-LLM-STATEMENTS und END-SUB-LLM-STATEMENTS. Hierbei werden zwei Sub-LLMs M1 und M2 *nacheinander* begonnen und beendet.

Anweisungen

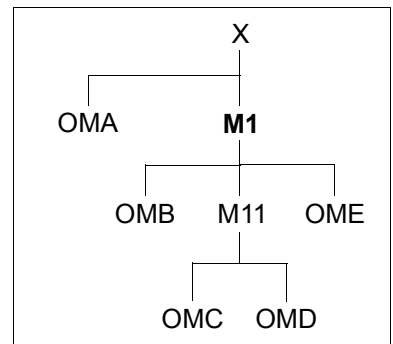
- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=OMA
- (3) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=M1
- (4) INCLUDE-MODULES ELEMENT=OMB
- (5) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=M11
- (6) INCLUDE-MODULES
ELEMENT=(OMC,OMD)

aktuelles LLM
(Arbeitsbereich)

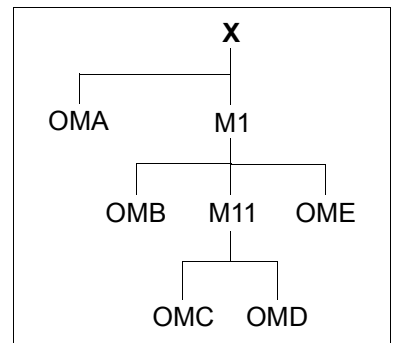
(7) END-SUB-LLM-STATEMENTS



(8) INCLUDE-MODULES ELEMENT=OME

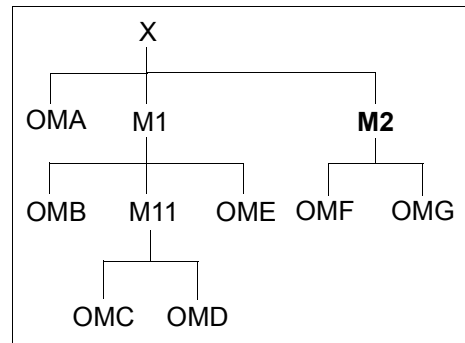


(9) END-SUB-LLM-STATEMENTS



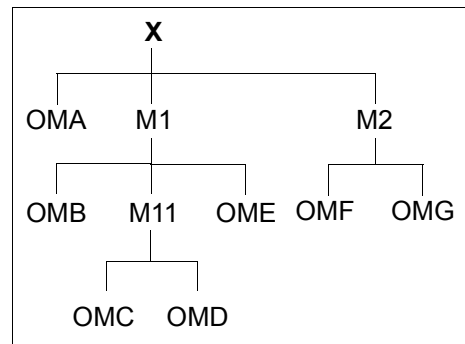
(10) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=M2

(11) INCLUDE-MODULES
ELEMENT=(OMF,OMG)



(12) END-SUB-LLM-STATEMENTS

(13) SAVE-LLM LIBRARY=LIB1



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt. Damit wird die Root X des LLM zum aktuellen LLM.
- (2) Aus der Programmbibliothek LIB1 wird das Bindemodul OMA eingefügt.
- (3) Ein Sub-LLM mit dem Namen M1 wird an der Root begonnen. M1 wird damit zum aktuellen Sub-LLM.
- (4) In das aktuelle Sub-LLM M1 wird aus der aktuellen Programmbibliothek LIB1 das Bindemodul OMB eingefügt.
- (5) Ein Sub-LLM mit dem Namen M11 wird am aktuellen Sub-LLM M1 begonnen. M11 wird damit zum aktuellen Sub-LLM.
- (6) In das aktuelle Sub-LLM M11 werden aus der aktuellen Programmbibliothek LIB1 die Bindemodule OMC und OMD eingefügt.
- (7) Das aktuelle Sub-LLM M11 wird abgeschlossen. Damit wird M1 zum aktuellen Sub-LLM.

- (8) In das aktuelle Sub-LLM M1 wird das Bindemodul OME eingefügt.
- (9) Das aktuelle Sub-LLM wird abgeschlossen. Damit wird die Root zum aktuellen Sub-LLM.
- (10) Ein Sub-LLM mit dem Namen M2 wird an der Root begonnen. M2 wird damit zum aktuellen Sub-LLM.
- (11) In das aktuelle Sub-LLM M2 werden die Bindemodule OMF und OMG eingefügt.
- (12) Das aktuelle Sub-LLM M2 wird abgeschlossen. Damit wird die Root X zum aktuellen LLM.
- (13) Das erzeugte LLM wird in der Programmbibliothek LIB1 gespeichert. Als Elementname wird der interne Name X aus der letzten Anweisung START-LLM-CREATION übernommen.

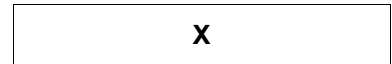
Beispiel 2

Dieses Beispiel zeigt die Anwendung der Anweisungen BEGIN-SUB-LLM-STATEMENTS und END-SUB-LLM-STATEMENTS mit Pfadnamen. Hierbei werden zwei Sub-LLMs M1 und M2 erzeugt. Nachdem M2 erzeugt ist, wird M1 durch ein Sub-LLM M11 ergänzt.

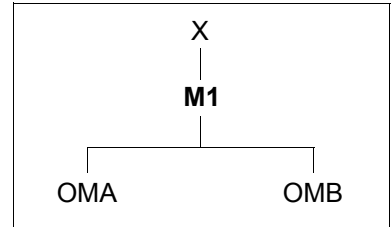
Anweisungen

aktuelles LLM
(Arbeitsbereich)

- (1) START-LLM-CREATION
INTERNAL-NAME=X

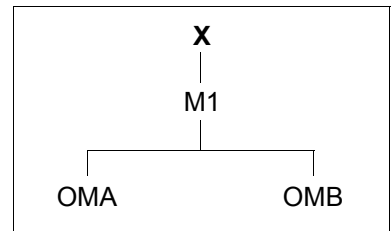


- (2) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=M1



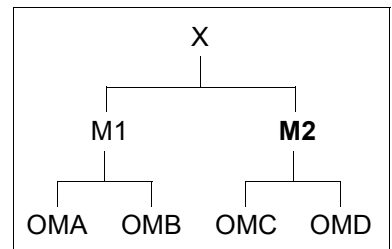
- (3) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(OMA,OMB)

- (4) END-SUB-LLM-STATEMENTS



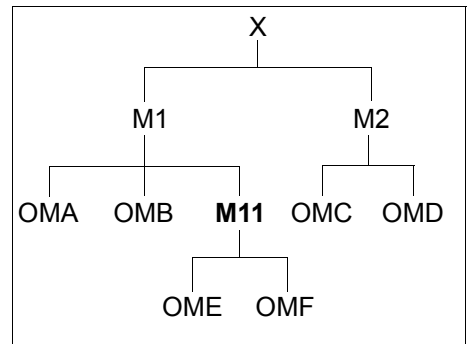
- (5) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=M2

- (6) INCLUDE-MODULES
ELEMENT=(OMC,OMD)

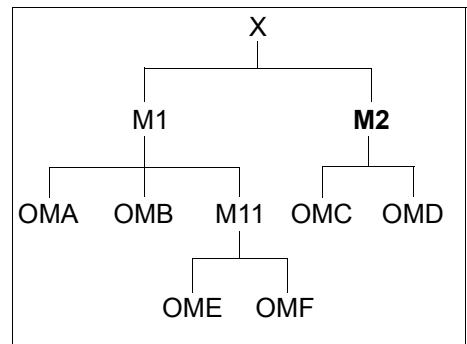


(7) BEGIN-SUB-LLM-STATEMENTS
 SUB-LLM-NAME=M11
 PATH-NAME=X.M11

(8) INCLUDE-MODULES
 ELEMENT=(OME,OMF)

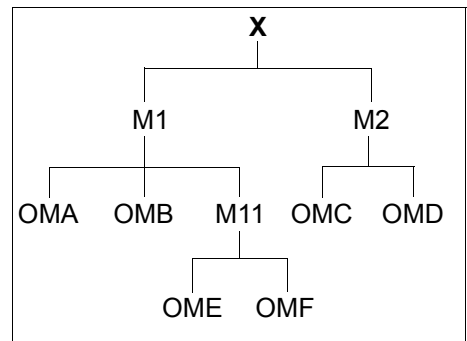


(9) END-SUB-LLM-STATEMENTS



(10) END-SUB-LLM-STATEMENTS

(11) SAVE-LLM LIBRARY=LIB1



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt. Damit wird die Root X des LLM zum aktuellen LLM.
- (2) Ein Sub-LLM mit dem Namen M1 wird begonnen. M1 wird damit zum aktuellen Sub-LLM.
- (3) In das aktuelle Sub-LLM M1 werden aus der Programmbibliothek LIB1 die Bindemodule OMA und OMB eingefügt.
- (4) M1 wird abgeschlossen. Damit wird die Root X des LLM wieder zum aktuellen LLM.
- (5) Ein Sub-LLM mit dem Namen M2 wird begonnen. M2 wird zum aktuellen Sub-LLM.
- (6) In das aktuelle Sub-LLM M2 werden aus der aktuellen Programmbibliothek LIB1 die Bindemodule OMC und OMD eingefügt.
- (7) Ein Sub-LLM M11 wird begonnen. Durch den Pfadnamen X.M1 wird festgelegt, dass M11 im Sub-LLM M1 beginnen soll. Das Sub-LLM M11 wird damit zum aktuellen Sub-LLM.
- (8) In das aktuelle Sub-LLM M11 werden aus der aktuellen Programmbibliothek LIB1 die Bindemodule OME und OMF eingefügt.
- (9) Das Sub-LLM M11 wird abgeschlossen. Das Sub-LLM M2, das vor Beginn des Sub-LLM M11 das aktuelle Sub-LLM war, wird damit wieder zum aktuellen Sub-LLM.
- (10) Das aktuelle Sub-LLM M2 wird abgeschlossen. Damit wird die Root X wieder zum aktuellen Sub-LLM.
- (11) Das erzeugte LLM wird in der Programmbibliothek LIB1 gespeichert.

3.3.2 Ändern der logischen Struktur eines LLM

Mit der Anweisung `MODIFY-MODULE-ATTRIBUTES` kann die logische Struktur eines LLM geändert werden. Dazu wird der Pfadname eines eingebundenen Moduls geändert und das Modul wird damit an anderer Stelle in das LLM eingebaut.

Die Anweisung `MODIFY-MODULE-ATTRIBUTES` kann neben dem Pfadnamen noch folgende Merkmale verändern:

- den logischen Namen des (Sub-)LLM (`NEW-NAME`),
- Vereinbarungen über Test- und Diagnoseinformationen (LSD) (`TEST-SUPPORT`),
- Maskierung aller Symbole eines Moduls (`RUN-TIME-VISIBILITY`),
- Behandlung von Namenskonflikten (`NAME-COLLISION`), wenn der Wert des Operanden `RUN-TIME-VISIBILITY` geändert wurde.

Das folgende Beispiel erläutert die Wirkungsweise der Anweisung `MODIFY-MODULE-ATTRIBUTES` beim Ändern der logischen Struktur des LLM.

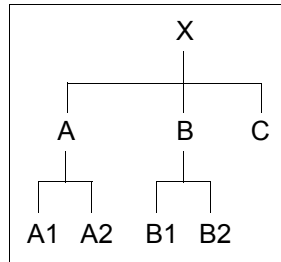
Beispiel

Ändern des Pfadnamens eines Sub-LLM

Anweisungen

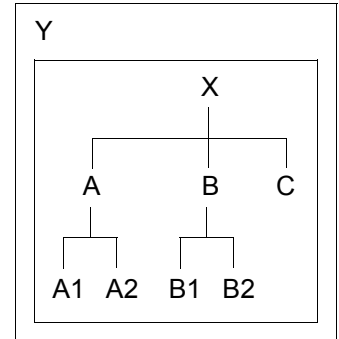
(1) `START-LLM-UPDATE`
`LIBRARY=LIB1,`
`ELEMENT=Y`

aktuelles LLM
 (Arbeitsbereich)



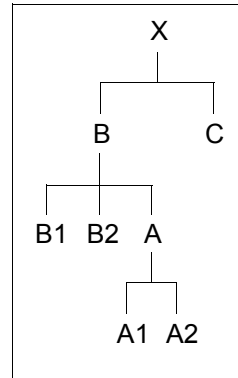
Programmbibliothek

LIB1

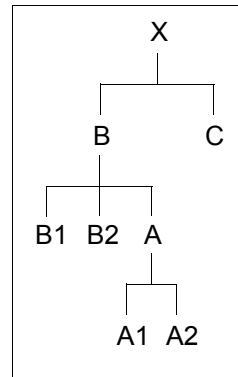


Anweisungen

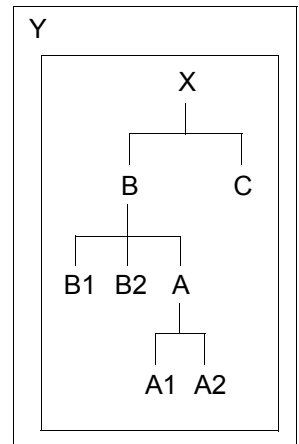
- (2) MODIFY-MODULE-ATTRIBUTES
 NAME=A,
 PATH-NAME=X,
 NEW-PATH-NAME=X.B

aktuelles LLM
(Arbeitsbereich)Programmbibliothek

- (3) SAVE-LLM
 LIBRARY=LIB1,
 ELEMENT=Y



LIB1



Bedeutung

- (1) Ein LLM, das als Element mit dem Namen Y in der Programmbibliothek LIB1 gespeichert ist, soll geändert werden. Das LLM wird dazu in den Arbeitsbereich geladen.
- (2) Der Pfadname des Sub-LLMs A wird von X nach X.B geändert und das Sub-LLM A somit an anderer Stelle in das LLM eingebaut.
- (3) Das geänderte LLM wird wieder unter dem Elementnamen Y in der Programmbibliothek LIB1 abgespeichert.

3.4 Erzeugen der physischen Struktur eines LLM

Die physische Struktur eines LLM ist durch Slices realisiert (siehe [Seite 18ff](#)). Im Folgenden wird das Erzeugen von physischen Strukturen eines LLM beschrieben, die aus

- benutzerdefinierten Slices (User defined Slices) oder
 - nach Attributen von CSECTs gebildeten Slices (Slices by Attributes)
- bestehen.

3.4.1 Benutzerdefinierte Slices

Beim Erzeugen der physischen Struktur eines LLM kann der Benutzer Slices definieren, die getrennt geladen werden. Zunächst lädt der Lader nur die **Root-Slice** in den Hauptspeicher. Die Root-Slice bleibt während des gesamten Ablaufs geladen. Die anderen Slices kann der Benutzer nachladen lassen, sobald sie für den Ablauf erforderlich sind.

Slices können sich gegenseitig überlagern, d.h. sie belegen nacheinander den gleichen Adressraum. Sie haben z.B. die gleiche Anfangsadresse und können geladen, überlagert und wieder geladen werden, so oft dies erforderlich ist. Der Benutzer kann anfordern, dass ein Overlay Control Module (OCM) in der Root-Slice des Benutzerprogrammes erzeugt wird. Dieses OCM steuert die erforderlichen LDSLICE-Makroaufrufe (siehe Handbuch „BLS-SERV Bindelader-Starter“ [1]), mit denen die Überlagerungsstruktur realisiert wird. Treten bei LDSLICE Fehler auf (z.B. wenn eine leere Slice geladen werden soll), wird das Benutzerprogramm mit dem Status ‚ABNORMAL‘ beendet.

Diagrammdarstellung der physischen Struktur

Eine Hilfe für den Benutzer ist die Darstellung der gewünschten Struktur als Diagramm, das die Beziehungen zwischen den einzelnen Slices grafisch darstellt (siehe [Bild 6](#)).

Jede vertikale Linie in diesem Diagramm stellt eine Slice dar. Die oberste Slice in der Struktur ist die **Root-Slice (%ROOT)**. Sie wird zu Beginn geladen, während die übrigen Slices erst bei Bedarf geladen werden.

Jede horizontale Linie ist ein Adresspegel. Alle Slices, die an demselben Adresspegel anfangen, haben dieselbe Ladeadresse, die sich an den vorausgehenden Slice anschließt. Sie können sich also überlagern. Man bezeichnet sie daher auch als **exklusive Slices**. Die Root-Slice wird nie überlagert.

Alle Slices, durch die sich eine verzweigungsfreie Linie ziehen lässt, liegen in einem gemeinsamen Ast. Alle Slices zwischen Root-Slice und einer Slice X bezeichnet man als „höher als X“ in dem betreffenden Ast des Diagramms. Alle Slices unterhalb einer Slice X nennt man „niedriger als X“, bezogen auf diese Slice. Die Root-Slice ist also höher als alle übrigen Slices. Alle exklusiven Slices sind niedriger als die Root-Slices.

Jeder Slice ist eine **Stufennummer** zugeordnet, die durch die Anzahl der höheren Slices in dem betreffenden Ast bestimmt ist. Die Root-Slice hat die Stufe Null.

Eine neue **Region** beginnt bei einem Adresspegel, der so liegt, dass eine Überlagerung durch vorangehende Slices ausgeschlossen ist.

Der Benutzer kann sich zu jedem Zeitpunkt mit der Anweisung SHOW-MAP eine Übersicht über die physische Struktur des aktuellen LLM ausgeben lassen.

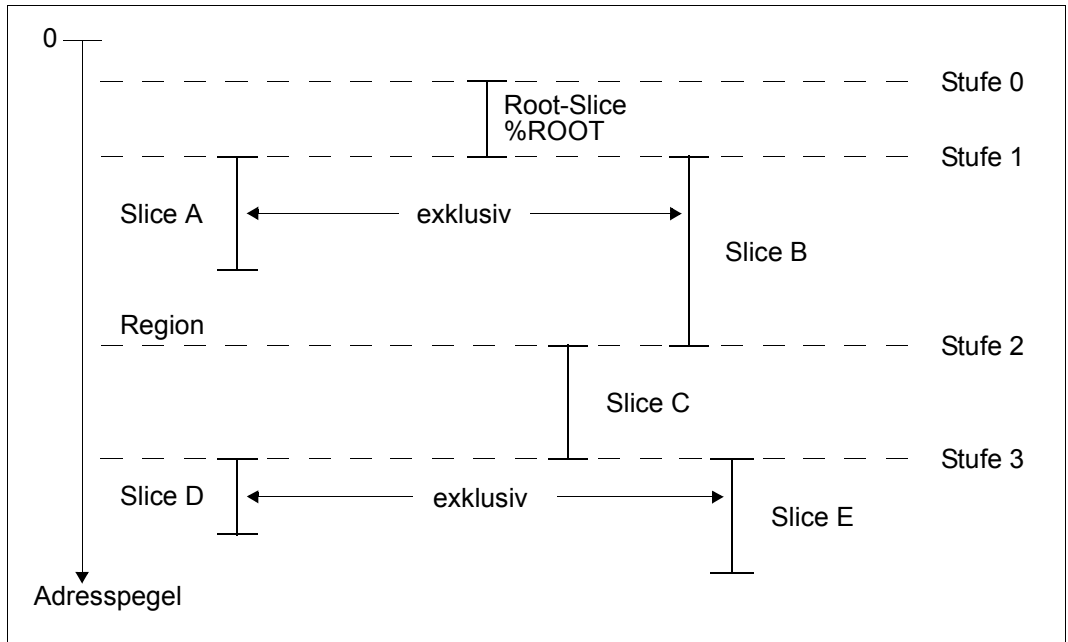


Bild 6: Beispiel für die Diagrammdarstellung der physischen Struktur

Festlegen der physischen Struktur

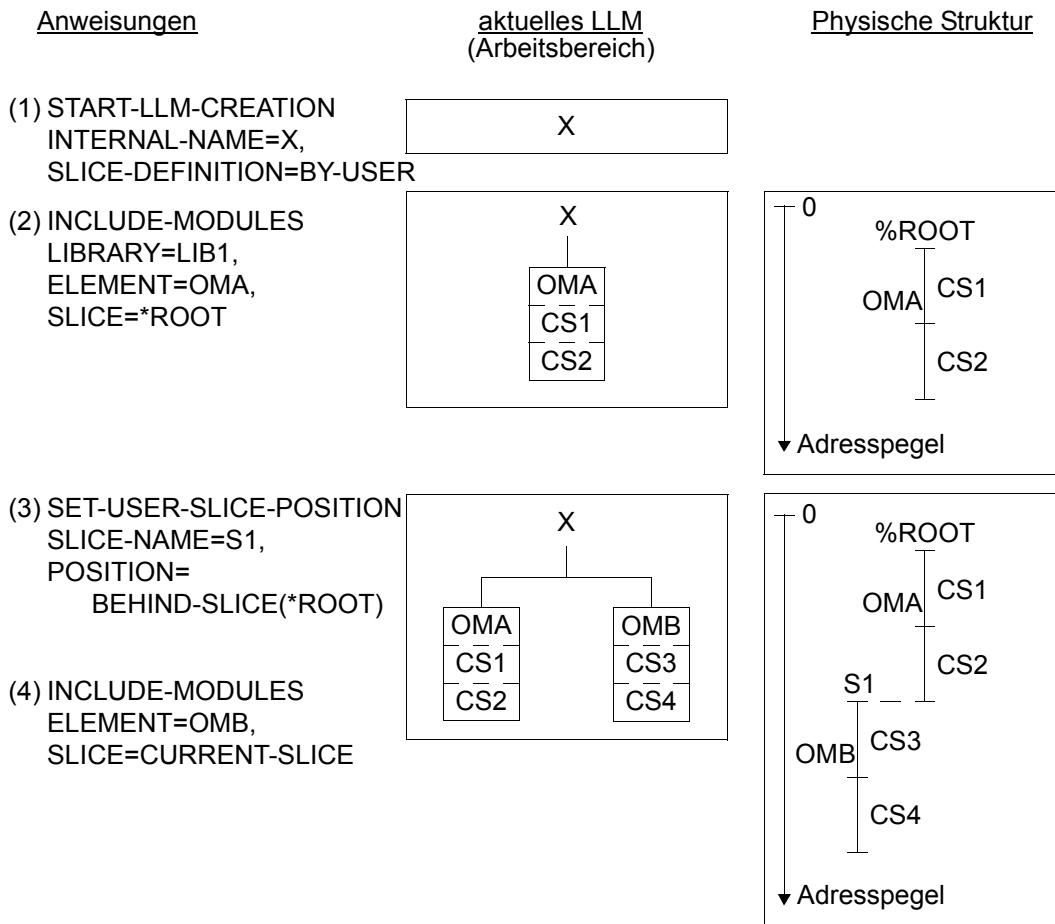
Die physische Struktur legt der Benutzer mit Anweisungen SET-USER-SLICE-POSITION fest. Die in einer Anweisung SET-USER-SLICE-POSITION angegebene Slice wird zur **aktuellen Slice**. Dies ist die Slice, in die Module eingefügt oder in der Module ersetzt werden, wenn in den Anweisungen INCLUDE-MODULES oder REPLACE-MODULES keine Angaben für die Slice gemacht wurden (Operand SLICE).

Wurde noch keine Slice mit einer Anweisung SET-USER-SLICE-POSITION festgelegt, wird die Root-Slice (%ROOT) als aktuelle Slice angenommen. Mit Autolink ausgewählte Module werden nur in die Root-Slice eingebunden.

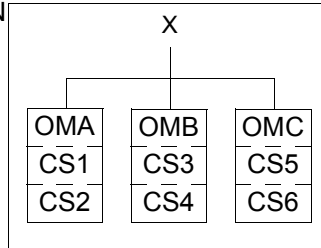
Soll die physische Struktur vom Benutzer mit Anweisungen SET-USER-SLICE-POSITION definiert werden, muss dies beim Erzeugen des LLM in der Anweisung START-LLM-CREATION vereinbart werden (Operand SLICE-DEFINITION=BY-USER).

Beispiel 1

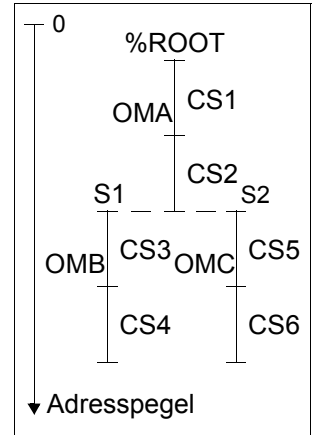
Erzeugen einer physischen Struktur, die aus der Root-Slice und zwei exklusiven Slices besteht.



- (5) SET-USER-SLICE-POSITION
SLICE-NAME=S2,
POSITION=
BEHIND-SLICE(*ROOT)
- (6) INCLUDE-MODULES
ELEMENT=OMC,
SLICE=S2



- (7) SAVE-LLM LIBRARY=LIB1

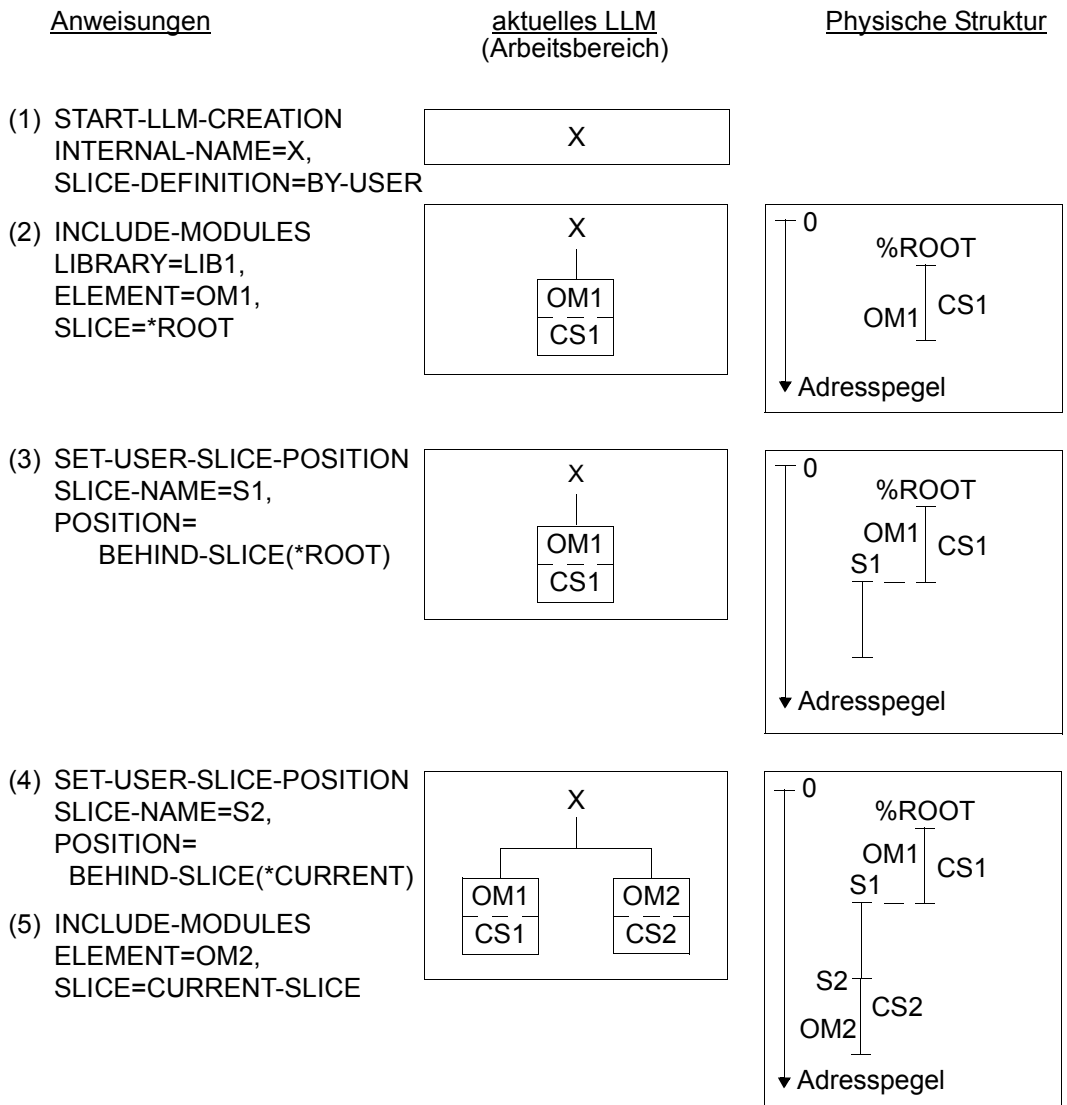


Bedeutung

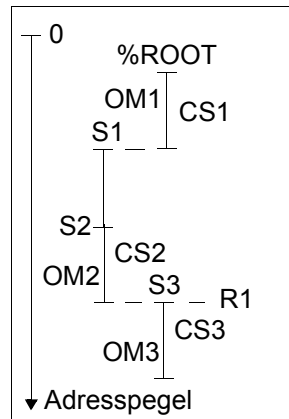
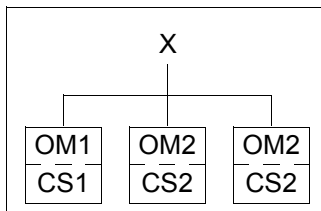
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt. Der Operand SLICE-DEFINITION vereinbart, dass die physische Struktur des LLM vom Benutzer mit Anweisungen SET-USER-SLICE-POSITION festgelegt wird. Die Root-Slice %ROOT wird festgelegt.
- (2) In das aktuelle LLM im Arbeitsbereich wird das Bindemodul OMA aus der Programmbibliothek LIB1 eingefügt. OMA enthält die beiden CSECTs CS1 und CS2. Der Operand SLICE=*ROOT legt fest, dass CS1 und CS2 die Root-Slice %ROOT bilden.
- (3) Die Slice S1 wird festgelegt. Sie schließt sich an die Root-Slice %ROOT an. S1 wird die aktuelle Slice.
- (4) In das aktuelle LLM wird das Bindemodul OMB aus der Programmbibliothek LIB1 eingefügt. OMB enthält die beiden CSECTs CS3 und CS4. Die CSECTs CS3 und CS4 bilden die Slice S1 (aktuelle Slice).
- (5) Die Slice S2 wird festgelegt. Sie schließt sich an die Root-Slice %ROOT an. S2 wird zur aktuellen Slice. Die Slice S2 und die Slice S1 sind exklusiv, d.h. S2 kann S1 überlagern.
- (6) In das aktuelle LLM wird das Bindemodul OMC aus der Programmbibliothek LIB1 eingefügt. OMC enthält die beiden CSECTs CS5 und CS6. Der Operand SLICE=S2 legt fest, dass CS5 und CS6 die Slice S2 bilden.
- (7) Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert.

Beispiel 2

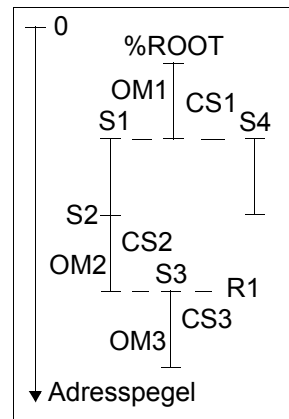
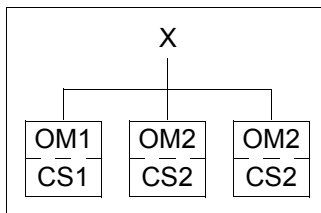
Erzeugen einer physischen Struktur, die aus der Root-Slice, zwei exklusiven Slices und einer neuen Region besteht.



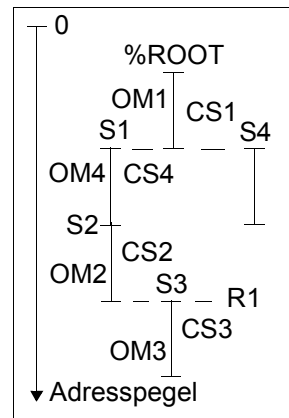
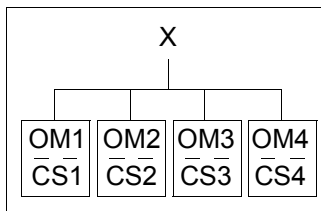
- (6) SET-USER-SLICE-POSITION
SLICE-NAME=S3,
POSITION=BEGIN-REGION
(REGION=R1,
- (7) INCLUDE-MODULES
ELEMENT=OM3,
SLICE=S3



- (8) SET-USER-SLICE-POSITION
SLICE-NAME=S4,
POSITION=
BEHIND-SLICE(*ROOT)



- (9) SET-USER-SLICE-POSITION
SLICE-NAME=S1,
- (10) INCLUDE-MODULES
ELEMENT=OM4,
SLICE=S1



- (11) SAVE-LLM LIBRARY=LIB1

Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt. Der Operand SLICE-DEFINITION vereinbart, dass die physische Struktur des LLM vom Benutzer mit Anweisungen SET-USER-SLICE-POSITION festgelegt wird. Die Root-Slice %ROOT wird festgelegt.
- (2) In das aktuelle LLM im Arbeitsbereich wird das Bindemodul OM1 aus der Programmbibliothek LIB1 eingefügt. OM1 enthält die CSECT CS1. Der Operand SLICE=*ROOT legt fest, dass CS1 die Root-Slice %ROOT bildet.
- (3) Die Slice S1 wird festgelegt. Sie schließt sich an die Root-Slice an. S1 wird zur aktuellen Slice.
- (4) Die Slice S2 wird festgelegt. Sie schließt sich an die aktuelle Slice S1 an (Operand *CURRENT). S2 wird zur aktuellen Slice.
- (5) In das aktuelle LLM im Arbeitsbereich wird das Bindemodul OM2 aus der Programmbibliothek LIB1 eingefügt. OM2 enthält die CSECT CS2. Die CSECT CS2 bildet die Slice S2 (aktuelle Slice).
- (6) Die Slice S3 wird festgelegt. Sie beginnt an einer neuen Region R1. Ihre Anfangsadresse liegt am Ende von S1, da die Slices der Region R1 die höheren Slices nicht überlagern dürfen. S3 wird zur aktuellen Slice.
- (7) In das aktuelle LLM im Arbeitsbereich wird das Bindemodul OM3 aus der Programmbibliothek LIB1 eingefügt. OM3 enthält die CSECT CS3. Der Operand SLICE=S3 legt fest, dass CS3 die Slice S3 bildet.
- (8) Die Slice S4 wird festgelegt. Sie schließt sich an die Root-Slice %ROOT an. S4 wird zur aktuellen Slice. Die Slice S4 ist exklusiv zu den Slices S1 und S2, d.h. sie kann S1 und S2 überlagern.
- (9) Die bereits vorhandene Slice S1 soll geändert werden (Operand MODE=UPDATE).
- (10) In das aktuelle LLM im Arbeitsbereich wird das Bindemodul OM4 aus der Programmbibliothek LIB1 eingefügt. OM4 enthält die CSECT CS4. Der Operand SLICE=S1 legt fest, dass CS4 die Slice S1 bildet.
- (11) Das aktuelle LLM wird in der Programmbibliothek LIB1 gespeichert.

3.4.2 Nach Attributen gebildete Slices

Verfahren zur Bildung der Slices

Bei einem LLM, dessen Slices nach Attributen von CSECTs gebildet werden sollen, bildet der BINDER automatisch Slices nach folgenden Attributen:

READ-ONLY

- Lesezugriff (READ-ONLY=YES)
Die CSECT kann nur gelesen werden. Dieses Attribut schützt die CSECT im Hauptspeicher vor dem Überschreiben.
- Lese- und Schreibzugriff (READ/WRITE) (READ-ONLY=NO)
Die CSECT kann gelesen und überschrieben werden.

RESIDENT

- Hauptspeicherresident (RESIDENT=YES)
Die CSECT wird in den Klasse-3-Speicher geladen und dort resident gehalten.
- Seitenwechselbar (PAGEABLE) (RESIDENT=NO)
Die CSECT ist seitenwechselbar.

Dieses Attribut ist *nur* für Shared Code des Systems relevant.

PUBLIC

- Gemeinsam benutzbar (PUBLIC=YES)
Die CSECT enthält Daten und Instruktionen, die gemeinsam benutzt werden können (siehe [Seite 76](#)).
- Nicht gemeinsam benutzbar (PRIVATE, d.h. PUBLIC=NO)
Die CSECT enthält Daten und Instruktionen, die nur privat benutzt werden können.

RMODE

- Residenzmodus (RMODE=ANY)
Die CSECT kann unterhalb 16 Mbyte und oberhalb 16 Mbyte geladen werden.
- Residenzmodus (RMODE=24)
Die CSECT kann nur unterhalb 16 Mbyte geladen werden.

Durch Kombination aller Attribute können bis zu 16 verschiedene Slices gebildet werden. Die Attribute nach denen Slices gebildet werden, legt der Benutzer in der Anweisung START-LLM-CREATION (Operand BY-ATTRIBUTES) fest.

Gibt der Benutzer z.B. an, dass Slices nach dem Attribut READ-ONLY gebildet werden sollen, generiert der BINDER folgende zwei Slices:

1. Eine Slice mit allen CSECTs READ-ONLY
2. Eine Slice mit allen CSECTs READ/WRITE

Alle übrigen Attribute werden nicht zur Bildung von Slices benutzt, da im Operanden BY-ATTRIBUTES die Standardwerte NO gelten.

Wenn keine CSECT mit dem Attribut READ-ONLY vorhanden ist, wird keine Slice generiert. Der BINDER generiert also keine leere Slice. Alle Slices werden als gesamte Einheit in den Hauptspeicher geladen. Im obigen Beispiel werden die READ-ONLY-Slices und die READ/WRITE-Slices in den Klasse-6-Speicher geladen.

Slices mit dem Attribut PUBLIC

Slices mit dem Attribut PUBLIC=YES müssen ablaufinvariant programmiert sein, da sie nur unter dieser Voraussetzung als Shared Code (siehe Handbuch „BLSSERV Bindelader-Starter“ [1]) verwendet werden können. Alle PUBLIC-Slices eines LLM können nur gemeinsam als Shared Code geladen werden und bilden den PUBLIC-Teil dieses LLM. Es gibt drei Möglichkeiten, den PUBLIC-Teil eines LLM als Shared Code zu laden:

1. Der Benutzer kann die PUBLIC-Slices mit dem DBL-Makro ASHARE in einen Common Memory Pool im Klasse-6-Speicher laden. Das Entladen der PUBLIC-Slices ist mit dem DBL-Makro DSHARE möglich.
2. Mit DSSM können die PUBLIC-Slices als unprivilegiertes Subsystem (siehe Handbuch „Einführung in die Systembetreuung“ [9]) in den Klasse-3- oder Klasse-4-Speicher geladen werden. Das Entladen ist nur über das DSSM möglich.

Wird ein solches LLM mit den Kommandos LOAD-EXECUTABLE-PROGRAM oder START-EXECUTABLE-PROGRAM (bzw. LOAD-PROGRAM/START-PROGRAM) bzw. mit dem DBL-Makro BIND aufgerufen, so werden die PRIVATE-Slices in den tasklokalen Klasse-6-Speicher geladen. Zur Befriedigung der Externverweise in den PRIVATE-Slices werden im gesamten Shared Code die zum LLM gehörenden PUBLIC-Slices gesucht. Diese Suche kann im Ladeaufruf (START-EXECUTABLE-PROGRAM, LOAD-EXECUTABLE-PROGRAM, START-PROGRAM, LOAD-PROGRAM, BIND-Makro) mit dem Operanden SHARE[-SCOPE] verboten werden (siehe Handbuch „BLSSERV Bindelader-Starter“ [1]). Externverweise in den PRIVATE-Slices werden von den PUBLIC-Slices befriedigt, Externverweise der PUBLIC-Slices werden *nicht* von den PRIVATE-Slices befriedigt. Wurde der PUBLIC-Teil dieses LLMs nicht im Shared Code gefunden, so wird das Laden des PUBLIC-Teils in den tasklokalen Klasse-6-Speicher veranlasst.

Slice-Name

Die Slice-Namen werden vom BINDER gebildet. Sie werden in den Listen des BINDER protokolliert (siehe [Seite 141ff](#)).

Der Slice-Name besteht aus 4 Teilnamen (für jedes Attribut ein Teilname) und hat folgenden Aufbau:

PUa-ROb-RTc-RMd

Es bedeuten:

PUa	Teilname für das Attribut PUBLIC
a	Kennzeichen für die Art der Slice
a=U	Attribut nicht zur Bildung der Slice benutzt (UNDEFINED)
a=Y	Slice ist gemeinsam benutzbar (PUBLIC=YES)
a=N	Slice ist nicht gemeinsam benutzbar (PUBLIC=NO)
ROb	Teilname für das Attribut READ-ONLY
b	Kennzeichen für die Art der Slice
b=U	Attribut nicht zur Bildung der Slice benutzt (UNDEFINED)
b=Y	Slice mit Lesezugriff (READ-ONLY=YES)
b=N	Slice mit Lese- und Schreibzugriff (READ-ONLY=NO)
RTc	Teilname für das Attribut RESIDENT
c	Kennzeichen für die Art der Slice
c=U	Attribut nicht zur Bildung der Slice benutzt (UNDEFINED)
c=Y	Slice ist hauptspeicherresident (RESIDENT=YES)
c=N	Slice ist nicht hauptspeicherresident (RESIDENT=NO)
RMd	Teilname für das Attribut RESIDENCY-MODE
d	Kennzeichen für die Art der Slice
d=U	Attribut nicht zur Bildung der Slice benutzt (UNDEFINED)
d=A	RMODE=ANY
d=4	RMODE=24

Beispiel

Soll nur das Attribut READ-ONLY zur Bildung von Slices verwendet werden, werden zwei Slices mit folgenden Slice-Namen gebildet:

1. PUU-ROY-RTU-RMU (Slice mit Lesezugriff)
2. PUU-RON-RTU-RMU (Slice mit Lese- und Schreibzugriff)

3.4.3 Ändern des Typs der physischen Struktur

Der Typ der physischen Struktur kann nachträglich mit der Anweisung MODIFY-LLM-ATTRIBUTES (Operand SLICE-DEFINITION) geändert werden. Folgende Änderungen des Typs der physischen Struktur sind erlaubt:

1. LLM mit nach Attributen gebildeten Slices in ein LLM mit Einzelslice
2. LLM mit Einzelslice in ein LLM mit nach Attributen gebildeten Slices
3. LLM mit nach Attributen gebildeten Slices in ein LLM mit nach *anderen* Attributen gebildeten Slices
4. LLM mit benutzerdefinierten Slices in ein LLM mit benutzerdefinierten Slices und geänderten Werten für AUTOMATIC-CONTROL und EXCLUSIVE-SLICE-CALL.

3.4.4 Verknüpfung zwischen PRIVATE- und PUBLIC-Slices

Die Verknüpfung zwischen PRIVATE- und PUBLIC-Slices kann mit dem Operanden CONNECTION-MODE gesteuert werden. Dieser Operand ist dem Operanden FORBS2000-VERSIONS in den Anweisungen MODIFY-STD-DEFAULTS und SAVE-LLM untergeordnet.

Der Operand CONNECTION-MODE wird nur berücksichtigt, wenn die Aufteilung des LLMs in Slices gemäß dem PUBLIC-Attribut der CSECTs erfolgte.

Die Verknüpfung zwischen PRIVATE- und PUBLIC-Slices kann auf zwei Weisen erfolgen:

1. Durch Auflösen (CONNECTION-MODE=*BY-RESOLUTION):

Beim Laden stellt DBL Verknüpfungen zwischen dem PRIVATE-Teil und allen Symbolen (Name und Typ) im PUBLIC-Teil her, auf die sich Relativierungsinformation aus dem PRIVATE-Teil bezieht.

2. Durch Relativieren (CONNECTION-MODE=*BY-RELOCATION):

Beim Laden stellt DBL eine einzige Verknüpfung zwischen dem PRIVATE- und dem PUBLIC-Teil her. Falls ein Subsystem-ENTRY angegeben ist, bezieht sich die Verknüpfung auf diesen.

Einschränkung

Bei LLM-Format 1 wird nur dann eine Verknüpfung zwischen dem PRIVATE- und dem PUBLIC-Teil hergestellt, wenn die PUBLIC-Slice mindestens eine Definition enthält, auf die sich ein Externverweis im PRIVATE-Teil bezieht.

Mit CONNECTION-MODE=*BY-RELOCATION ist die Performance von DBL beim Aufbau der Verknüpfung besser als mit CONNECTION-MODE=*BY-RESOLUTION. Falls jedoch der Mechanismus des Indirekten Bindens (Indirect Linkage Mechanism, siehe Handbuch „BLSSERV Bindelader-Starter“ [1]) zur Verknüpfung mit Modulen in der PUBLIC-Slice eingesetzt wird, sollte CONNECTION-MODE=*BY-RESOLUTION bevorzugt werden.

Anmerkung

Wenn der Anwender mit der Anweisung MODIFY-LLM-ATTRIBUTES Subsystem-Entries festgelegt hat, wird die Information dazu in das gespeicherte LLM ausgegeben unabhängig von deren Format. Bei CONNECTION-MODE=*BY-RESOLUTION hat die Angabe von Subsystem-Entries keine Auswirkung.

3.5 Befriedigen von Externverweisen

Externverweise in den Modulen, die in das LLM eingefügt werden, verweisen auf einen Programmabschnitt (CSECT), eine Einsprungstelle (ENTRY) oder einen COMMON-Bereich in einem anderen Modul. Der BINDER versucht, alle Externverweise, sofort zu befriedigen d.h. er sucht innerhalb des erzeugten LLM eine CSECT, einen ENTRY oder einen COMMON-Bereich mit gleichem Namen und trägt die gefundene Adresse in den Externverweis ein.

Zur Befriedigung von noch unbefriedigten Externverweisen kann der Benutzer die **Autolink-Funktion** des BINDER mit der Anweisung RESOLVE-BY-AUTOLINK aufrufen. Die Bibliotheken, die bei RESOLVE-BY-AUTOLINK angegeben werden, durchsucht BINDER nach CSECTs, ENTRYs und COMMONs in Modulen, die die unbefriedigten Externverweise befriedigen. Die entsprechenden Module werden in das aktuelle LLM eingefügt.

Der Benutzer kann angeben, dass Eingabebibliotheken, die mit dem Dateikettungsnamen BLSLIBnn ($00 \leq nn \leq 99$) zugewiesen wurden, durchsucht werden.

Die Bibliotheken werden in der Reihenfolge nach aufsteigenden Werten „nn“ dieses Dateikettungsnamen durchsucht. Sie müssen vor dem BINDER-Lauf zugewiesen werden. Diese Dateikettungsnamen werden nach einem BINDER-Lauf *nicht* freigegeben.

Die Autolink-Funktion erspart vor allem den Benutzern der höheren Programmiersprachen, die oft recht zahlreich benötigten Module des Laufzeitsystems mit Anweisungen INCLUDE-MODULES einzufügen.

Bedingte Externverweise (WXTRNs) können mit Autolink *nicht* befriedigt werden. Die Namen unbefriedigter WXTRNs protokolliert der BINDER nach Ausführung der Funktion Autolink in der Liste der unbefriedigten bedingten Externverweise (siehe [Seite 141ff](#)). Externverweise, auf die im Programm nicht Bezug genommen wird (nicht referenzierte Externverweise), können ebenfalls nicht mit Autolink befriedigt werden.

Hinweis

Die Zeit, die Autolink benötigt, hängt im Wesentlichen davon ab, wie viele Bibliotheken durchsucht werden müssen, bis passende Module gefunden werden. Im ungünstigsten Fall müssen sämtliche angegebene Bibliotheken durchsucht werden.

Eine Liste der unbefriedigten Externverweise wird ausgegeben, wenn sie der Benutzer in den Anweisungen SHOW-MAP oder MODIFY-MAP-DEFAULTS nicht unterdrückt (siehe [Seite 141ff](#)).

Zur Befriedigung von Externverweisen kann der Benutzer abhängig von der Betriebsart des BINDER folgende Maßnahmen treffen:

Dialogbetrieb

Der Benutzer kann weitere Anweisungen INCLUDE-MODULES oder RESOLVE-BY-AUTOLINK eingeben, um noch offene Externverweise zu befriedigen. Will der Benutzer dies nicht, behandelt der BINDER die unbefriedigten Externverweise wie in der Anweisung SET-EXTERN-RESOLUTION festgelegt wurde (siehe [Seite 96ff](#)).

Stapelbetrieb

Nicht befriedigte Externverweise werden behandelt, wie in der Anweisung SET-EXTERN-RESOLUTION festgelegt wurde (siehe [Seite 96ff](#)).

3.5.1 Regeln zur Befriedigung von Externverweisen

BINDER versucht zunächst, Externverweise mit Namen von sichtbaren CSECTs, ENTRYs und COMMON-Bereichen aus Modulen zu befriedigen, die er in das aktuelle LLM eingefügt hat. Ein Externverweis wird jedoch nie mit einem Namen aus einem Modul befriedigt, das im FORBIDDEN-SCOPE des Moduls enthalten ist, das den Verweis enthält.

Für die Befriedigung der Externverweise gelten folgende Regeln:

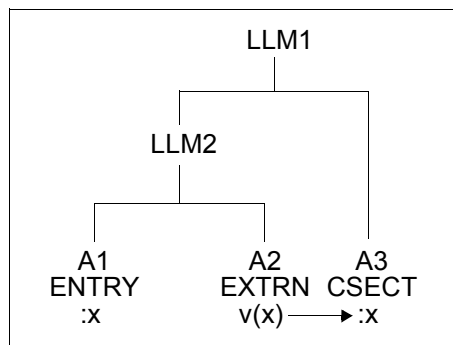
- **Regel 1**
CSECTs, ENTRYs und COMMON-Bereiche im HIGH-PRIORITY-SCOPE des Moduls, das den Externverweis enthält, haben Vorrang vor allen anderen Programmdefinitionen. CSECTs, ENTRYs und COMMON-Bereiche im LOW-PRIORITY-SCOPE dieses Moduls werden erst nach allen anderen Programmdefinitionen berücksichtigt.
- **Regel 2**
Wird ein Modul in ein LLM eingefügt, haben die Namen von CSECTs, ENTRYs und COMMON-Bereichen innerhalb desselben Sub-LLM Vorrang vor den Namen von CSECTs, ENTRYs und COMMON-Bereichen in anderen Zweigen. Beim Durchsuchen wird dabei mit dem Sub-LLM der *höchsten* Stufe begonnen, das Referenzen und Programmdefinitionen enthält.
- **Regel 3**
Eine CSECT hat Vorrang vor einem ENTRY und ein ENTRY hat Vorrang vor einem COMMON-Bereich.
- **Regel 4**
Die Externverweise werden *nacheinander* behandelt. Die OMs und Sub-LLMs des aktuellen LLM werden dabei in der Reihenfolge abgearbeitet, in der sie in der logischen Struktur angeordnet sind (im LLM-Strukturbaum von links nach rechts).

Eine Ausnahme bilden LLMs mit benutzerdefinierten Slices. Für sie gilt vorrangig vor den drei Standardregeln folgende Suchreihenfolge:

1. Suchen nach Definitionen in der Slice, in der sich auch der Externverweis befindet.
2. Suchen nach Definitionen in den Slices mit demselben Pfad und derselben Region wie die Slice, die den Externverweis enthält.
3. Suchen nach Definitionen in Slices, die hinter der Slice mit dem Externverweis liegen.
4. Suchen nach Definitionen in Slices, die in anderen Regionen liegen.
5. Suchen nach Definitionen in konkurrierenden Slices, wenn in der Anweisung START-LLM-CREATION ... EXCLUSIVE-SLICE-CALL=YES angegeben wurde.

Beispiel 1 (Regel 1)

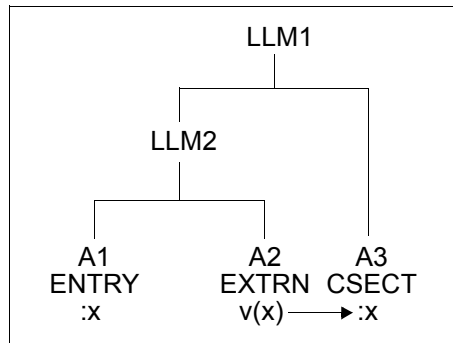
aktuelles LLM



Voraussetzung:

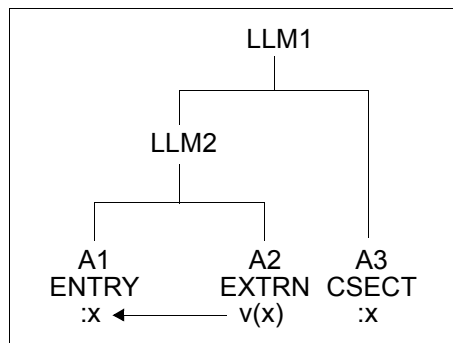
Für das Modul A2 ist ein HIGH-PRIORITY-SCOPE vereinbart, der das Modul A3 enthält.

Um den EXTRN $v(x)$ im Sub-LLM A2 zu befriedigen, durchsucht BINDER zunächst den HIGH-PRIORITY-SCOPE von A2, also A3 und nicht A1, wie es ohne RESOLUTION-SCOPE der Fall wäre. ENTRY x in A3 befriedigt EXTRN $v(x)$.

*Beispiel 2 (Regel 1)*aktuelles LLM**Voraussetzung:**

Für das Sub-LLM LLM2 ist ein LOW-PRIORITY-SCOPE definiert, der das Modul A1 enthält.

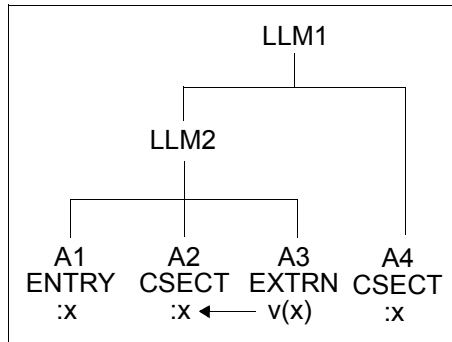
Das Modul A2 erbt den LOW-PRIORITY-SCOPE vom übergeordneten Knoten, d.h. vom Sub-LLM LLM2. BINDER sucht den unbefriedigten EXTRN $v(x)$ im Modul A2 zunächst in Modulen, die nicht im LOW-PRIORITY-SCOPE von A2 enthalten sind. Daher wird EXTRN $v(x)$ mit ENTRY x in A3 befriedigt.

*Beispiel 3 (Regel 2)*aktuelles LLM

Das Bindemodul A2 mit dem EXTRN v(x) wird in das Sub-LLM LLM2 eingefügt. Der EXTRN v(x) im Bindemodul A2 wird vom ENTRY x im Bindemodul A1 befriedigt, da A2 und A1 im selben Sub-LLM LLM2 liegen. Die CSECT x im Bindemodul A3 liegt in einem anderen Zweig und wird nicht berücksichtigt.

Beispiel 4 (Regel 2 und 3)

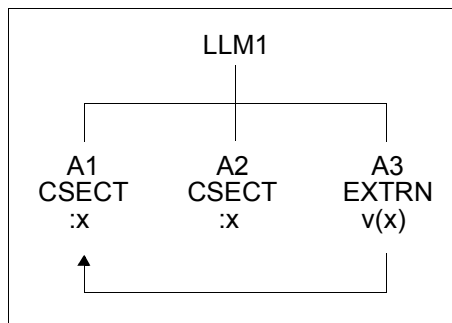
aktuelles LLM



Das Bindemodul A3 mit dem EXTRN v(x) wird in das Sub-LLM LLM2 eingefügt. Der EXTRN v(x) im Bindemodul A3 wird von der CSECT x im Bindemodul A2 befriedigt, da A3 und A2 im selben Sub-LLM LLM2 liegen. Innerhalb LLM2 hat die CSECT x in A2 Vorrang vor dem ENTRY x in A1.

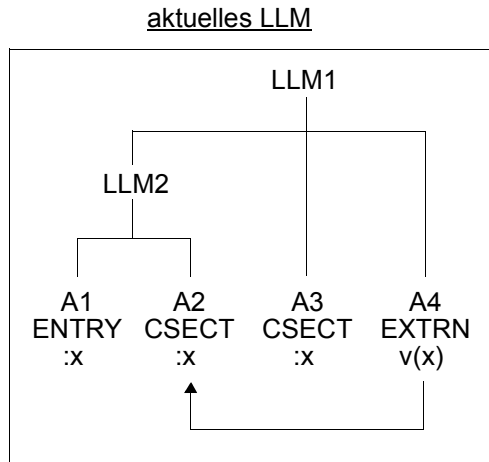
Beispiel 5 (Regel 4)

aktuelles LLM



Das Bindemodul A3 mit dem EXTRN $v(x)$ wird in das LLM1 eingefügt. Der EXTRN $v(x)$ im Bindemodul A3 wird von der CSECT x im Bindemodul A1 befriedigt, da der LLM-Strukturbaum von links nach rechts durchsucht wird.

Beispiel 6 (Regel 2 und 3)



Das Bindemodul A4 mit dem EXTRN $v(x)$ wird in das LLM1 eingefügt. Der EXTRN $v(x)$ im Bindemodul A4 wird von der CSECT x im Bindemodul A2 nach folgenden Regeln befriedigt:

- Beim Durchsuchen wird mit dem Sub-LLM der höchsten Stufe begonnen. Dies ist das Sub-LLM LLM2 (Regel 1).
- Innerhalb des Sub-LLM LLM2 hat die CSECT x im Bindemodul A2 eine höhere Priorität als der ENTRY x im Bindemodul A1 (Regel 2).

3.5.2 Autolink-Funktion

Die Autolink-Funktion des BINDER ermöglicht das automatische Einfügen von Modulen. Zur Befriedigung von Externverweisen sucht BINDER dabei nach CSECTs und ENTRYs in Modulen und Bibliotheken, die in der Anweisung RESOLVE-BY-AUTOLINK angegeben wurden. Dabei gelten folgende Regeln:

- **Regel 1**

Die Externverweise werden *nacheinander* behandelt. Die OMs und Sub-LLMs des aktuellen LLM werden dabei in der Reihenfolge abgearbeitet, in der sie in der logischen Struktur angeordnet sind (im LLM-Strukturbaum von links nach rechts). Innerhalb eines OM werden die Externverweise entsprechend ihrer Reihenfolge im OM behandelt.

Bei Modulen einer Programmbibliothek werden alle Elemente entsprechend dem angegebenen Typ (Operand TYPE) durchsucht.

- **Regel 2**

Wenn ein Modul eingefügt wird, um einen Externverweis zu befriedigen, versucht der BINDER auch, weitere nicht befriedigte Externverweise innerhalb des gesamten LLM mit CSECTs, ENTRYs und COMMON-Bereichen dieses Moduls zu befriedigen.

- **Regel 3**

Sind in einer Anweisung RESOLVE-BY-AUTOLINK *mehrere* Bibliotheken angegeben, werden sie in der Reihenfolge durchsucht, wie sie in der Anweisung angegeben wurden. Für jede einzelne Bibliothek werden die Module nach *Regel 1* durchsucht.

- **Regel 4**

In der Anweisung RESOLVE-BY-AUTOLINK kann ein Geltungsbereich festgelegt werden (Operand SCOPE). Dieser Geltungsbereich definiert den Teil des LLM-Strukturbaums, in dem Externverweise befriedigt werden. Externverweise außerhalb des Geltungsbereichs werden nicht befriedigt.

- **Regel 5**

In der Anweisung RESOLVE-BY-AUTOLINK kann ein Pfadname angegeben werden (Operand PATH-NAME). Er legt fest, in welches Sub-LLM des aktuellen LLM Module eingefügt werden.

- **Regel 6**

Externverweise, die beim Einfügen von Modulen neu hinzukommen, werden in die Liste der *unbefriedigten* Externverweise aufgenommen. Voraussetzung ist, dass sie im angegebenen Geltungsbereich (Operand SCOPE) und im gültigen Pfad (Operand PATH-NAME) definiert sind.

Für die mit RESOLVE-BY-AUTOLINK eingefügten Module kann festgelegt werden, ob sie als Laufzeitmodule zu betrachten sind oder nicht (Operand RUN-TIME-VISIBILITY). Wenn das eingefügte Modul ein Laufzeitmodul sein soll, werden beim Abspeichern alle Symbole dieses Moduls maskiert. Dadurch können Namenskonflikte beim Laden des LLM durch den DBL vermieden werden. Diese Maskierung wird aber zurückgenommen, falls das Modul erneut gelesen wird (z.B. bei INCLUDE-MODULES oder START-LLM-UPDATE).

Hinweis

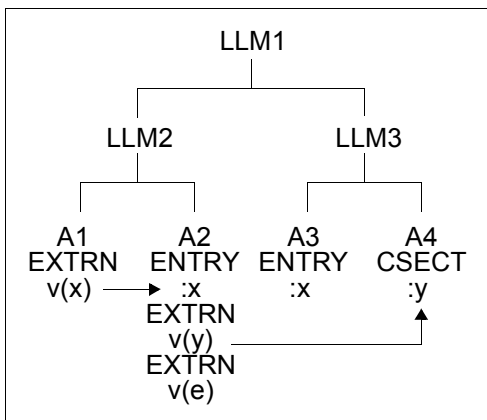
Die Autolink-Funktion wird nicht durchgeführt, wenn die LLMs keine Relativierungsinformation, keine Logische Strukturinformation oder kein Externadressbuch haben oder wenn LLMs mit benutzerdefinierten Slices eingefügt werden sollen. In diesen Fall wird die Autolink-Funktion abgebrochen. Der Benutzer ist selbst dafür verantwortlich, dass keine derartigen Module in Bibliotheken enthalten sind.

Beispiel 1

Stufe 1:

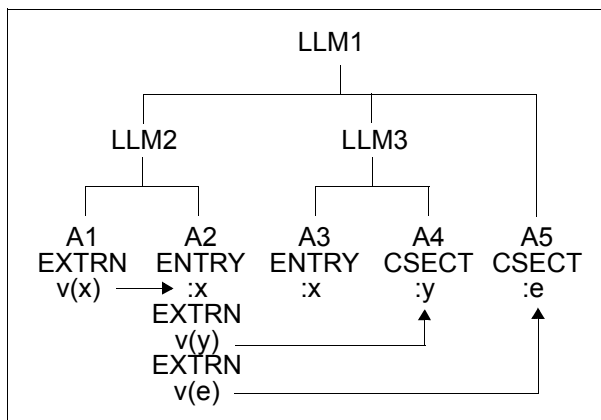
Der EXTRN $v(x)$ im Bindemodul A1 wird vom ENTRY x im Bindemodul A2 befriedigt, da A1 und A2 im selben Sub-LLM liegen. Der ENTRY x im Bindemodul A3 liegt in einem anderen Zweig und wird nicht berücksichtigt.

Den EXTRN $v(y)$ im Bindemodul A2 befriedigt die CSECT y im Bindemodul A4.

aktuelles LLM

Stufe 2:

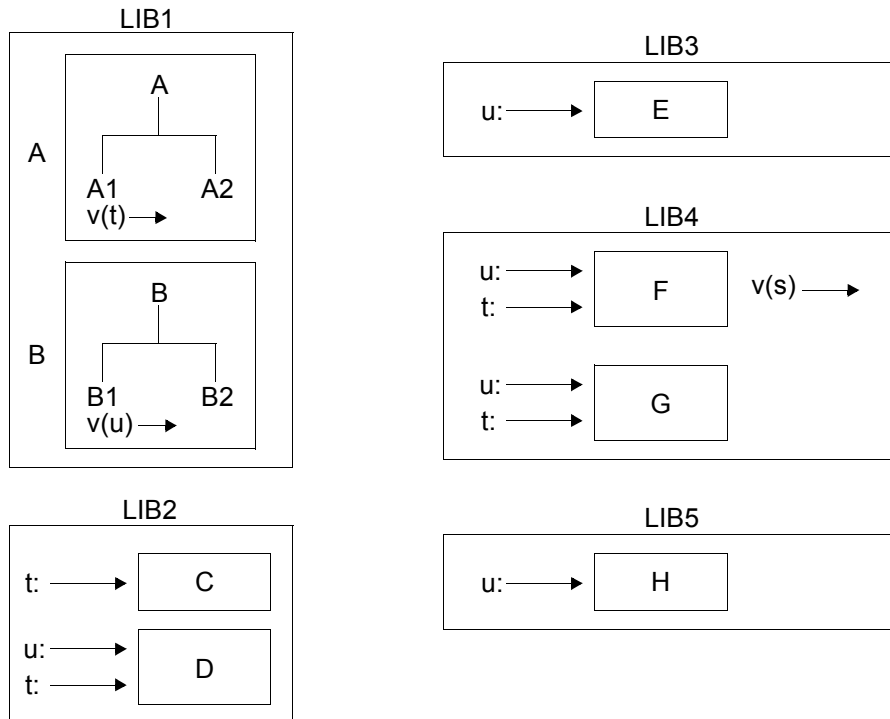
Der noch offene EXTRN $v(e)$ im Bindemodul A2 wird von der CSECT e des Bindemoduls A5 befriedigt, das mit Autolink eingefügt wird.

aktuelles LLM

Beispiel 2

Gegeben sind folgende Programmbibliotheken (siehe unten):

- Eine Programmbibliothek LIB1.
Sie enthält das LLM A mit einem unbefriedigten EXTRN $v(t)$ und das LLM B mit einem unbefriedigten EXTRN $v(u)$.
- Eine Programmbibliothek LIB2.
Sie enthält das OM C mit dem ENTRY t und das OM D mit zwei ENTRYs u und t .
- Eine Programmbibliothek LIB3.
Sie enthält das OM E mit dem ENTRY u .
- Eine Programmbibliothek LIB4.
Sie enthält folgende Module:
 - das OM F mit dem unbefriedigten EXTRN $v(s)$ und zwei ENTRYs u und t ,
 - das OM G mit zwei ENTRYs s und t .
- Eine Programmbibliothek LIB5.
Sie enthält das OM H mit dem ENTRY u .

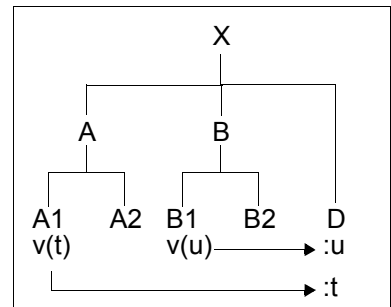
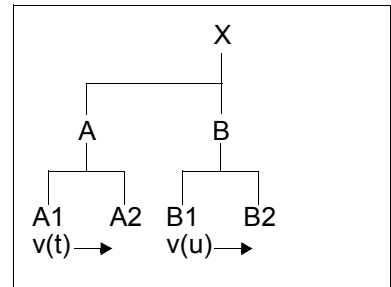
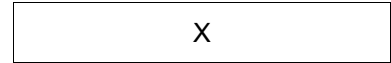


Fall 1 (Regel 1 und 2)

Anweisungen

- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B),TYPE=L
- (3) RESOLVE-BY-AUTOLINK
LIBRARY=LIB2,TYPE=(L,R),
SYMBOL-NAME=U

aktuelles LLM
(Arbeitsbereich)



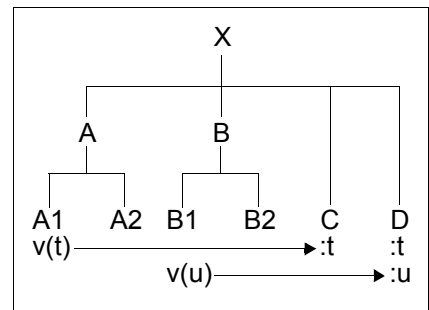
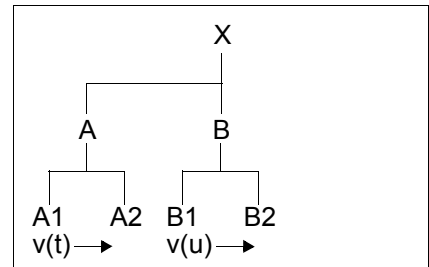
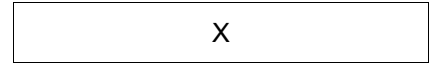
Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die LLMs A und B werden aus der Programmbibliothek LIB1 in das aktuelle LLM eingefügt.
- (3) Die Programmbibliothek LIB2 wird nach dem unbefriedigten EXTRN v(u) durchsucht. Das OM D befriedigt den EXTRN v(u). Der BINDER fügt das OM D an der Root des aktuellen LLM ein und versucht, den unbefriedigten EXTRN v(t) mit einem ENTRY im OM D zu befriedigen. Der ENTRY t im OM D befriedigt den EXTRN v(t).

Fall 2 (Regel 1)

Anweisungen

- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B),TYPE=L
- (3) RESOLVE-BY-AUTOLINK
LIBRARY=LIB2,TYPE=(L,R)

aktuelles LLM
(Arbeitsbereich)

Bedeutung

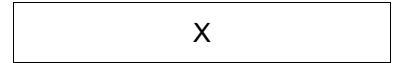
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die LLMs A und B werden aus der Programmbibliothek LIB1 in das aktuelle LLM eingefügt.
- (3) Die Programmbibliothek LIB2 wird nach *allen* unbefriedigten EXTRNs durchsucht (Standardwert SYMBOL-NAME=*ALL). Das OM C befriedigt den EXTRN v(t), das OM D den EXTRN v(u). Der BINDER fügt die OMs C und D an der Root des aktuellen LLM ein.

Fall 3 (Regel 3, 4 und 5)

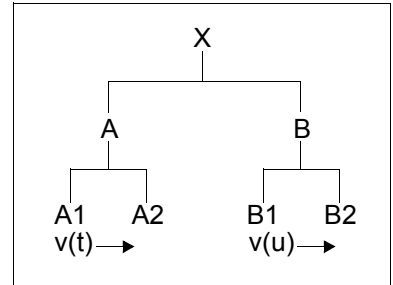
Anweisungen

(1) START-LLM-CREATION
INTERNAL-NAME=X

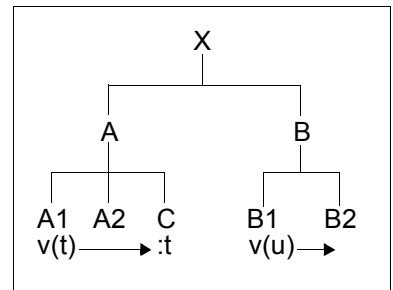
aktuelles LLM
(Arbeitsbereich)



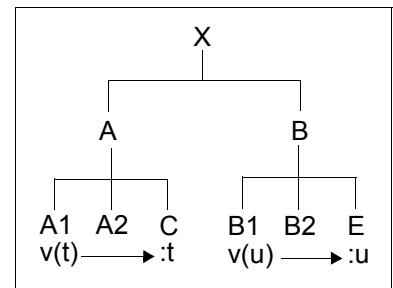
(2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B),TYPE=L



(3) RESOLVE-BY-AUTOLINK
LIBRARY=LIB2,TYPE=(L,R),
SYMBOL-NAME=T,
PATH-NAME=X.A



(4) RESOLVE-BY-AUTOLINK
LIBRARY=(LIB3,LIB4,LIB5),
TYPE=(L,R),
SCOPE=EXPLICIT(WITHIN-SUB-LLM=B),
PATH-NAME=X.B



Bedeutung

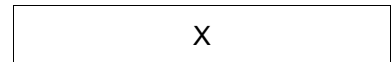
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die LLMs A und B werden aus der Programmbibliothek LIB1 in das aktuelle LLM eingefügt.
- (3) Die Programmbibliothek LIB2 wird nach dem unbefriedigten EXTRN v(t) durchsucht. Der erste zugehörige ENTRY t wird im OM C gefunden. Der BINDER fügt das OM C in das Sub-LLM A ein (Pfadname X.A).
- (4) Die Programmbibliotheken LIB3, LIB4, LIB5 werden in der Reihenfolge LIB3, LIB4, LIB5 nach allen noch nicht befriedigten EXTRNs durchsucht (Standardwert SYMBOL-NAME=*ALL). Nur die unbefriedigten EXTRNs innerhalb des Sub-LLM B sollen berücksichtigt werden (SCOPE-Operand). Der erste zugehörige ENTRY u wird in LIB3 im OM E gefunden. Der BINDER fügt das OM E in das Sub-LLM B ein (Pfadname X.B).

Fall 4 (Regel 3, 4 und 5)

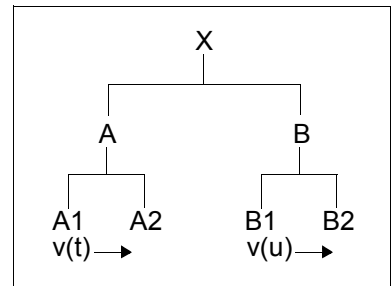
Anweisungen

aktuelles LLM
(Arbeitsbereich)

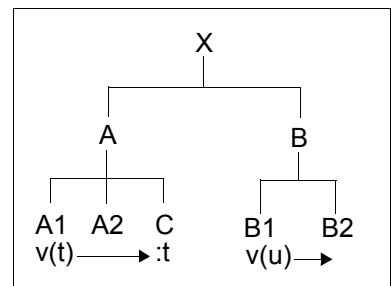
- (1) START-LLM-CREATION
INTERNAL-NAME=X



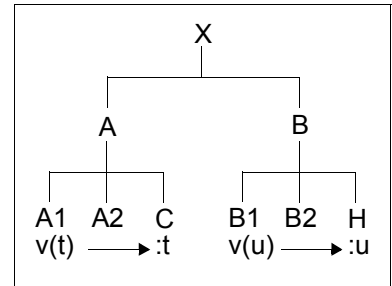
- (2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B),TYPE=L



- (3) RESOLVE-BY-AUTOLINK
LIBRARY=LIB2,TYPE=(L,R),
SYMBOL-NAME=T,
PATH-NAME=X.A



- (4) RESOLVE-BY-AUTOLINK
 LIBRARY=(LIB5,LIB4,LIB3),
 TYPE=(L,R),
 SCOPE=EXPLICIT(WITHIN-SUB-LLM=B),
 PATH-NAME=X.B



Bedeutung

(1), (2), (3) wie in Fall 3.

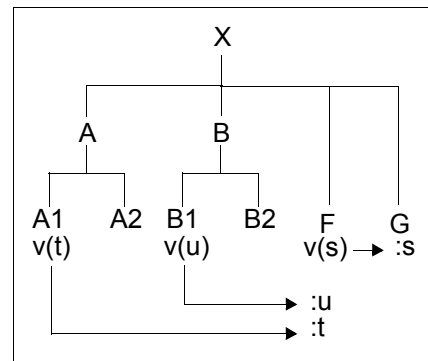
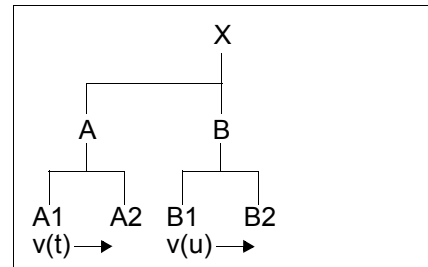
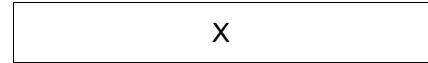
- (4) Die Programmbibliotheken LIB3, LIB4, LIB5 werden hier in der Reihenfolge LIB5, LIB4, LIB3 nach allen noch nicht befriedigten EXTRNs durchsucht (Standardwert SYMBOL-NAME=*ALL). Nur die unbefriedigten EXTRNs innerhalb des Sub-LLM B sollen berücksichtigt werden (SCOPE-Operand). Der erste zugehörige ENTRY u wird in LIB5 im OM H gefunden. Der BINDER fügt das OM H in das Sub-LLM B ein (Pfadname X.B).

Fall 5 (Regel 6)

Anweisungen

- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B),TYPE=L
- (3) RESOLVE-BY-AUTOLINK
LIBRARY=LIB4,TYPE=(L,R)

aktuelles LLM
(Arbeitsbereich)



Bedeutung

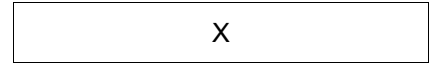
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die LLMs A und B werden aus der Programmbibliothek LIB1 in das aktuelle LLM eingefügt.
- (3) Die Programmbibliothek LIB4 wird nach allen unbefriedigten EXTRNs durchsucht (Standardwert SYMBOL-NAME=*ALL). Die OMs in der Programmbibliothek LIB4 werden in alphabetischer Reihenfolge durchsucht. Das OM F befriedigt die EXTRNs v(t) und v(u). Der BINDER fügt das OM F an der Root des aktuellen LLM ein. Der unbefriedigte Externverweis v(s) im OM F wird in die Liste der unbefriedigten Externverweise eingetragen. Der BINDER durchsucht die Programmbibliothek LIB4 nach einem Modul, das den EXTRN v(s) befriedigt, und findet das OM G. Das OM G wird an der Root des aktuellen LLM eingefügt.

Fall 6 (Regel 4 und 6)

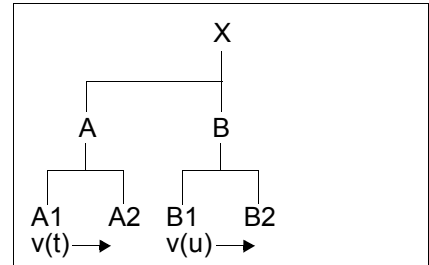
Anweisungen

(1) START-LLM-CREATION
INTERNAL-NAME=X

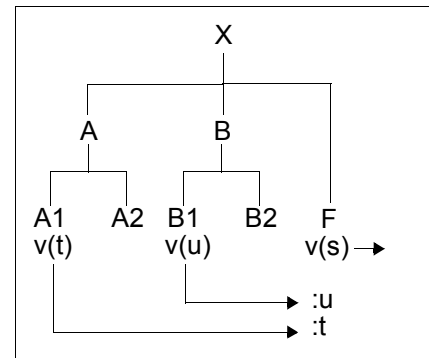
aktuelles LLM
(Arbeitsbereich)



(2) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B),TYPE=L



(3) RESOLVE-BY-AUTOLINK
LIBRARY=LIB4,TYPE=(L,R)
SCOPE=EXPLICIT(WITHIN-SUB-LLM=B)



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die LLMs A und B werden aus der Programmbibliothek LIB1 in das aktuelle LLM eingefügt.
- (3) Die Programmbibliothek LIB4 wird nach *allen* unbefriedigten EXTRNs des LLM B durchsucht (Standardwert SYMBOL-NAME=*ALL). Die OMs in der Programmbibliothek LIB4 werden in alphabetischer Reihenfolge durchsucht. Das OM F befriedigt den EXTRN v(u). Der BINDER fügt das OM F an der Root des aktuellen LLM ein. Der unbefriedigte Externverweis v(s) im OM F wird in die Liste der unbefriedigten Externverweise eingetragen. Der EXTRN v(s) bleibt unbefriedigt, da er außerhalb des Geltungsbereichs liegt. Obwohl der EXTRN v(t) außerhalb des Geltungsbereichs liegt, wird er befriedigt, da das Modul F mit dem ENTRY t eingefügt wurde.

3.5.3 Behandeln unbefriedigter Externverweise

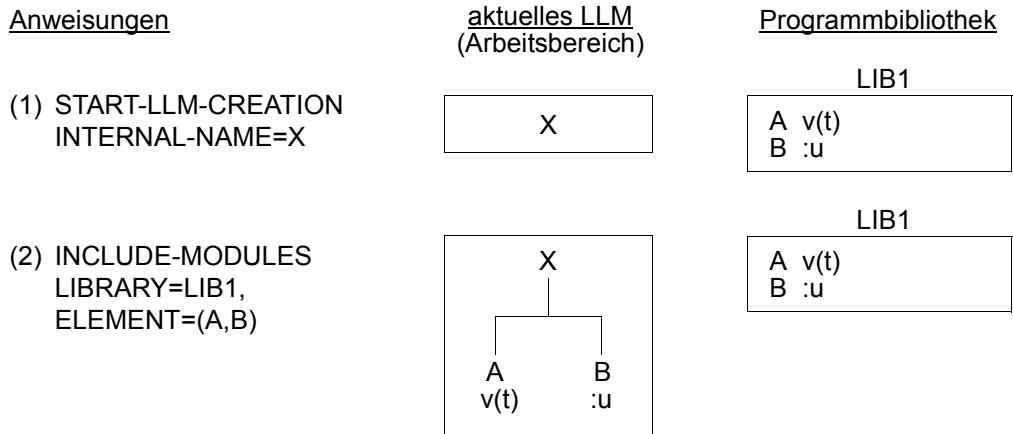
Mit der Anweisung SET-EXTERN-RESOLUTION legt der Benutzer für das aktuelle LLM fest, wie der BINDER verbleibende Externverweise behandeln soll, die nicht befriedigt werden können. Es kann festgelegt werden, dass nicht befriedigte Externverweise zulässig oder unzulässig sind. Zulässige unbefriedigte Externverweise können mit der Adresse eines angegebenen Symbols besetzt werden.

Sind nicht befriedigte Externverweise zulässig, werden sie beim Speichern des LLM übernommen. Falls nicht befriedigte Externverweise unzulässig sind, wird das LLM beim Speichern abgewiesen.

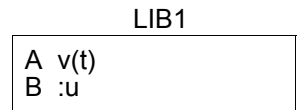
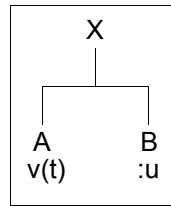
Die Anweisung SET-EXTERN-RESOLUTION wird erst beim Speichern des aktuellen LLM mit der Anweisung SAVE-LLM wirksam. Das aktuelle LLM im Arbeitsbereich bleibt unverändert. Wenn zwischen den Anweisungen SET-EXTERN-RESOLUTION und der Anweisung SAVE-LLM mit einer Anweisung INCLUDE-MODULES Module eingefügt werden, und ein eingefügtes Modul die unbefriedigten Externverweise befriedigen kann, wird die Anweisung SET-EXTERN-RESOLUTION übergangen.

Der Geltungsbereich für die Behandlung von unbefriedigten Externverweisen kann auf bestimmte OMs und Sub-LLMs im aktuellen LLM eingeschränkt werden. (Operand SCOPE).

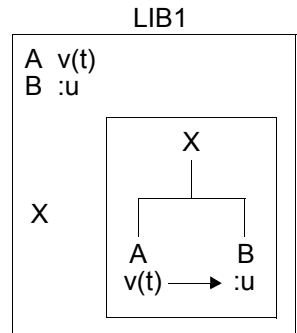
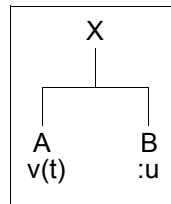
Beispiel 1



- (3) SET-EXTERN-RESOLUTION
 SYMBOL-NAME=T,
 SYMBOL-TYPE=REFERENCES
 RESOLUTION=BY-SYMBOL
 (SYMBOL=U)



- (4) SAVE-LLM
 LIBRARY=LIB1



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die Bindemodule A und B werden aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt. Bindemodul A enthält den unbefriedigten EXTRN v(t), Bindemodul B den ENTRY u. Der EXTRN v(t) bleibt unbefriedigt, da er vom ENTRY u nicht befriedigt werden kann.
- (3) Es wird festgelegt, dass der unbefriedigte EXTRN v(t) beim Speichern des aktuellen LLM mit der Adresse des ENTRY u besetzt wird. Das aktuelle LLM im Arbeitsbereich bleibt unverändert.
- (4) Das aktuelle LLM wird als Element mit dem Elementnamen X in der Programmbibliothek LIB1 gespeichert. Die Anweisung SET-EXTERN-RESOLUTION wird ausgeführt. Der EXTRN v(t) wird mit der Adresse des ENTRY u belegt.

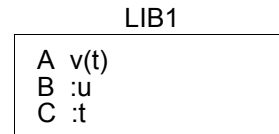
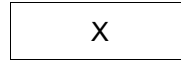
Beispiel 2

Anweisungen

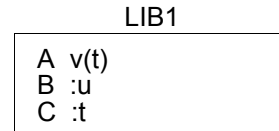
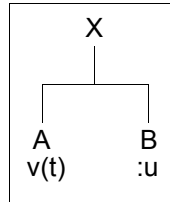
aktuelles LLM
(Arbeitsbereich)

Programmbibliothek

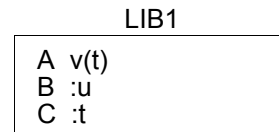
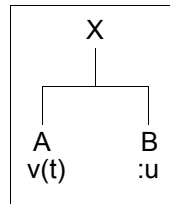
(1) START-LLM-CREATION
INTERNAL-NAME=X



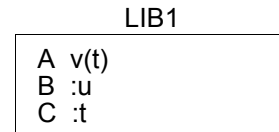
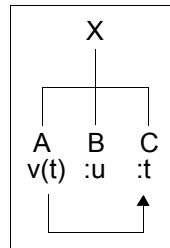
(2) INCLUDE-MODULES
LIBRARY=LIB1,
ELEMENT=(A,B)



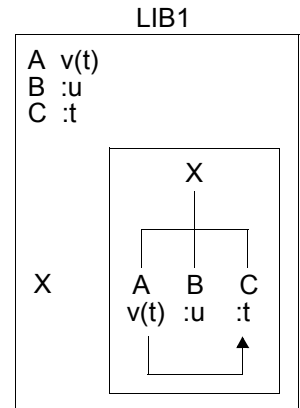
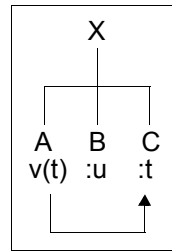
(3) SET-EXTERN-RESOLUTION
SYMBOL-NAME=T,
SYMBOL-TYPE=REFERENCES
RESOLUTION=BY-SYMBOL
(SYMBOL=U)



(4) INCLUDE-MODULES
LIBRARY=LIB1,
ELEMENT=C



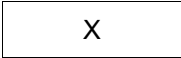

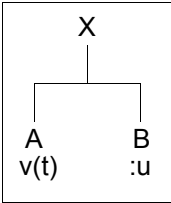

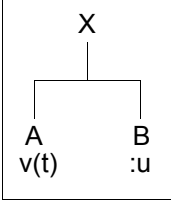

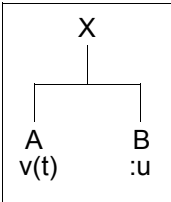
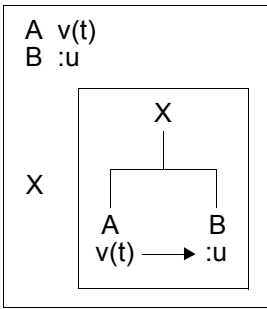
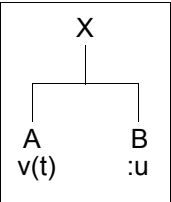
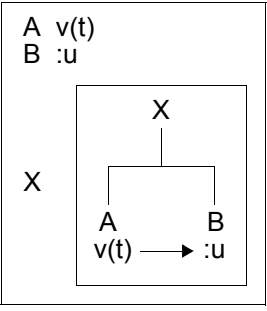
- (5) SAVE-LLM
LIBRARY=LIB1,
ELEMENT=X



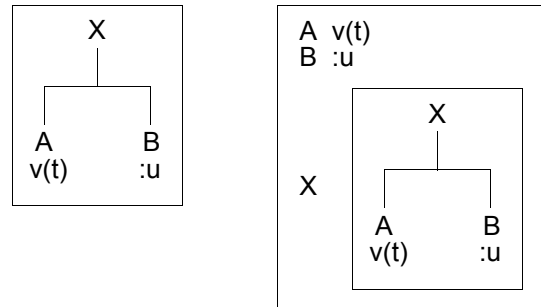
Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die Bindemodule A und B werden aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt. Bindemodul A enthält den unbefriedigten EXTRN v(t), Bindemodul B den ENTRY u. Der EXTRN v(t) bleibt unbefriedigt, da er vom ENTRY u nicht befriedigt werden kann.
- (3) Es wird festgelegt, dass der unbefriedigte EXTRN v(t) beim Speichern des aktuellen LLM mit der Adresse des ENTRY t besetzt wird. Das aktuelle LLM im Arbeitsbereich bleibt unverändert.
- (4) Das Bindemodul C mit dem ENTRY t wird aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt. Der ENTRY t kann EXTRN v(t) befriedigen.
- (5) Das aktuelle LLM wird als Element mit dem Elementnamen X in der Programmbibliothek LIB1 gespeichert. Da der EXTRN v(t) von ENTRY t befriedigt wurde, wird die Anweisung SET-EXTERN-RESOLUTION übergangen.

Beispiel 3

<u>Anweisungen</u>	<u>aktuelles LLM</u> (Arbeitsbereich)	<u>Programmbibliothek</u>
(1) START-LLM-CREATION INTERNAL-NAME=X		<p>LIB1</p> 
(2) INCLUDE-MODULES LIBRARY=LIB1, ELEMENT=(A,B)		<p>LIB1</p> 
(3) SET-EXTERN-RESOLUTION SYMBOL-NAME=T, SYMBOL-TYPE=REFERENCES RESOLUTION=BY-SYMBOL (SYMBOL=U)		<p>LIB1</p> 
(4) SAVE-LLM LIBRARY=LIB1		<p>LIB1</p> 
(5) START-LLM-UPDATE LIBRARY=LIB1 ELEMENT=X		<p>LIB1</p> 

(6) SAVE-LLM
LIBRARY=LIB1



Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die Bindemodule A und B werden aus der Programmbibliothek LIB1 gelesen und in das aktuelle LLM eingefügt. Bindemodul A enthält den unbefriedigten EXTRN v(t), Bindemodul B den ENTRY u. Der EXTRN v(t) bleibt unbefriedigt, da er vom ENTRY u nicht befriedigt werden kann.
- (3) Es wird festgelegt, dass der unbefriedigte EXTRN v(t) beim Speichern des aktuellen LLM mit der Adresse des ENTRY u besetzt wird. Das aktuelle LLM im Arbeitsbereich bleibt unverändert.
- (4) Das aktuelle LLM wird als Element mit dem Elementnamen X in der Programmbibliothek LIB1 gespeichert. Die Anweisung SET-EXTERN-RESOLUTION wird ausgeführt. Der EXTRN v(t) wird mit der Adresse des ENTRY u belegt.
- (5) Dasselbe LLM mit dem Elementnamen X in der Programmbibliothek LIB1 wird geändert.
- (6) Das aktuelle LLM wird als Element mit dem Elementnamen X in der Programmbibliothek LIB1 gespeichert. Die Angaben unter (3) in der Anweisung SET-EXTERN-RESOLUTION werden nicht mehr berücksichtigt. Das LLM wird mit unbefriedigtem EXTRN v(t) gespeichert (Standardwert in der Anweisung SET-EXTERN-RESOLUTION).

3.6 Behandlung von Namenskonflikten

Namenskonflikte können entstehen, wenn im Externadressbuch (ESV) des LLM mehrere Einträge mit gleichem Namen vorkommen. Aber nicht jede Namensgleichheit ist auch ein Namenskonflikt.

In der folgenden Tabelle ist das Verhalten des BINDER bei Namensgleichheit dargestellt.

Eintrag 2	Eintrag 1			
	CSECT	ENTRY	COMMON	XDSEC-D
CSECT	(1)	(1)	(2)	—
ENTRY	(1)	(1)	(1)	—
COMMON	(2)	(1)	(3)	—
XDSEC-D	—	—	—	(4)

Bedeutung

- (1) Ein Namenskonflikt wurde entdeckt.
Der BINDER akzeptiert den Namenskonflikt und versucht ihn zu beheben. Dabei geht er nach den Regeln vor, die beim Befriedigen von Externverweisen beschrieben sind (siehe [Seite 80ff](#)).

Der Benutzer findet Informationen über Namenskonflikte in den Listen, die mit der Anweisung SHOW-MAP ausgegeben werden (siehe [Seite 141ff](#)). Er kann dann durch Umbenennen von Symbolen oder Ändern der Maskierung von Symbolen selbst Maßnahmen treffen, um den Namenskonflikt zu beheben (siehe [Seite 105ff](#)).
- (2) Der BINDER wählt die Länge des COMMON-Bereichs so groß, dass die längste CSECT dieses Namens bzw. der längste COMMON-Bereich dieses Namens hineinpasst.
- (3) Der BINDER wählt die Länge des COMMON-Bereichs so groß, dass der längste COMMON-Bereich hineinpasst.
- (4) Ein Namenskonflikt wurde entdeckt.
Der BINDER geht vor wie in Punkt (1) beschrieben. Definitionen von XDSECs können jedoch vom Benutzer *nicht* umbenannt oder maskiert werden.

In den Anweisungen

INCLUDE-MODULES,
MERGE-MODULES,
MODIFY-MODULE-ATTRIBUTES,
MODIFY-SYMBOL-VISIBILITY,
RENAME-SYMBOLS,
REPLACE-MODULS,
RESOLVE-BY-AUTOLINK und
SAVE-LLM

kann mit dem Operanden NAME-COLLISION die Behandlung eines Namenskonfliktes, den die jeweilige Anweisung verursacht hat, gesteuert werden. Andere eventuell vorhandene Namenskonflikte werden davon jedoch nicht berührt.

3.7 Behandlung der COMMON-Bereiche

COMMON-Bereiche sind Abschnitte, die zum Zeitpunkt des Bindens noch keine Daten oder Befehle enthalten, sondern nur Platz dafür reservieren. Diese Bereiche können nach dem Laden des LLM als Daten-Kommunikationsbereiche zwischen verschiedenen Modulen des LLM verwendet oder als Platz für CSECTs eingesetzt werden.

Der BINDER weist COMMON-Bereichen gleichen Namens *einen* gemeinschaftlichen Speicherbereich zu. Diesen Bereich wählt er so groß, dass der längste COMMON-Bereich bzw. die längste CSECT, die einen COMMON-Bereich dieses Namens initialisiert, hineinpasst. Die COMMON-Bereiche haben nach dem Initialisieren die gleiche Ladeadresse wie die CSECT, durch die sie initialisiert wurden. Diese Initialisierung wird als „Common Promotion“ bezeichnet.

Bei LLMs mit Einzel-Slice oder mit nach Attributen gebildeten Slices werden alle COMMON-Bereiche gleichen Namens mit der ersten CSECT dieses Namens initialisiert, die beim Durchsuchen des LLM-Strukturbaumes von links nach rechts gefunden wird.

Bei LLMs mit benutzerdefinierten Slices werden die COMMON-Bereiche slice-weise initialisiert: In jeder Slice werden die COMMON-Bereiche gleichen Namens durch die erste CSECT in eben dieser Slice initialisiert, die beim Durchsuchen des LLM-Strukturbaumes von links nach rechts gefunden wird.

Bei LLMs, die kein Externadressbuch haben, kann ein nicht initialisierter COMMON-Bereich zum Ladezeitpunkt auch nicht initialisiert werden.

Unbenannte COMMON-Bereiche behandelt der BINDER wie benannte, mit der Ausnahme, dass er das Namensfeld nicht mit den Namen von CSECTs vergleicht. Text in unbenannten CSECTs wird also nicht verwendet, um einen COMMON-Bereich zu initialisieren.

3.8 Behandlung der Pseudoregister

Pseudoregister sind Hauptspeicherbereiche, die der Kommunikation verschiedener Programmteile untereinander dienen. Die Sprachübersetzer berechnen Ausrichtung und Länge der Pseudoregister und geben diese Informationen als ESV-Informationen an den BINDER weiter. Der BINDER fasst die Pseudoregister in den Modulen zu **Pseudoregistervektoren** zusammen und berechnet die Ausrichtung und die maximale Länge der Pseudoregistervektoren. Die maximale Länge eines Pseudoregistervektors darf maximal 4096 Byte betragen.

Der BINDER reserviert nicht den Hauptspeicherbereich für die Pseudoregistervektoren. Der Benutzer muss den notwendigen Hauptspeicherbereich selbst reservieren.

3.9 Relativierung der Adressen

Jedes Modul, das in das LLM eingefügt wird, besteht aus einer oder mehreren CSECTs, deren Adressen auf den Beginn des zugehörigen Moduls bezogen sind. Der BINDER ordnet beim Binden des LLM den einzelnen CSECTs relative Adressen zu. Als Bezugsadresse wird dabei die Adresse der ersten CSECT des LLM verwendet. Alle weiteren CSECTs in den Modulen, die in das LLM eingefügt werden, erhalten eine Adresse relativ zu dieser Bezugsadresse. Zusätzlich werden sämtliche Adressbezüge in den CSECTs ihrer relativen Lage angepasst. Hat eine CSECT z.B. die relative Lage 300, bezogen auf die Adresse der ersten CSECT im LLM, werden sämtliche Adressbezüge in der CSECT um 300 erhöht.

Wenn die angegebene Ladeadresse größer oder kleiner als die Bezugsadresse des LLM ist, legt der DBL die absoluten Adressen des LLM fest, indem er aus der Summe von Bezugsadresse und Ladeadresse eine Adresskonstante ermittelt und diese zu den relativen Adressen der CSECTs addiert.

3.10 Behandlung von Symbolen

Unter dem Oberbegriff **Symbole** sind im folgenden **Programmdefinitionen** und **Referenzen** in einem LLM zusammengefasst. Jedes Symbol ist durch seinen Namen gekennzeichnet.

Programmdefinitionen sind:

- Programmabschnitte (CSECTs)
- Einsprungstellen (ENTRYs)
- COMMON-Bereiche
- Externe Pseudoabschnitte als Definitionen (XDSECS-D)

Referenzen sind:

- Externverweise (EXTRNs)
- V-Konstanten
- bedingte Externverweise (WXTRNs)
- Externe Pseudoabschnitte als Referenzen (XDSECS-R)

Symbole behandelt der BINDER mit folgenden Funktionen:

- Ändern der Namen von Symbolen (RENAME-SYMBOLS)
- Ändern der Attribute von Symbolen (MODIFY-SYMBOL-ATTRIBUTES)
- Ändern der Maskierung von Symbolen (MODIFY-SYMBOL-VISIBILITY)
- Ändern des Typs von Symbolen (MODIFY-SYMBOL-TYPE)

3.10.1 Symbolnamen

Neben den herkömmlichen EN-Namen (external names) unterstützt BINDER einen weiteren Typ von Symbolnamen: die EEN-Namen (extended external names).

- EN-Namen sind maximal 32 Zeichen lang und dürfen nur die folgenden Zeichen enthalten:

A-Z, a-z, 0-9, @, #, \$, _, &, %, -

Dabei dürfen - und % nicht am Anfang stehen.

- EEN-Namen können sich aus bis zu 32723 beliebigen (auch nicht-abdruckbaren) Zeichen zusammensetzen. Solche Namen werden insbesondere von Compilern für objektorientierte Programmiersprachen (z.B. C/C++ V3.0) benötigt.

Zur Ein- und Ausgabe dieser Namen an Benutzerschnittstellen sind besondere Maßnahmen erforderlich:

- Die Eingabe von EEN-Namen über Benutzerschnittstellen des BINDER ist nicht zulässig. EEN-Namen können nur direkt von entsprechenden Compilern erzeugt wurden und in den von diesen Compilern generierten LLMs enthalten sein.
- Die Ausgabe der EEN-Namen in Listen findet folgendermaßen statt:

BINDER begrenzt die darstellbare Länge für die Ausgabe von EEN-Namen auf 32 Zeichen. Der Compiler, der den Namen generiert hat, stellt einen Algorithmus zur Verfügung. Mit diesem ermittelt BINDER eine abdruckbare Zeichenkette, die den EEN-Namen in dieser Maximallänge repräsentiert. Diese Zeichenfolge ist zwangsläufig nicht eindeutig und dient deshalb nur zur Information. Insbesondere kann ein bestimmter EEN-Name von einer anderen Anwendung unterschiedlich dargestellt werden, wenn diese eine andere darstellbare Länge festlegt. Aus diesem Grund können solche abdruckbaren Name auch nie als Eingabeparameter dienen.

Hinweise

- Der Typ eines Symbolnamens kann nicht geändert werden.
- Beide Typen von Symbolnamen können in einem LLM nebeneinander vorkommen.
- Der Typ eines Symbolnames ist bei der Prüfung auf Namensgleichheit nicht relevant.
- Zum Laden von LLMs, die EEN-Namen enthalten, ist BLSSERV erforderlich. D.h. sie können in BS2000-Versionen kleiner V3.0 nicht geladen werden, und in BS2000 V3.0 nur, wenn BLSSERV ab Version V2.0 eingesetzt wird.

3.10.2 Ändern von Symbolnamen

Die Namen von Programmdefinitionen und Referenzen in einem LLM können mit der Anweisung `RENAME-SYMBOLS` geändert werden.

Folgende Programmdefinitionen können umbenannt werden:

- Programmabschnitte (CSECTs)
- Einsprungstellen (ENTRYs)
- COMMON-Bereiche

Folgende Referenzen können umbenannt werden:

- Externverweise (EXTRNs)
- V-Konstanten
- bedingte Externverweise (WXTRNs)

Programmdefinitionen und Referenzen können *gleichzeitig* umbenannt werden. Welcher Typ umbenannt wird, legt der Operand `SYMBOL-TYPE` fest.

Mit der Anweisung `RENAME-SYMBOLS` können auch *maskierte* Symbole umbenannt werden. Beim Umbenennen werden Externverweise zu den maskierten Symbolen aufgelöst, falls dies in der Anweisung `MODIFY-SYMBOL-VISIBILITY` gefordert wurde.

Der Geltungsbereich für das Umbenennen kann auf bestimmte OMs und Sub-LLMs im aktuellen LLM eingeschränkt werden.

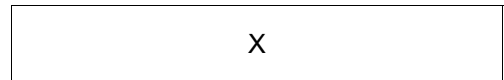
Die Behandlung von eventuell auftretenden Namenskonflikten kann mit dem Operanden `NAME-COLLISION` gesteuert werden.

Beispiel 1

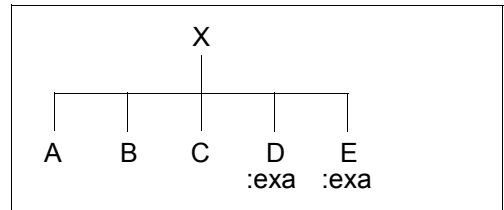
Dieses Beispiel zeigt das Umbenennen von ENTRYs in einem LLM. Dabei wird ein Binde-modul ausgeschlossen.

Anweisungenaktuelles LLM
(Arbeitsbereich)

(1) START-LLM-CREATION
INTERNAL-NAME=X

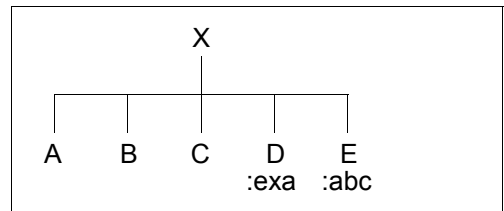


(2) INCLUDE-MODULES
LIBRARY=LIB,
ELEMENT=(A,B,C)



(3) INCLUDE-MODULES
LIBRARY=LIB,
ELEMENT=(D,E)

(4) RENAME-SYMBOLS
SYMBOL-NAME=EXA,
SYMBOL-TYPE=ENTRY,
SCOPE=EXPLICIT(EXCEPT-
SUB-LLM=X.D),
NEW-NAME=ABC



Bedeutung

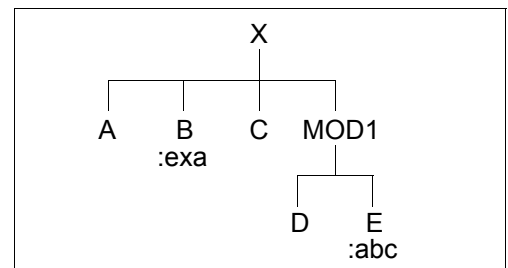
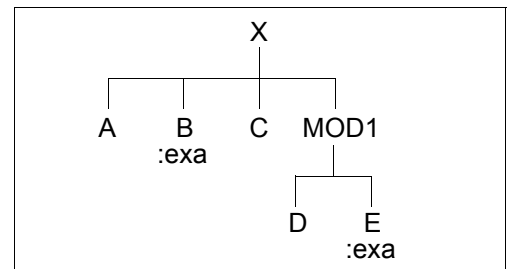
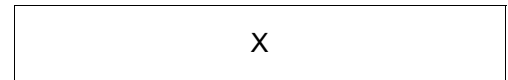
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die OMs A, B und C werden aus der Programmbibliothek LIB in das aktuelle LLM eingefügt.
- (3) Die OMs D und E werden aus der Programmbibliothek LIB in das aktuelle LLM eingefügt. D und E haben ENTRYs mit dem Namen „exa“.
- (4) Nur der Name „exa“ des ENTRY im Modul E wird in „abc“ geändert. Der Name „exa“ des ENTRY im Modul D bleibt unverändert, da er mit dem Operanden SCOPE ausgeschlossen wurde.

Beispiel 2

Dieses Beispiel zeigt das Umbenennen von ENTRYs in einem LLM. Dabei wird nur ein bestimmtes Sub-LLM berücksichtigt.

Anweisungen

- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) INCLUDE-MODULES
LIBRARY=LIB,
ELEMENT=(A,B,C)
- (3) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=MOD1
- (4) INCLUDE-MODULES
LIBRARY=LIB,
ELEMENT=(D,E)
- (5) RENAME-SYMBOLS
SYMBOL-NAME=EXA,
SYMBOL-TYPE=ENTRY,
SCOPE=EXPLICIT(WITHIN-SUB-
LLM=X.MOD1),
NEW-NAME=ABC
- (6) END-SUB-LLM-STATEMENTS

aktuelles LLM
(Arbeitsbereich)

Bedeutung

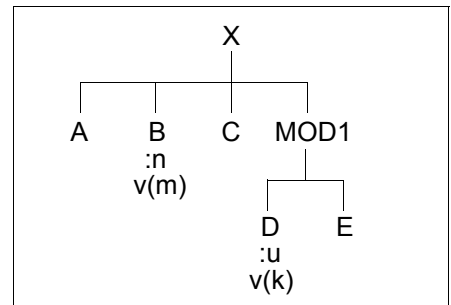
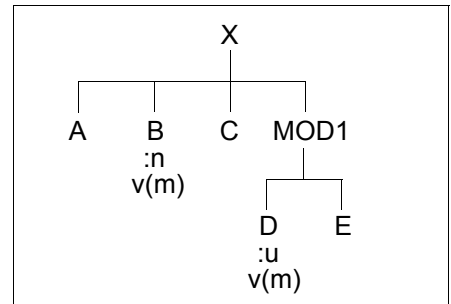
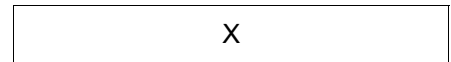
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die OMs A, B und C werden aus der Programmbibliothek LIB in das aktuelle LLM eingefügt. B hat einen ENTRY mit dem Namen „exa“.
- (3) Ein Sub-LLM mit dem Namen MOD1 wird an der Root begonnen.
- (4) In das aktuelle Sub-LLM werden aus der Programmbibliothek LIB die OMs D und E eingefügt. E hat ebenfalls einen ENTRY mit dem Namen „exa“.
- (5) Nur der Name „exa“ des ENTRY im Modul E wird in „abc“ geändert, da mit dem Operanden SCOPE das Umbenennen auf das Sub-LLM MOD1 beschränkt wurde. Der Name „exa“ des ENTRY im Modul B bleibt unverändert.
- (6) Das aktuelle Sub-LLM wird abgeschlossen.

Beispiel 3

Dieses Beispiel zeigt das Umbenennen von EXTRNs in einem LLM. Dabei wird nur ein bestimmtes Sub-LLM berücksichtigt.

Anweisungen

- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) INCLUDE-MODULES
LIBRARY=LIB,
ELEMENT=(A,B,C)
- (3) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=MOD1
- (4) INCLUDE-MODULES
LIBRARY=LIB,
ELEMENT=(D,E)
- (5) RENAME-SYMBOLS
SYMBOL-NAME=M,
SYMBOL-OCCURENCE=PARAMETERS
(OCCURENCE-NUMBER=ALL),
SYMBOL-TYPE=REFERENCES,
SCOPE=EXPLICIT(WITHIN-SUB-
LLM=X.MOD1),
NEW-NAME=K
- (6) END-SUB-LLM-STATEMENTS

aktuelles LLM
(Arbeitsbereich)Bedeutung

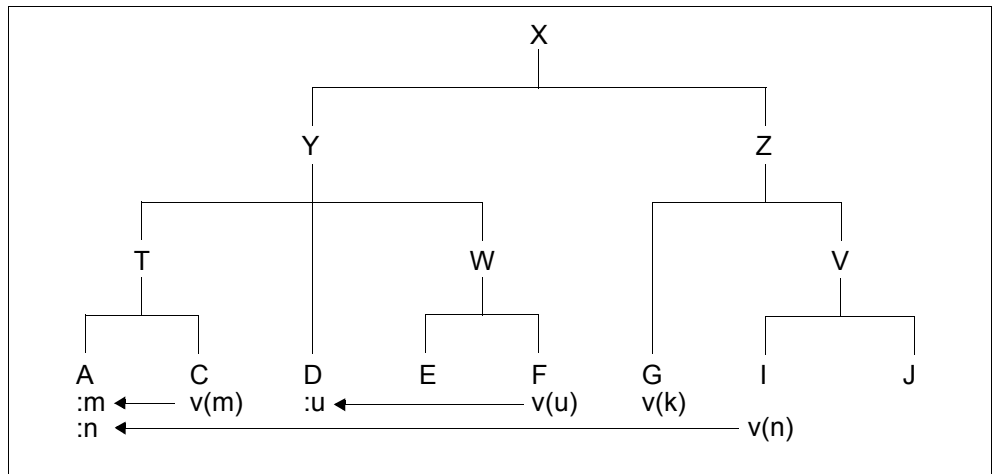
- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Die OMs A, B und C werden aus der Programmbibliothek LIB in das aktuelle LLM eingefügt. B hat einen ENTRY mit dem Namen „n“ und einen EXTRN mit dem Namen „m“.
- (3) Ein Sub-LLM mit dem Namen MOD1 wird an der Root begonnen.
- (4) In das aktuelle Sub-LLM werden aus der Programmbibliothek LIB die OMs D und E eingefügt. D hat einen ENTRY mit dem Namen „u“ und einen EXTRN mit dem Namen „m“.

- (5) Der Name „m“ von Referenzen soll in „k“ geändert werden (Operand SYMBOL-TYPE=REFERENCES). Nur der Name „m“ des EXTRN im Modul D wird in „k“ geändert, da mit dem Operanden SCOPE das Umbenennen auf das Sub-LLM MOD1 beschränkt wurde. Das Umbenennen gilt für jedes Auftreten des Namens (Operand OCCURRENCE-NUMBER=ALL).
- (6) Das aktuelle Sub-LLM wird abgeschlossen.

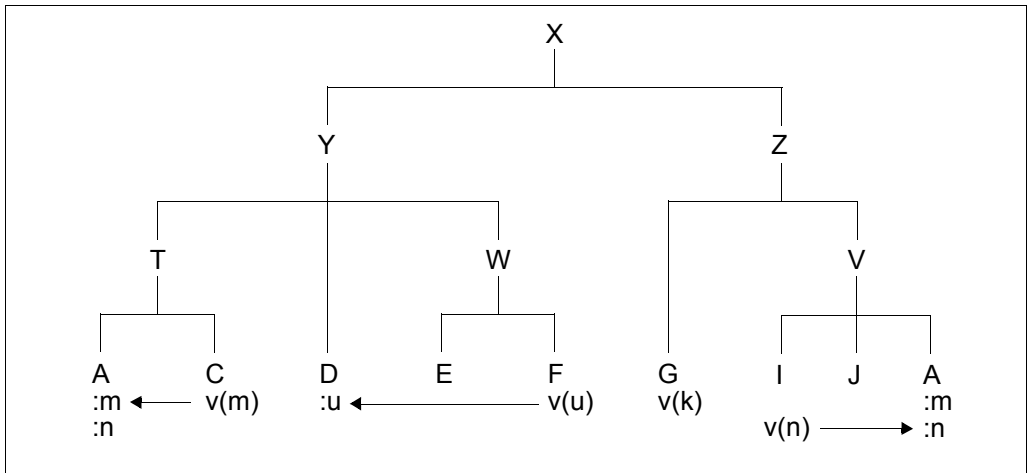
Beispiel 4

Dieses Beispiel zeigt, wie die Befriedigung von Externverweisen durch das Umbenennen von Programmdefinitionen beeinflusst wird.

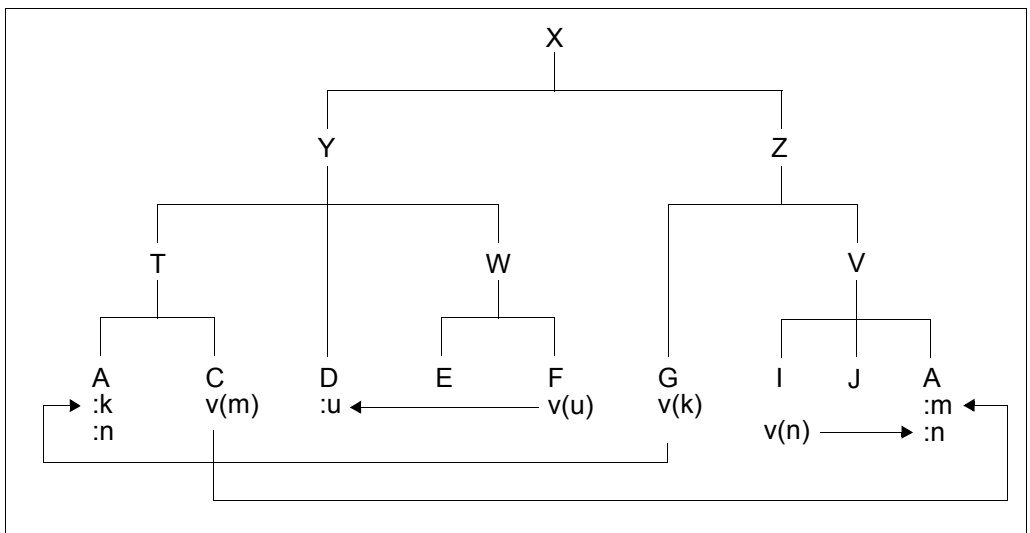
- (1) START-LLM-UPDATE LIBRARY=LIB,ELEMENT=LLM1



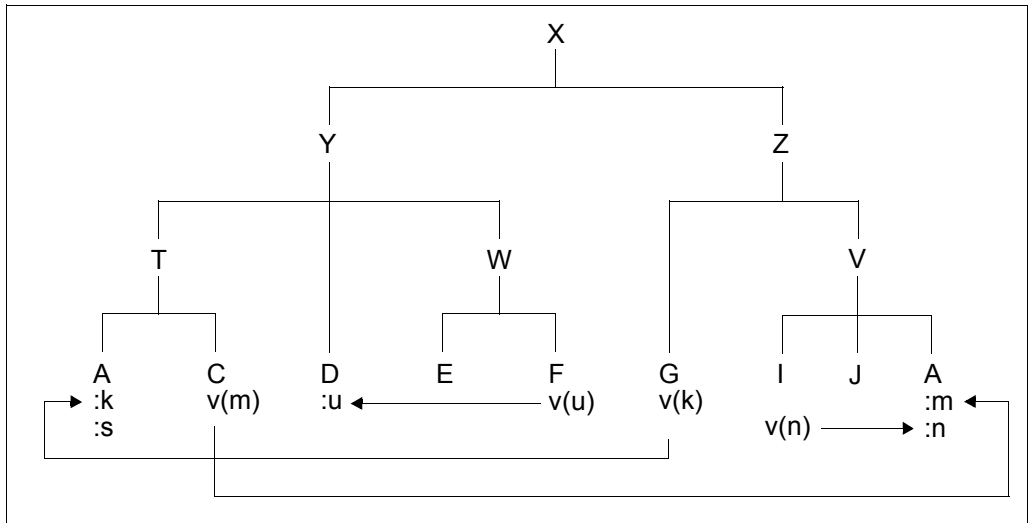
(2) INCLUDE-MODULES LIBRARY=LIB2,ELEMENT=A,PATH-NAME=X.Z.V



(3) RENAME-SYMBOLS SYMBOL-NAME=M,SYMBOL-TYPE=DEFINITIONS, SCOPE=EXPLICIT(WITHIN-SUB-LLM=X.Y.T.A),NEW-NAME=K



- (4) RENAME-SYMBOLS SYMBOL-NAME=N,SYMBOL-TYPE=DEFINITIONS,
SCOPE=EXPLICIT(WITHIN-SUB-LLM=X.Y.T.A),NEW-NAME=S



Bedeutung

- (1) Ein LLM, das als Element mit dem Namen LLM1 in der Programmbibliothek LIB gespeichert ist, soll geändert werden. Das LLM enthält folgende Programmdefinitionen und Referenzen:
 - den EXTRN $v(m)$ im OM C befriedigt von ENTRY „m“ im OM A,
 - den EXTRN $v(u)$ im OM F befriedigt von ENTRY „u“ im OM D,
 - den unbefriedigten EXTRN $v(k)$ im OM G,
 - den EXTRN $v(n)$ im OM I befriedigt von ENTRY „n“ im OM A.
- (2) Dasselbe Bindemodul A, das bereits im Sub-LLM T eingefügt ist, wird aus der Programmbibliothek LIB2 gelesen und in das Sub-LLM V eingefügt. Den EXTRN $v(n)$ im OM I befriedigt jetzt der ENTRY „n“ im zuletzt eingefügten OM A, da OM A im selben Sub-LLM wie OM I liegt.
- (3) Der ENTRY „m“ im OM A des Sub-LLM T wird umbenannt in „k“. Damit werden die Externverweise wie folgt befriedigt:
 - EXTRN $v(k)$ im OM G wird befriedigt von ENTRY „k“ im OM A des Sub-LLM T,
 - ENTRY $v(m)$ im OM C wird befriedigt von ENTRY „m“ im OM A des Sub-LLM V.
- (4) Der ENTRY „n“ im OM A des Sub-LLM T wird umbenannt in „s“. Dies hat keinen Einfluss auf die Befriedigung der Externverweise.

3.10.3 Ändern der Attribute von Symbolen

Die Attribute von Programmabschnitten (CSECTs) und COMMON-Bereichen im aktuellen LLM können mit der Anweisung `MODIFY-SYMBOL-ATTRIBUTES` geändert werden.

Folgende Attribute können geändert werden (siehe [Seite 18ff](#)):

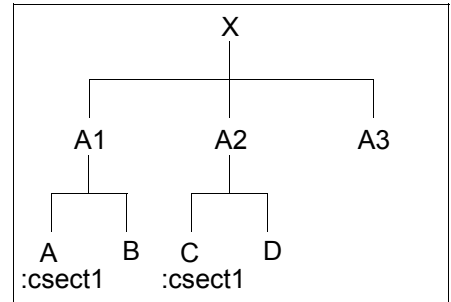
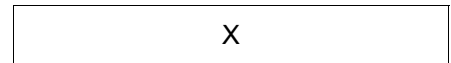
- Hauptspeicherresident (RESIDENT)
- gemeinsam benutzbar (PUBLIC)
- Lesezugriff (READ-ONLY)
- Ausrichtung (ALIGNMENT)
- Adressierungsmodus (AMODE)
- Residenzmodus (RMODE)

Beim Ändern der Attribute ist zu beachten, dass alle COMMON-Bereiche den gleichen Namen und alle CSECTs gleichen Namens, die diese COMMON-Bereiche mit Daten initialisieren, denselben Wert für das Attribut READ-ONLY haben müssen.

Der Geltungsbereich für das Ändern der Attribute kann auf bestimmte OMs und Sub-LLMs im aktuellen LLM eingeschränkt werden (Operand SCOPE).

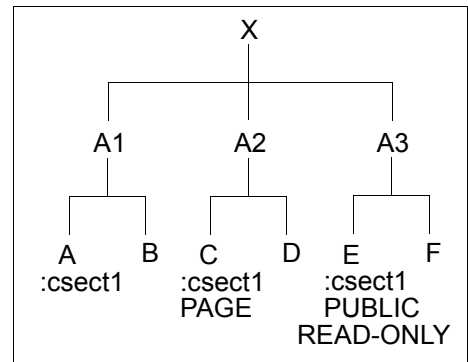
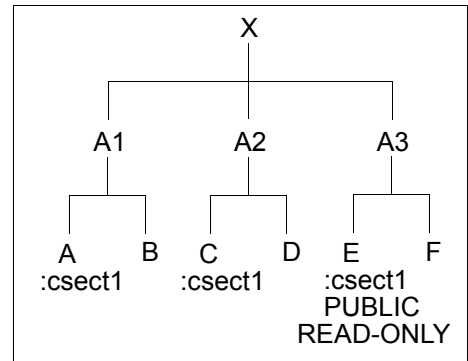
*Beispiel*Anweisungen

- (1) START-LLM-CREATION
INTERNAL-NAME=X
- (2) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=A1
- (3) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(A,B)
- (4) END-SUB-LLM-STATEMENTS
- (5) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=A2
- (6) INCLUDE-MODULES LIBRARY=LIB1,
ELEMENT=(C,D)
- (7) END-SUB-LLM-STATEMENTS
- (8) BEGIN-SUB-LLM-STATEMENTS
SUB-LLM-NAME=A3

aktuelles LLM
(Arbeitsbereich)

Anweisungen

- (9) INCLUDE-MODULES
LIBRARY=LIB1,
ELEMENT=(E,F)
- (10) MODIFY-SYMBOL-ATTRIBUTES
SYMBOL-NAME=CSECT1,
READ-ONLY=YES,
PUBLIC=YES
- (11) END-SUB-LLM-STATEMENTS
- (12) MODIFY-SYMBOL-ATTRIBUTES
SYMBOL-NAME=CSECT1,
SCOPE=EXPLICIT(WITHIN-SUB-
LLM=X.A2),
ALIGNMENT=PAGE

aktuelles LLM
(Arbeitsbereich)

Bedeutung

- (1) Ein LLM mit dem internen Namen X wird im Arbeitsbereich erzeugt.
- (2) Ein Sub-LLM mit dem Namen A1 wird begonnen.
- (3) In das aktuelle Sub-LLM A1 werden aus der Programmbibliothek LIB1 die Bindemodule A und B eingefügt. Das Bindemodul A enthält eine CSECT mit dem Namen CSECT1.
- (4) A1 wird abgeschlossen.
- (5) Ein Sub-LLM mit dem Namen A2 wird begonnen.
- (6) In A2 werden aus der Programmbibliothek LIB1 die Bindemodule C und D eingefügt. Das Bindemodul C enthält ein CSECT mit dem Namen CSECT1.
- (7) A2 wird abgeschlossen.
- (8) Ein Sub-LLM mit dem Namen A3 wird begonnen.

- (9) In A3 werden aus der Programmbibliothek LIB1 die Bindemodule E und F eingefügt. Das Bindemodul E enthält eine CSECT mit dem Namen CSECT1.
- (10) Die Attribute PUBLIC und READ-ONLY der CSECT1 werden geändert. Angesprochen wird das *aktuelle* Sub-LLM. Dies ist das Sub-LLM A3. Die Attribute der CSECTs „CSECT1“ in den Bindemodulen A und C bleiben unverändert.
- (11) A3 wird abgeschlossen.
- (12) Das Attribut ALIGNMENT der CSECT1 wird geändert. Angesprochen wird mit dem Pfadnamen X.A2 das Bindemodul C im Sub-LLM A2. Die Attribute der CSECTs „CSECT1“ in den Bindemodulen A und E bleiben unverändert.

3.10.4 Ändern der Maskierung von Symbolen

Der Benutzer hat die Möglichkeit, Programmabschnitte (CSECTs) und Einsprungsstellen (ENTRYS) im aktuellen LLM zu maskieren. Ein maskiertes Symbol ist zwar im Externadressbuch (ESV) des LLM gespeichert, es wird aber mit einem Kennzeichen (einer „Maske“) versehen. Dadurch ist es für die Autolink-Funktion nicht mehr sichtbar und kann auch nicht mehr zur Befriedigung von Externverweisen herangezogen werden.

In welchem Umfang die Symbole sichtbar bleiben oder maskiert werden, legt der Benutzer mit der Anweisung MODIFY-SYMBOL-VISIBILITY fest. Bei maskierten Symbolen kann ausgewählt werden, ob befriedigte Externverweise zu den angegebenen Symbolen befriedigt bleiben oder aufgelöst werden sollen (Operand KEEP-RESOLUTION).

Der Geltungsbereich für das Ändern der Maskierung von Symbolen kann auf bestimmte OMs und Sub-LLMs im aktuellen LLM eingeschränkt werden (Operand SCOPE).

Die Behandlung von eventuell auftretenden Namenskonflikten kann mit dem Operanden NAME-COLLISION gesteuert werden.

Beispiel

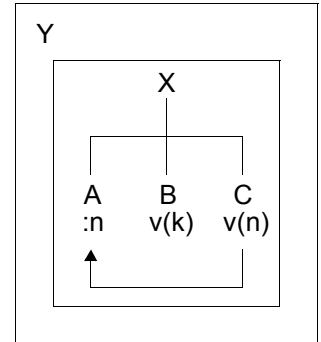
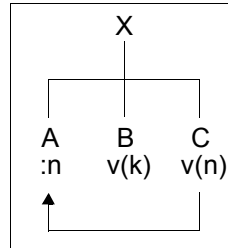
Dieses Beispiel zeigt, wie durch das Maskieren und Umbenennen von Symbolen das Befriedigen von Externverweisen beeinflusst wird.

Anweisungen

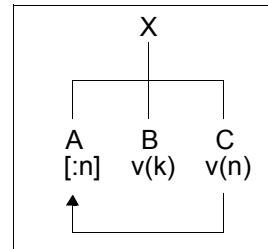
aktuelles LLM
(Arbeitsbereich)

Programmbibliothek
(LIB1)

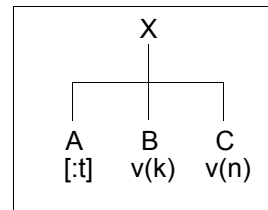
- (1) START-LLM-UPDATE
LIBRARY=LIB1,
ELEMENT=Y



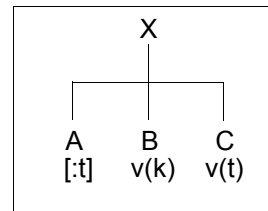
- (2) MODIFY-SYMBOL-VISIBILITY
SYMBOL-NAME=N,
VISIBLE=NO (KEEP-
RESOLUTION=YES)



- (3) RENAME-SYMBOLS
SYMBOL-NAME=N,
SYMBOL-TYPE=DEFINITIONS,
NEW-NAME=T



- (4) RENAME-SYMBOLS
SYMBOL-NAME=N,
SYMBOL-TYPE=REFERENCES,
NEW-NAME=T



Bedeutung

- (1) Ein LLM X, das als Element mit dem Elementnamen Y in der Programmbibliothek LIB1 gespeichert ist, soll geändert werden. Das LLM X enthält den EXTRN v(n), der von dem ENTRY n befriedigt wird, und den unbefriedigten EXTRN v(k).
- (2) Der ENTRY n im LLM X wird maskiert (Maskierung durch eckige Klammern dargestellt). Der befriedigte EXTRN v(n) bleibt befriedigt (Operand KEEP-RESOLUTION=YES).
- (3) Der Name „n“ des maskierten ENTRY wird in „t“ geändert. Damit wird der befriedigte EXTRN (n) aufgelöst.
- (4) Die Namen „n“ der EXTRNs werden in „t“ geändert. Der EXTRN v(t) kann nicht vom ENTRY t befriedigt werden, da der ENTRY t maskiert ist.

3.10.5 Ändern des Typs von Symbolen

Der Benutzer hat mit der Anweisung MODIFY-SYMBOL-TYPE die Möglichkeit, den Typ von Symbolen im aktuellen LLM zu ändern. Unter „Symbolen“ sind hier jedoch nur Referenzen zu verstehen. Den Typ von Programmdefinitionen (CSECTs, ENTRYs) kann der Benutzer nicht ändern.

Externverweise (EXTRN),V-Konstanten (VCON) und bedingte Externverweise (WXTRN) können fast ohne Einschränkungen ineinander umgewandelt werden. Der Benutzer hat z.B. die Möglichkeit, EXTRNs oder VCONs in WXTRNs umzuwandeln. Damit verhindert er, dass diese Externverweise durch die Autolink-Funktion befriedigt werden. Es ist jedoch nicht möglich, nicht referenzierte EXTRNs und WXTRNs in VCONs umzuwandeln.

Ein weiterer Anwendungsfall für die Anweisung MODIFY-SYMBOL-TYPE ergibt sich aus der Nutzung von OCM (Overlay Control Module) bei benutzerdefinierten Slices (siehe Anweisung START-LLM-CREATION). Da in den OCMs nur VCON als Referenzen verwendet werden können, müssen alle Referenzen von Symboltyp VCON sein.

Der Geltungsbereich für das Ändern des Typs von Symbolen kann auf bestimmte Binde- module und Sub-LLMs im aktuellen LLM eingeschränkt werden (Operand SCOPE).

Auf bereits befriedigte Externverweise hat die Anweisung MODIFY-SYMBOL-TYPE keine Auswirkungen.

3.11 Mischen von Modulen

Ein Sub-LLM oder auch ein ganzes LLM kann „gemischt“ werden, d.h. sämtliche CSECTs des Sub-LLMs werden zusammengefasst und nach dem Mischvorgang besteht es nur noch aus einer einzigen CSECT. Externverweise zwischen den gemischten Modulen bleiben befriedigt und werden aus dem Externadressbuch gelöscht. Mit dem Operanden ENTRY-LIST kann der Benutzer angeben, welche CSECTs und ENTRYs im Externadressbuch bleiben sollen und somit weiterhin zur Befriedigung von Externverweisen genutzt werden können. Jede CSECT, die eingemischt wurde und noch im Externadressbuch steht, wird in einen ENTRY umgewandelt.

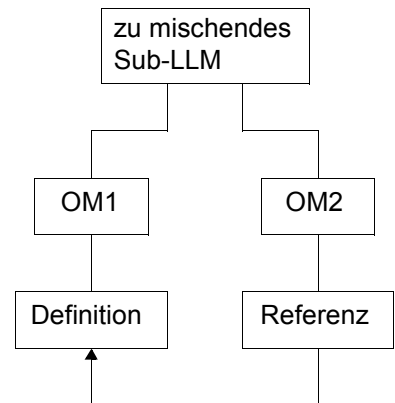
Im Folgenden werden drei mögliche Situationen dargestellt, die beim Mischen eines Sub-LLMs auftreten können.

Fall 1

Das zu mischende Sub-LLM enthält nur Verweise auf Symbole innerhalb desselben Sub-LLM.

Folgerung:

Der Externverweis ist endgültig befriedigt und wird aus dem Externadressbuch gestrichen. Gibt der Benutzer die Definition im Operanden ENTRY-LIST an, so bleibt sie im Externadressbuch und wird in einen ENTRY umgewandelt, falls es eine CSECT war.



OM: Object Module (Bindemodul)

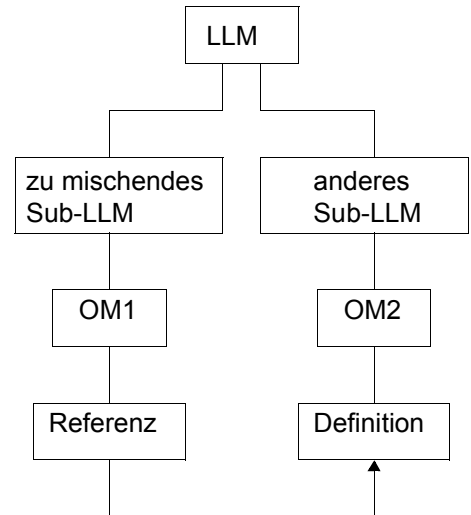
Fall 2

Das zu mischende Sub-LLM enthält Verweise auf Symbole in einem oder mehreren anderen Sub-LLMs.

Folgerung:

An der Befriedigung der Externverweise ändert sich nichts.

Das Externadressbuch ist davon ebenfalls nicht betroffen.



OM: Object Module (Bindemodul)

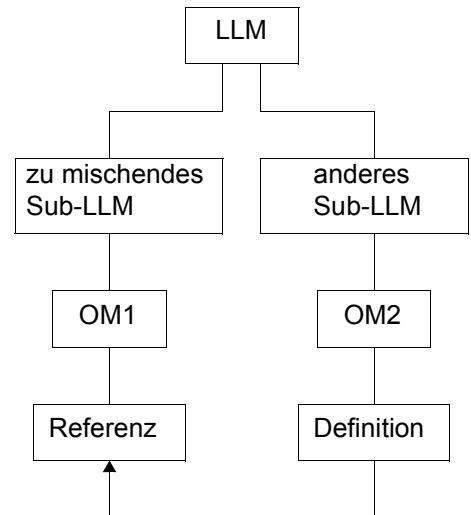
Fall 3

Das zu mischende Sub-LLM enthält Definitionen, auf die in anderen Sub-LLMs verwiesen wird.

Folgerung:

Gibt der Benutzer diese Definitionen im Operanden ENTRY-LIST an, so bleiben sie im Externadressbuch und werden in ENTRYs umgewandelt, falls es CSECTs waren. An der Befriedigung des Externverweises in OM2 ändert sich nichts.

Gibt der Benutzer den Operanden ENTRY-LIST jedoch nicht an, so wird die Definition aus dem Externadressbuch gestrichen. Der Externverweis in OM2 ist deshalb unbefriedigt und BINDER versucht, diesen Externverweis mit den im Externadressbuch vorhandenen Definitionen zu befriedigen.



OM: Object Module (Bindemodul)

Nach dem Mischen der OMs eines Sub-LLM bleibt folgende logische Struktur:

- ein Sub-LLM-Knoten mit dem Namen des gemischten Sub-LLM,
- ein Großmodul (als „Blatt“ in der logischen Struktur, siehe [Seite 16ff](#)), das denselben Namen wie das gemischte Sub-LLM hat.

Alle Knoten, die durch das Mischen beseitigt wurden, sind während des BINDER-Laufes in der BINDER-Liste noch sichtbar. Sie werden jedoch bei SAVE-LLM nicht mit abgespeichert.

Wenn die zu mischenden CSECTs in verschiedenen Slices enthalten sind, ist Folgendes zu beachten:

- LLMs mit benutzerdefinierten Slices dürfen nicht gemischt werden.
- Bei nach Attributen gebildeten Slices wird die neue CSECT in die Slice eingefügt, deren Attribute denjenigen entsprechen, die mit dem Operanden NEW-CSECT-ATTRIBUTES angegeben wurden.

Ist eine Definition, die in ENTRY-LIST angegeben wurde, mehrfach im zu mischenden Sub-LLM vorhanden, so wird nur eine Definition dieses Namens im Externadressbuch bleiben und es gilt:

- CSECTs haben Vorrang vor ENTRYs,
- die erste im Externadressbuch auftretende Definition dieses Namens bleibt im Externadressbuch.

Für den Inhalt der neuen CSECT wird keine Test- und Diagnoseinformation (LSD) erzeugt. Das gemischte Modul wird nicht als Laufzeitmodul betrachtet, demzufolge wird an der Sichtbarkeit der Symbole auch nichts verändert. Beibehalten werden auch gemeinsame Speicherbereiche (COMMONs), externe Pseudoabschnitte (XDSEC-D) und Pseudoregister in den gemischten Modulen. Die Initialisierung der COMMON-Bereiche wird ebenfalls aktualisiert und das Externadressbuch auf den neuesten Stand gebracht.

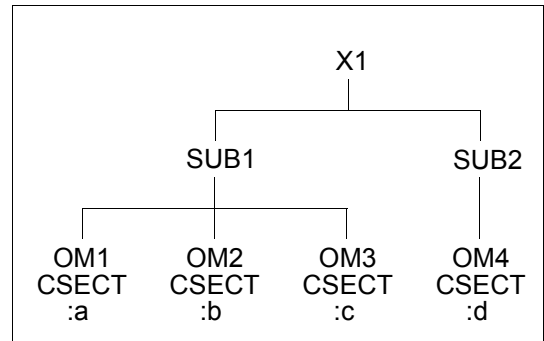
Beispiel 1

Mischen des gesamten LLMs. Das Ergebnis ist ein LLM, das nur ein einziges Modul (Großmodul) enthält.

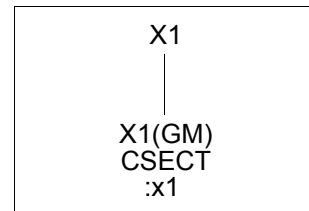
Anweisungen

- (1) START-LLM-UPDATE
LIBRARY=LIB1,
ELEMENT=M

aktuelles LLM
(Arbeitsbereich)



- (2) MERGE-MODULES
NAME=X1,
PATH-NAME=*NONE



GM: Großmodul

Bedeutung

- (1) Ein LLM, das als Element mit dem Namen M in der Programmbibliothek LIB1 gespeichert ist, soll gemischt werden. Es wird dazu in den Arbeitsbereich eingelesen. Das LLM hat den internen Namen X1.
- (2) Das gesamte LLM mit dem internen Namen X1 wird gemischt, da für PATH-NAME=*NONE festgelegt wurde. Die neue CSECT erhält denselben Namen wie das LLM, weil für NEW-CSECT-NAME der Standardwert *NAME gilt.

Beispiel 2

Mischen eines Sub-LLM, wobei die Befriedigung der Externverweise beeinflusst wird.

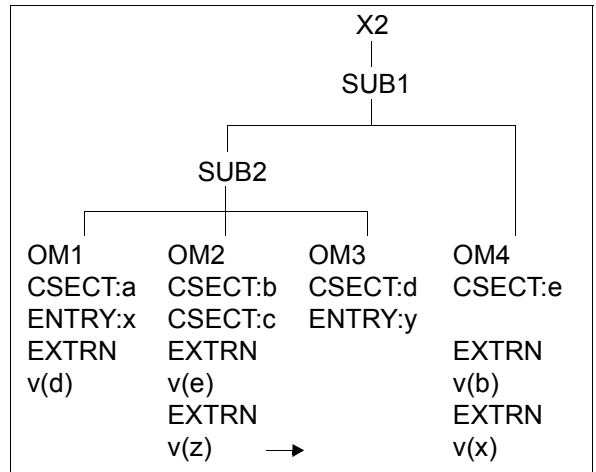
Ausgangssituation:

Im LLM X2 sind mit Ausnahme von EXTRN v(z) alle Externverweise befriedigt durch CSECTs oder ENTRYs im LLM X2.

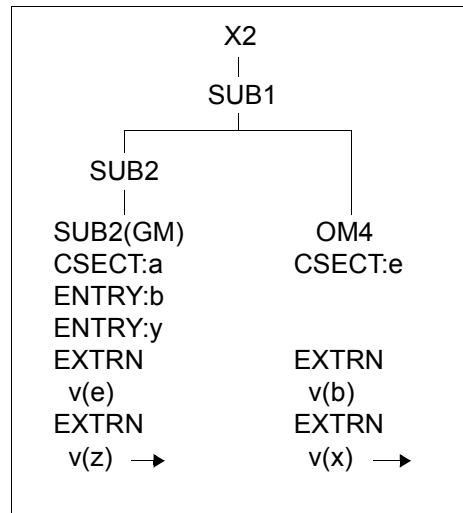
Anweisungen

- (1) START-LLM-UPDATE
LIBRARY=LIB1,
ELEMENT=N

aktuelles LLM
(Arbeitsbereich)



- (2) MERGE-MODULES
NAME=SUB2,
PATH-NAME=X2.SUB1,
NEW-CSECT-NAME=*STD,
ENTRY-LIST=(b,y)



GM: Großmodul

Bedeutung

- (1) Ein Sub-LLM des LLM mit dem internen Namen X2 soll gemischt werden. Dazu wird das LLM X2, das als Bibliothekselement N in der Programmbibliothek LIB1 gespeichert ist, in den Arbeitsbereich eingelesen.
- (2) Das Sub-LLM SUB2, das über den Pfad X2.SUB1 zu erreichen ist, wird gemischt. Die neue CSECT erhält den Namen der ersten CSECT im Sub-LLM SUB2. Die Programmdefinitionen b und y aus dem Sub-LLM SUB2 bleiben im Externadressbuch, wobei b in einen ENTRY umgewandelt wird. Alle anderen Programmdefinitionen aus SUB2 (c,d,x) werden aus dem Externadressbuch gelöscht.

Nach dem Mischen von SUB2 ist:

- EXTRN v(d) endgültig befriedigt und erscheint nicht mehr im Externadressbuch,
- EXTRN v(z) immer noch unbefriedigt,
- EXTRN v(x) nun auch unbefriedigt, weil er sich außerhalb des Sub-LLM SUB2 befindet und der ENTRY x in SUB2 aus dem Externadressbuch gelöscht wurde.

3.12 Festlegen von Standardwerten

Für einen BINDER-Lauf oder für einen Edit-Lauf (siehe [Seite 138](#)) kann sich der Benutzer für einige Operanden die Standardwerte selbst festlegen.

Folgende Standardwerte können festgelegt werden:

- **MAP-Standardwerte:**
Sie werden mit der Anweisung `MODIFY-MAP-DEFAULTS` festgelegt und gelten für die Dauer eines BINDER-Laufs.
- **CURRENT-Standardwerte:**
 - Die aktuelle Eingabebibliothek (`CURRENT-INPUT-LIB`) zum Einfügen und Ersetzen von Modulen wird mit den Anweisungen `INCLUDE-MODULES` und `REPLACE-MODULES` eingestellt.
 - Die aktuelle Ein-/Ausgabebibliothek für LLMs, der aktuelle Elementname und die aktuelle Elementversion werden bei `START-LLM-UPDATE` und `SAVE-LLM` festgelegt.
 - Das aktuelle Sub-LLM wird mit der Anweisung `BEGIN-SUB-LLM-STATEMENTS` eingestellt. Mit der Anweisung `END-SUB-LLM-STATEMENTS` wird das ihm übergeordnete LLM zum aktuellen Sub-LLM.
- **INCLUSION-DEFAULTS:**
Sie werden in den Anweisungen `START-LLM-CREATION`, `START-LLM-UPDATE` und `MODIFY-LLM-ATTRIBUTES` für die Dauer eines Edit-Laufes eingestellt und können temporär für die Dauer der Anweisung `INCLUDE-MODULES` bzw. `REPLACE-MODULES` geändert werden.
- **SAVE-Standardwerte:**
Die Werte werden in der Anweisung `SAVE-LLM ...=LAST-SAVE...` eingestellt und sind für die Dauer eines Edit-Laufes verfügbar.
- **globale Standardwerte (STD-DEFAULTS):**
Globale Standardwerte, die das Format des LLM und die Behandlung von Namenskonflikten festlegen, sind mit der Anweisung `MODIFY-STD-DEFAULTS` einstellbar. Sie gelten für einen BINDER-Lauf.

3.13 Anzeigefunktionen

3.13.1 Anzeigen von Standardwerten

Die Anweisung SHOW-DEFAULTS ermöglicht es dem Benutzer, sich Standardwerte anzeigen zu lassen (siehe [Seite 126](#)). Er kann dabei wählen, ob er sich

- die globalen Standardwerte (STD-DEFAULTS),
- die CURRENT-Standardwerte,
- die INCLUSION-DEFAULTS,
- die Werte für das letzte Abspeichern von LLMs (LAST-SAVE) oder
- die Standardwerte für die BINDER-Listen (MAP-DEFAULTS)

anzeigen lassen will.

Die Operanden CURRENT-DEFAULTS, INCLUSION-DEFAULTS und LAST-SAVE sind erst nach der Abarbeitung der Anweisungen START-LLM-CREATION oder START-LLM-UPDATE sinnvoll belegt. Die Werte werden auf der Standardausgabe SYSOUT ausgegeben.

Beispiel

Das folgende Beispiel zeigt die Ausgabe der Anweisung SHOW-DEFAULTS.

```

/start-binder
% BND0500 BINDER VERSION 'V02.7A00' STARTED
//start-llm-creation internal-name=complex1
//show-defaults
STD-DEFAULTS:
  OVERWRITE           =YES
  FOR-BS2000-VERSIONS =FROM-CURRENT
  CONNECTION-MODE     =OSD-DEFAULT
  REQUIRED-COMPRESSION=NO
  NAME-COLLISION:
    INCLUSION         =IGNORED
    SAVE              =IGNORED
    SYMBOL-PROCESSING=IGNORED
CURRENT-DEFAULTS:
  CURRENT-SUB-LLM     =COMPLEX1
LIBRARY:
  CURRENT             =
  CURRENT-INPUT-LIB=
ELEMENT:
  CURRENT             =

```

```
VERSION:
  CURRENT-VERSION =
INCLUSION-DEFAULTS:
  LOGICAL-STRUCTURE=WHOLE-LLM
  TEST-SUPPORT      =NO
LAST-SAVE:
  OVERWRITE          =STD
  FOR-BS2000-VERSIONS =STD
  REQUIRED-COMPRESSION=STD
  NAME-COLLISION     =STD
  SYMBOL-Dictionary  =YES
  RELOCATION-DATA     =YES
  LOGICAL-STRUCTURE  =WHOLE-LLM
  TEST-SUPPORT       =YES
  LOAD-ADDRESS       =UNDEFINED
  ENTRY-POINT        =*STD
  MAP                 =YES
MAP-DEFAULTS:
  MAP-NAME           =*STD
  COMMENT            =NONE
  HELP-INFORMATION   =YES
  GLOBAL-INFORMATION =YES
  LOGICAL-STRUCTURE  =YES
  RESOLUTION-SCOPE   =YES
  PHYSICAL-STRUCTURE =YES
  PROGRAM-MAP        =PARAMETERS
  DEFINITIONS        =ALL
  INVERTED-XREF-LIST =NONE
  REFERENCES         =ALL
  UNRESOLVED-LIST    =SORTED (WXTRN=YES,NOREF=NO)
  SORTED-PROGRAM-MAP =NO
  PSEUDO-REGISTER    =NO
  UNUSED-MODULE-LIST =NO
  DUPLICATED-LIST    =NO
  MERGED-MODULES     =YES
  INPUT-INFORMATION  =YES
  STATEMENT-LIST     =NO
  OUTPUT             =*SYSLST
  SYSLST-NUMBER      =STD
  LINES-PER-PAGE     =64
  LINE-SIZE          =136
//end
% BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
```


3.13.2 Anzeigen von Symbolinformationen

Mit der Anweisung SHOW-SYMBOL-INFORMATION kann sich der Benutzer Informationen über Symbole anzeigen lassen. Die Menge der angezeigten Informationen lässt sich mit dem Operanden INFORMATION einschränken, je nachdem, was für den Benutzer im Moment interessant ist. Er kann sich z.B. die Position von Symbolen in der logischen Struktur des LLM, ihre Attribute oder ihre relative Adresse (Name der Slice und relative Adresse in der Slice) einzeln oder alle zusammen anschauen. Der Benutzer kann außerdem wählen, ob er sich

- alle sichtbaren Definitionen,
- die COMMON-Bereiche und ihre Initialisierung,
- die befriedigten Externverweise,
- eine Liste von mehrfach verwendeten Symbolnamen,
- eine Liste der unbefriedigten Externverweise

anzeigen lassen will. Die Art der Darstellung entspricht im Allgemeinen der Darstellung in den BINDER-Listen (siehe [Seite 141ff](#)). Die Informationen werden auf der Standardausgabe SYSOUT ausgegeben.

Beispiel

```

/start-binder
% BND0500 BINDER VERSION 'V02.7A00' STARTED
//start-llm-update library=bnd.llmlib,element=complex1 _____ (1)
//show-symbol-information information=all _____ (2)
AUTOA SD _____ (3)
  @=000000A8 L=00000040 (AA.....)
  IN MODULE : AUTOA
  IN SLICE:PUU-ROU-RTU-RMU + 000000A8
AUTOA SD
  @=00000068 L=00000040 (AA.....)
  IN MODULE : AUTOA
  IN SLICE:PUU-ROU-RTU-RMU + 00000068
AUTOA SD
  @=00000000 L=00000040 (AA.....)
  IN MODULE : AUTOA
  IN SLICE:PUU-ROU-RTU-RMU + 00000000
AUTO2 CM
  NOT PROMOTED L=00000050 (AA.....)
  IN MODULE : AUTO2
AUTO2 CM
  NOT PROMOTED L=00000050 (AA.....)
  IN MODULE : AUTO2
AUTO2 SD
  @=000000E8 L=00000018 (AA.....)
  IN MODULE : AUTO2
  IN SLICE:PUU-ROU-RTU-RMU + 000000E8
AUTO2 SD
  @=00000040 L=00000018 (AA.....)
  IN MODULE : AUTO2
  IN SLICE:PUU-ROU-RTU-RMU + 00000040
AUTO22 SD
  @=00000118 L=00000006 (AA.....)
  IN MODULE : AUTO22
  IN SLICE:PUU-ROU-RTU-RMU + 00000118
AUTO22 CM
  @=00000118 L=00000050 (AA.....)
  IN MODULE : AUTO23
AUTO2 SD
  @=00000100 L=0000000C (AA.....)
  IN MODULE : AUTO2
  IN SLICE:PUU-ROU-RTU-RMU + 00000100
AUTO2 SD
  @=00000058 L=0000000C (AA.....)
  IN MODULE : AUTO2
  IN SLICE:PUU-ROU-RTU-RMU + 00000058

```

```
AUT021                                SD
  @=00000110      L=00000006      (AA.....)
  IN MODULE : AUT021
  IN SLICE:PUU-ROU-RTU-RMU + 00000110
AUT023                                SD
  @=00000168      L=00000006      (AA.....)
  IN MODULE : AUT023
  IN SLICE:PUU-ROU-RTU-RMU + 00000168
//end
% BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
```

- (1) Das LLM COMPLEX1 wird in den Arbeitsbereich des BINDER eingelesen.
- (2) Folgende Informationen werden für alle Symbole im LLM ausgegeben:
 - Symbolname
 - Art des Symbols (CSECT, COMMON,...)
 - Ladeadresse bei CSECTs und ENTRYs bzw. bei COMMON-Bereichen die Ladeadresse der CSECT, durch die der COMMON-Bereich initialisiert wurde
 - Länge der Textinformation bei CSECTs
 - Attribute
 - Name des Moduls, das das Symbol enthält
 - Name der Slice, die das Symbol enthält und relative Adresse des Symbols in der Slice
- (3) Das erste Symbol mit dem Namen AUTOA ist eine CSECT und hat die Ladeadresse 000000A8 (Länge der Textinformation der CSECT: 00000040). Die CSECT hat folgende Attribute: AMODE=ANY, RMODE=ANY. Andere Attribute sind nicht spezifiziert. Das Symbol befindet sich im Modul AUTOA und in der Slice mit dem Namen PUU-ROU-RTU-RMU und hat dort die relative Adresse 000000A8.

3.13.3 Anzeigen und Prüfen von Bibliothekselementen

Das Anzeigen und Prüfen von LLMs oder Bindemodulen in bestimmten Bibliotheken ist mit der Anweisung SHOW-LIBRARY-ELEMENTS möglich. Der Benutzer kann sich wie bei der Listen-Ausgabefunktion SHOW-MAP selbst aussuchen, auf welchem Ausgabemedium die Informationen erscheinen sollen. Um Namenskonflikte zu vermeiden, kann sich der Benutzer während des BINDER-Laufs z.B. eine Liste von gefährdeten Symbolen (DUPLICATE SYMBOLS) generieren lassen. Die erzeugten Listen sind (mit Ausnahme der auf SYSLST ausgegebenen Listen) standardmäßig ISAM-Dateien mit ISAM-Schlüsseln der Länge 8. Die ISAM-Schlüssel können zur Auswertung der Listen verwendet werden. Sie finden die Beschreibung der ISAM-Schlüssel im Anhang ([Seite 369f](#)).

Beispiel

Im folgenden Beispiel lässt sich der Benutzer zuerst Informationen über die LLMs AUTOLINKL und AUTOLINKR aus der Programmbibliothek BND.LLMLIB auf SYSLST ausgeben. Danach fordert er für diese beiden LLMs eine Liste der Symbole mit gleichen Namen (DUPLICATE SYMBOLS) an und lässt sich diese Liste ebenfalls auf SYSLST ausgeben.

```

/start-binder
% BND0500 BINDER VERSION 'V02.7A00' STARTED
//show-library-elements library=bnd.llmlib, -
//                               element=(autolinkl,autolinkr) _____ (1)
//show-library-elements library=bnd.llmlib,-
//                               element=(autolinkl,autolinkr),-
//                               select=name-collision _____ (2)
%//end
% BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'

```

(1) Informationen über die Bibliothekselemente AUTOLINKL und AUTOLINKR:

```

BINDER V02.7A *LIBRARY CONTENT* DATE=2016-02-03 10:43:51 PAGE 1
LIBRARY LINKNAME TYPE
TYP ELEMENT VERSION
TY SYMBOL TY SYMBOL TY SYMBOL TY SYMBOL TY SYMBOL
-----
:CTID:$USERID.BND.LLMLIB PLAM
(L) AUTOLINKL @
SD AUTOA SD AUTOC SD AUTO2 SD AUTOA SD AUTOA
SD AUTOC SD AUTO2
(L) AUTOLINKR @
SD AUTO21 SD AUTO22 SD AUTO23

```

--- END OF SECTION ---

(2) Liste der Symbole mit gleichen Namen für die Bibliothekselemente AUTOLINKL und AUTOLINKR:

BINDER V02.7A *DUPLICATE SYMBOLS* DATE=2016-02-03 10:43:51 PAGE 1
 SYMBOL

LIBRARY LINKNAME TYPE
 TYP ELEMENT VERSION
 SYMBOL TYPE

```

-----
AUTOA
:CTID:$USERID.BND.LLMLIB          PLAM
(L) AUTOLINKL @
  CSECT
  CSECT
  CSECT
AUTOC
:CTID:$USERID.BND.LLMLIB          PLAM
(L) AUTOLINKL @
  CSECT
  CSECT
AUTO2
:CTID:$USERID.BND.LLMLIB          PLAM
(L) AUTOLINKL @
  CSECT
  CSECT
  
```

--- END OF SECTION ---

3.14 Steuern der Protokollierung

Listen mit Informationen über das aktuelle LLM werden ausgegeben:

- mit der Anweisung SHOW-MAP,
- mit der Anweisung SAVE-LLM (Operand MAP) beim Speichern eines LLM.

Dabei kann als Ausgabeziel gewählt werden:

- die Systemdatei SYSLST,
- eine Datei, die automatisch mit einem impliziten Kommando SHOW-FILE (siehe Handbuch „Kommandos“ [5]) auf SYSOUT ausgegeben wird. Diese Datei wird unter dem Standarddateinamen angelegt und nach Beenden der Ausgabe sofort wieder gelöscht, wenn der Benutzer das Löschen nicht verbietet.
- eine Datei, deren Name durch den Benutzer festgelegt wird.
- eine Datei, die durch den Standard-Dateikettungsnamen oder einen angegebenen Dateikettungsnamen festgelegt wurde.
- ein benutzereigenes Unterprogramm, das die gelieferten Informationen auswertet. Die Informationen werden in eine ISAM-Datei ausgegeben, auf die das Unterprogramm zugreifen muss.

Diese Ausgabeziele stehen auch in der Anweisung SHOW-LIBRARY-ELEMENTS zur Verfügung.

Welche Informationen ausgegeben werden, bestimmen die Operanden in der Anweisung SHOW-MAP. Standardeinstellungen können mit der Anweisung MODIFY-MAP-DEFAULTS geändert werden. Die verschiedenen Listen sind auf [Seite 141ff](#) beschrieben.

Mit den Anweisungen START-STATEMENT-RECORDING und STOP-STATEMENT-RECORDING wird die Protokollierung von BINDER-Anweisungen ein- bzw. ausgeschaltet. Die so protokollierten Anweisungen werden mit der Anweisung SHOW-MAP in eine Liste ausgegeben, wenn der Operand STATEMENT-LIST auf YES gesetzt ist.

3.15 Steuern der Fehlerbehandlung

3.15.1 Fehlerklassen

Für die Fehlerbehandlung benutzt der BINDER eine Tabelle der **Fehlerklassen**. Sie beschreiben alle möglichen Maßnahmen, die die Fehlerbehandlungsroutinen des BINDER treffen sollen, wenn im BINDER-Lauf Fehler aufgetreten sind.

Folgende Fehlerklassen sind möglich:

Fehlerklasse	Bedeutung
INFORMATION	Kein Fehler entdeckt. Eine Informationsmeldung wird ausgegeben.
WARNING	Möglicherweise kann das LLM nicht geladen werden. Eine Warnungsmeldung wird ausgegeben.
UNRESOLVED EXTERNS	Im LLM gibt es unbefriedigte Externverweise. Laden ist möglich.
SYNTAX ERROR	Ein Fehler wurde bei der Syntaxprüfung einer Anweisung erkannt.
RECOVERABLE ERROR	Ein Fehler wurde erkannt. Bei einigen Fehlerarten wird die Verarbeitung automatisch oder nach Eingabe einer Quittierung fortgesetzt.
FATAL ERROR	Ein Fehler wurde erkannt. Die Verarbeitung des LLM wird ohne Ausgabe eines Dump abgebrochen.
INTERNAL ERROR	Interner Fehler. Die Verarbeitung des LLM wird abgebrochen. Der Benutzer kann fordern, dass ein Dump ausgegeben wird.

Jede Fehlerklasse hat einen bestimmten Wichtigkeitsgrad. Die Fehlerklasse INFORMATION hat den niedrigsten, die Fehlerklasse INTERNAL ERROR den höchsten Wichtigkeitsgrad. Die Fehlerklassen, die der BINDER berücksichtigen soll, legt der Benutzer mit der Anweisung MODIFY-ERROR-PROCESSING fest.

Der Benutzer kann für die Fehlerklassen festlegen, ob bei Auftreten von Fehlern bestimmter Fehlerklassen

- der BINDER-Lauf beendet werden soll,
- Fehlermeldungen ausgegeben werden sollen,
- Benutzer- und/oder Auftragsschalter gesetzt werden sollen.

3.15.2 Meldungsbehandlung

Mit dem Operanden MESSAGE-CONTROL in der Anweisung MODIFY-ERROR-PROCESSING kann man festlegen, für welche *Fehlerklassen* Meldungen ausgegeben werden sollen. Dieser Operand bestimmt die *niedrigste* Fehlerklasse, ab der Meldungen ausgegeben werden. Die folgende Tabelle zeigt, wie der Operand MESSAGE-CONTROL die Meldungsausgabe steuert.

Meldungen der Fehlerklassen	MESSAGE-CONTROL		
	INFORMATION	WARNING	ERROR
INFORMATION	ja	nein	nein
WARNING	ja	ja	nein
UNRESOLVED EXTERNS	ja	ja	nein
SYNTAX ERROR	ja	ja	ja
RECOVERABLE ERROR	ja	ja	ja
FATAL ERROR	ja	ja	ja
INTERNAL ERROR	ja	ja	ja

4 BINDER-Ein-/Ausgabe

Einen Überblick über die möglichen Ein- und Ausgabedateien des BINDER gibt [Bild 7](#):

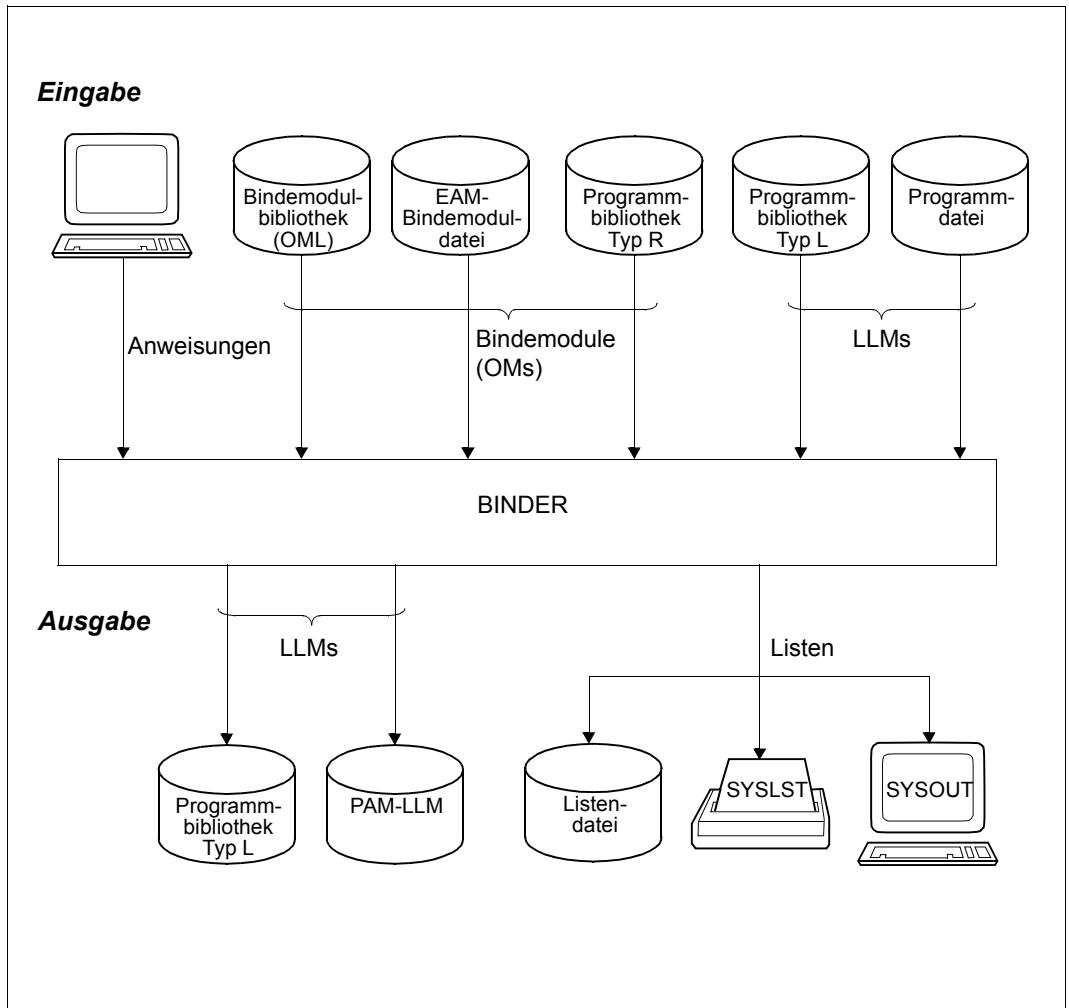


Bild 7: Ein- und Ausgabedateien des BINDER

4.1 Eingaben für den BINDER

Die Eingaben für den BINDER sind Anweisungen und Module.

Anweisungen

Anweisungen erwartet der BINDER aus der Systemdatei SYSDTA. Die Anweisungen können im Dialog oder aus Prozedurdateien eingegeben werden. Für die Eingabe im Dialog steht dem Benutzer die Dialogschnittstelle **SDF** zur Verfügung (siehe [Seite 192ff](#)). Jede Anweisung wird nach der Eingabe *sofort* verarbeitet.

In den BINDER-Anweisungen können Operandenwerte als globale Werte angegeben werden. Diese Werte können dann innerhalb eines festgelegten Geltungsbereichs in nachfolgenden Anweisungen als Standardwerte verwendet werden. Folgende Geltungsbereiche sind festgelegt:

- Geltungsbereich für *eine* Anweisung
Dieser Geltungsbereich ist lokal. Ein Operandenwert gilt nur für die zugehörige Anweisung.
- Geltungsbereich für einen Edit-Lauf
Ein **Edit-Lauf** umfasst eine Folge von Anweisungen, die mit der Anweisung START-LLM-CREATION oder START-LLM-UPDATE beginnt und mit der nächsten Anweisung START-LLM-CREATION oder START-LLM-UPDATE oder mit der END-Anweisung endet.
Ein Operandenwert gilt für einen Edit-Lauf, wenn er als Standardwert in anderen Anweisungen desselben Edit-Laufs verwendet werden kann.
- Geltungsbereich für einen BINDER-Lauf
Ein **BINDER-Lauf** ist eine Folge von Anweisungen, die nach dem Ladeaufruf für den BINDER beginnt und mit der END-Anweisung endet.
Ein Operandenwert gilt für einen BINDER-Lauf, wenn er als Standardwert in anderen Anweisungen desselben BINDER-Laufs verwendet werden kann.

Beispiel

BINDER-Lauf

```

START-BINDER _____ (1)
START-LLM-CREATION INTERNAL-NAME=X
INCLUDE-MODULES ...
.
.
SAVE-LLM LIBRARY=LIB1,ELEMENT=*CURRENT-NAME _____ (2)
INCLUDE-MODULES ...
.
.
SAVE-LLM LIBRARY=*CURRENT,ELEMENT=A1 _____ (3)
INCLUDE-MODULES ...
.
.
SAVE-LLM LIBRARY=*CURRENT,ELEMENT=*CURRENT-NAME _____ (4)

START-LLM-UPDATE LIBRARY=LIB2,ELEMENT=B
REPLACE-MODULES ...
.
.
SAVE-LLM LIBRARY=*CURRENT,ELEMENT=B _____ (5)

END

```

Erster Edit-Lauf

Zweiter Edit-Lauf

Bedeutung

- (1) Ladeaufruf für den BINDER
- (2) Der interne Name X wird aus der Anweisung START-LLM-CREATION desselben Edit-Laufs als Elementname übernommen.
- (3) Die Programmbibliothek LIB1 wird aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs übernommen.
- (4) Die Programmbibliothek LIB1 und der Elementname A1 werden aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs übernommen.
- (5) Die Programmbibliothek LIB2 wird aus der Anweisung START-LLM-UPDATE desselben Edit-Laufs übernommen.

Module

Module können Bindemodule (OMs), Bindelademodule (LLMs) oder beide sein. Eingabequellen für Module sind:

- für OMs eine Programmbibliothek (Elementtyp R), eine Bindemodulbibliothek (OML) oder die EAM-Bindemoduldatei,
- für LLMs eine Programmbibliothek (Elementtyp L).

Eröffnen der Eingabequellen

Programmbibliotheken, Modulbibliotheken und die EAM-Bindemoduldatei werden in der Reihenfolge, in der sie in der Anweisungsfolge angegeben sind, für die Eingabe eröffnet. Sie bleiben während des gesamten BINDER-Laufs eröffnet und werden erst mit der Anweisung END geschlossen. Eine Eingabebibliothek kann daher nicht gleichzeitig als Ausgabebibliothek verwendet werden. Für alle Dateien, die im Eingabemodus gelesen werden, nutzt BINDER die „secondary read“-Funktion des DVS (siehe Handbuch „Einführung in das DVS“ [15]).

Konkurrierende BINDER-Läufe

Alle Eingabequellen, auf die in BINDER-Anweisungen zugegriffen wird, werden nur lesbar eröffnet und somit für schreibende Zugriffe (wie bei SAVE-LLM) gesperrt. Dadurch werden Lesezugriffe verschiedener Tasks auf ein- und dasselbe LLM ermöglicht. So können z.B. mehrere Tasks ein LLM gleichzeitig mit START-LLM-UPDATE in den BINDER-Arbeitsbereich bringen und ändern. Eine Task kann das geänderte LLM aber nur dann unter demselben Elementnamen und derselben Elementversion abspeichern, wenn keine andere Task dieses LLM gerade bearbeitet.

4.2 Ausgaben des BINDER

Ausgaben des BINDER sind:

- gespeicherte LLMs. Jedes LLM wird mit einem eigenen Elementnamen in einer Programmbibliothek gespeichert.
- Listen mit Auskunftsinformationen über den Zustand des aktuellen LLM oder mit Endinformationen über das gespeicherte LLM.

Mit Hilfe der Anzeigefunktionen (siehe [Abschnitt „Anzeigefunktionen“ auf Seite 127ff](#)) kann sich der Benutzer Informationen über Standardwerte, Symbole und Bibliothekselemente anzeigen lassen.

4.2.1 Gespeicherte LLMs

Der BINDER kann in einem BINDER-Lauf auch *mehrere* LLMs binden. Mit jeder Anweisung SAVE-LLM wird das aktuelle LLM in seiner jeweiligen Struktur als *eigenes* Bibliothekselement vom Typ L in einer Programmbibliothek gespeichert. LLMs werden nur mit der Anweisung SAVE-LLM gespeichert. Wenn eine neue Anweisung START-LLM-CREATION oder START-LLM-UPDATE gegeben wird, das aktuelle LLM jedoch nicht vorher mit der Anweisung SAVE-LLM gespeichert wurde, wird das vorhergehende aktuelle LLM überschrieben. Die Anweisung END beendet den BINDER-Lauf, ohne das aktuelle LLM zu speichern.

4.2.2 Listen

Listen mit Informationen über den Zustand des LLM werden mit der Anweisung SHOW-MAP ausgegeben, Listen mit Endinformation über das gespeicherte LLM mit der Anweisung SAVE-LLM (Operand MAP=YES).

Das Ausgabeziel kann sein:

- die Systemdatei SYSLST,
- eine Datei, die automatisch mit einem impliziten Kommando SHOW-FILE (siehe Handbuch „Kommandos“ [5]) auf SYSOUT ausgegeben wird. Diese Datei wird unter dem Standarddateinamen angelegt und nach Beenden der Ausgabe sofort wieder gelöscht, falls der Benutzer das Löschen nicht verbietet.
- eine Datei, deren Name durch den Benutzer festgelegt wird.
- eine Datei, die durch den Standard-Dateikettungsamen oder einen angegebenen Dateikettungsamen festgelegt wurde.

- ein benutzereigenes Unterprogramm, das die gelieferten Informationen auswertet. Die Informationen werden in eine ISAM-Datei ausgegeben, auf die das Unterprogramm zugreifen muss.

Diese Ausgabeziele stehen auch in der Anweisung SHOW-LIBRARY-ELEMENTS zur Verfügung.

Die bei SHOW-MAP oder SHOW-LIBRARY-ELEMENTS erzeugten Listen sind (mit Ausnahme der auf SYSLST ausgegebenen Listen) standardmäßig ISAM-Dateien mit ISAM-Schlüsseln der Länge 8. Die ISAM-Schlüssel können zur Auswertung der Listen verwendet werden. Im Anhang finden Sie die Beschreibung der ISAM-Schlüssel ([Seite 369f](#)).

Welche Informationslisten ausgegeben werden, bestimmen die Operanden in der Anweisung SHOW-MAP. Standardeinstellungen können mit einer vorhergehenden Anweisung MODIFY-MAP-DEFAULTS geändert werden. Eine Liste mit dem Namen *STD wird immer automatisch erzeugt. Der Benutzer kann sich aber in einem BINDER-Lauf auch andere Listen mit MODIFY-MAP-DEFAULTS definieren, weil er sie einzeln benennen kann (Operand MAP-NAME). Der Benutzer kann sich dann für z.B. für jede Liste andere Standardwerte definieren bzw. den Umfang der Listen bestimmen. Wenn die unter MAP-NAME angegebene Liste noch nicht existierte, wird sie neu angelegt. Ansonsten werden die Werte für die Liste aktualisiert. Mit SHOW-MAP können Listen *nicht* definiert werden. Das ist nur mit MODIFY-MAP-DEFAULTS möglich. Eine SHOW-MAP-Anweisung auf eine existierende Liste hat keinerlei Auswirkungen auf spätere SHOW-MAP-Anweisungen und ändert auch nichts an anderen MAP-Definitionen.

Die folgende Tabelle gibt eine Übersicht über die verschiedenen Listen mit den zugehörigen Operanden. Anschließend sind die einzelnen Listen an einem Beispiel dargestellt.

Für alle Listen kann eine Kopfzeile mit einem Benutzerkommentar festgelegt werden (Operand USER-COMMENT).

Liste	Operand
Liste der Abkürzungen (Help Information)	HELP-INFORMATION
Globale Informationen (Global Information)	GLOBAL-INFORMATION
Übersicht über die logische Struktur (Logical Structure)	LOGICAL-STRUCTURE
Übersicht über die physische Struktur (Physical Structure)	PHYSICAL-STRUCTURE
Programmübersicht (Program Map)	PROGRAM-MAP
Liste der unbefriedigten Externverweise (Unresolved Definitions List)	UNRESOLVED-LIST

Liste	Operand
Sortierte Liste der Programmdefinitionen (Sorted Symbols Definitions List)	SORTED-PROGRAM-MAP
Liste der Pseudoregister (Pseudo Registers List)	PSEUDO-REGISTER
Liste der nicht benutzten Module (Unused Modules List)	UNUSED-MODULE-LIST
Gemischte Module (Merged Modules) (werden in den anderen Listen dargestellt)	MERGED-MODULES
Liste der mehrfachen Programmdefinitionen (Duplicate Symbols Definitions List)	DUPLICATE-LIST
Liste mit Eingabeinformationen (Input Information)	INPUT-INFORMATION
Liste mit BINDER-Anweisungen (Statement List)	STATEMENT-LIST

Beispiel für Listenausgabe

```

/start-binder _____ (1)
//mod-map-def user-comment=c'LISTEN-BEISPIEL', - _____ (2)
//          help-information=*yes, - _____ (3)
//          global-information=*yes, - _____ (4)
//          logical-structure=*yes(res-scope=*yes,hsi-code=*yes), - - (5)
//          physical-structure=*yes, - _____ (6)
//          program-map=*par(def=*all,inv-xref-list=*all,ref=*all), - (7)
//          unresolved-list=*yes( - _____ (8)
//                                noref=*yes), - _____ (9)
//          sorted-program-map=*yes, - _____ (10)
//          pseudo-register=*yes, - _____ (11)
//          unused-module-list=*yes, - _____ (12)
//          duplicate-list=*yes(inverted-xref-list=*yes), - _____ (13)
//          merged-modules=*yes, - _____ (14)
//          input-information=*yes, - _____ (15)
//          statement-list=*yes, - _____ (16)
//          output=*syslst _____ (17)
//start-statement-recording _____ (18)
//start-llm-creation internal-name=complex1 _____ (19)
//include-modules library=bnd.llmlib,element=(autolinkl,autolinkr) _____ (20)
//save-llm library=bnd.llmlib,element=complex1 _____ (21)
//end _____ (22)

```

- (1) Aufruf des BINDER
- (2) Definieren von Standardwerten für die BINDER-Listen. Für alle Listen wird eine Kopfzeile mit dem Titel „LISTEN-BEISPIEL“ ausgegeben.

- (3) Liste der Abkürzungen
- (4) Globale Informationen
- (5) Übersicht über die logische Struktur
- (6) Übersicht über die physische Struktur
- (7) Programmübersicht
- (8) Liste der unbefriedigten Externverweise
- (9) Liste der nicht referenzierten Externverweise
- (10) Sortierte Liste der Programmdefinitionen
- (11) Liste der Pseudoregister
- (12) Liste der unbenutzten Module
- (13) Liste der mehrfachen Programmdefinitionen
- (14) In den Listen werden auch die gemischten Module ausgegeben
- (15) Liste mit Eingabeinformationen
- (16) Liste der protokollierten BINDER-Anweisungen
- (17) Das Ausgabeziel ist die Systemdatei SYSLST
- (18) Einschalten der Protokollierung von BINDER-Anweisungen
- (19) Ein LLM mit dem internen Namen COMPLEX1 wird erzeugt
- (20) In das LLM werden LLMs AUTOLINKL und AUTOLINKR eingefügt, die in der Programmbibliothek BND.LLMLIB abgespeichert sind
- (21) Das LLM wird als Element vom Typ L in der Programmbibliothek BND.LLMLIB abgespeichert. Das Bibliothekselement erhält den Elementnamen COMPLEX1. Durch die BINDER-Anweisung SAVE-LLM wird die Ausgabe der BINDER-Liste ausgelöst, wenn bei SAVE-LLM MAP=YES eingestellt ist. Zur Listenausgabe werden die unter (2) bis (17) eingestellten Standardwerte verwendet.
- (22) Beenden des BINDER-Laufs

Die unter Punkt (3) bis (16) ausgegebenen Listen sind anschließend näher erläutert.

Liste der Abkürzungen (3)

In dieser Liste sind die Abkürzungen erklärt, die in den nachfolgenden Listen verwendet werden (z.B. SD für CSECT).

BINDER V02.7A *HELP INFORMATION*
 LISTEN-BEISPIEL
 ATTRIBUTES CONVERSION

LLM: COMPLEX1

DATE=2016-02-03 10:43:51 PAGE 1

```

+----- AMODE: S=24 BITS, X=31 BITS, A=ANY
| +----- RMODE: S=24 BITS, A=ANY
| | +----- ALIGNMENT: P=PAGE ALIGNED, O=OTHER
| | | +----- ACCESS: R=READ-ONLY
| | | | +----- INVISIBILITY: I=INVISIBLE, R=RUN-TIME
| | | | | +----- RESIDENT: R=RESIDENT
| | | | | | +----- PUBLIC/PRIVATE: P=PUBLIC
| | | | | | | +----- PRIVILEGE: P=PRIVILEGED
    
```

V V V V V V V V (. = NOT SPECIFIED)

S S . R I R P P

X A P . R . . .

A 0

MAP GLOSSARY

```

-----
: CM : COMMON           : SD : CSECT           :
: ER : EXTERN           : SUB: SUB-LLM        :
: GM : GROSSMODULE      : VC : VCON            :
: LD : ENTRY            : WX : WEAK-EXTERN     :
: LLM: LINK AND LOAD MODULE : XD : XDSECT-DEF       :
: OM : OBJECT MODULE    : XR : XDSECT-REF       :
: RT : RUN-TIME         :    :                     :
-----
    
```

--- END OF SECTION ---

Globale Informationen (4)

Diese Liste enthält folgende Informationen:

ENTRY POINT

Adresse und Lage der Startadresse des LLM (bezogen auf die zugehörige Slice), HSI-Code, Memory-Access-Mode sowie Name des Symbols.

Diese Angabe ist nur nach dem Speichern des LLM von Bedeutung. Der Memory-Access-Mode ist vorläufig nur für interne Anwendungen auf Anlagen mit SPARC-Architektur relevant.

LOAD POINT

Ladeadresse des LLM.

LLM LENGTH

Länge des LLM.

COPYRIGHT

Copyright-Information.

Die folgenden Angaben sind nur nach dem Abspeichern des LLM von Bedeutung.

LIBRARY NAME

Dateiname der Programmbibliothek, in der das LLM gespeichert wurde.

ELEMENT NAME/VERSION

Elementname und Elementversion des LLM in der Programmbibliothek.

LLM FORMAT

Format, mit dem das LLM abgespeichert wurde:.

Format 1: Das Laden des LLM ist in allen BS2000-Versionen möglich.

Format 2: Das Laden des LLM ist in allen BS2000-Versionen möglich.

Format 3: Das Laden des LLM ist erst ab BS2000 V3.0A möglich.

Format 4: Das Laden des LLM ist nur mit BLSSERV ab BS2000 V3.0A möglich.

SLICE TYPE

Typ der Slice:

SINGLE Einzel-Slice

BY-USER benutzerdefinierte Slice

BY-ATTR nach Attributen gebildete Slice

Zusätzlich werden die Attribute ausgegeben, aus denen die Slice gebildet wurde. Angabe abgekürzt nach der Liste der Abkürzungen (siehe [Seite 144](#)).

SYMBOL DICTIONARY

Externadressbuch gespeichert: ja/nein

RELOCATION DATA

Relativierungsinformation gespeichert: ja/nein

LOGICAL STRUCTURE

Art der gespeicherten Strukturinformation:

WHOLE-LLM gesamte Strukturinformation

OBJECT-MODULES nur eine Struktur mit dem internen Namen des LLM
und den Bindemodulen wird gespeichert

NONE nur der interne Name des LLM wird gespeichert

TEST SUPPORT

Test- und Diagnoseinformation gespeichert: ja/nein



Dieser Wert zeigt nur die Einstellung des Operanden TEST-SUPPORT der entsprechenden BINDER-Anweisung an. Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen.

```

BINDER V02.7A  *GLOBAL INFORMATION*          LLM:  COMPLEX1      DATE=2016-02-03 10:43:51 PAGE    2
LISTEN-BEISPIEL
      ADDRESS  IN SLICE          HSI  MMODE  SYMBOL
ENTRY POINT: 00000000      1 + 00000000 /7500  TU4K  AUTOA
-----
LOAD POINT:  00000000
-----
LLM LENGTH:  00000800
-----
COPYRIGHT: 2016 FUJITSU TECHNOLOGY SOLUTIONS GMBH. ALL RIGHTS RESERVED
-----
LIBRARY NAME: :CTID:$USERID.BND.LLMLIB
-----
ELEMENT NAME/VERSION: COMPLEX1/@
-----
LLM FORMAT:      1
-----
SLICE TYPE: SINGLE
-----
SYMBOL DICTIONARY: YES
-----
RELOCATION DATA: YES
-----
LOGICAL STRUCTURE: WHOLE-LLM
-----
TEST SUPPORT: YES
-----

```

--- END OF SECTION ---

Übersicht über die logische Struktur (5)

Diese Liste besteht aus zwei Teilen.

Der erste Teil ist eine Darstellung des Strukturbaums des LLM. Er enthält die internen Namen der eingefügten Bindemodule und Sub-LLMs in der Reihenfolge, wie sie im Strukturbaum eingefügt wurden (von links nach rechts).

Es bedeuten:

SLICE	Kennzeichen für die Slice Bezieht sich auf die Spalte SLICE in der Übersicht über die physische Struktur (siehe Seite 150). Ist ein logischer Knoten auf mehrere Slices verteilt, so ist für die Nummer der Slice '-' angegeben.	
TYPE	Typ des Knotens im Strukturbaum des LLM.	
	LLM	Root
	OM	Bindemodul
	GM	Großmodul
	SUB	Sub-LLM
	RT	Runtime (falls das Modul als Laufzeit-Modul betrachtet werden soll)
HSI	Typ des Codes des LLM.	
	/7500	/390-Code
	/4000	RISC(MIPS)-Code
	MIXED	Mixed-Binary-Code (= /390 + RISC(MIPS))
	SPARC	SPARC-Code
	MIXSP	Mixed-SPARC-Code (= /390 + SPARC)
	X86	X86-Code
	MIXX8	Mixed-X86-Code (= /390 + X86)
MMODE	Memory-Access-Mode des LLM.	
	TU4K	TU-Code 4K-abhängig
	NATIV	NATIVE TU-Code unabhängig von der Speicherseitengröße
	COMP	COMPATIBLE Code, kann mit TU4K oder NATIV gebunden werden
	TPR	TPR-Code
	UNDEF	Undefinierter MMODE
	UNKNW	Unbekannter MMODE

UNSPEC Nicht spezifizierter MMODE
 LEVEL Stufe des Knotens im Strukturbaum des LLM.
 STR# Fortlaufende Nummer des Knotens im Strukturbaum des LLM
 Auf diese Nummer wird im zweiten Teil der Liste Bezug genommen.
 NAME Interner Name des Knotens im Strukturbaum des LLM.
 T&D Test- und Diagnoseinformation vorhanden: ja/nein

BINDER V02.7A *LOGICAL STRUCTURE* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 3
 LISTEN-BEISPIEL
 SLICE TYPE HSI MMODE LEVEL STR# NAME T&D

SLICE	TYPE	HSI	MMODE	LEVEL	STR#	NAME	T&D
1	LLM	/7500	TU4K	0	1	COMPLEX1	
1	SUB	/7500	TU4K	1	2	AUTOLINKL	
1	SUB	/7500	TU4K	2	3	SUB1	
1	OM	/7500	TU4K	3	4	AUTOA	NO
1	OM	/7500	TU4K	3	5	AUTOC	NO
1	OM	/7500	TU4K	3	6	AUTO2	NO
1	SUB	/7500	TU4K	2	7	SUB2	
1	OM	/7500	TU4K	3	8	AUTOA	NO
1	SUB	/7500	TU4K	3	9	SUB3	
1	OM	/7500	TU4K	4	10	AUTOA	NO
1	OM	/7500	TU4K	4	11	AUTOC	NO
1	OM	/7500	TU4K	4	12	AUTO2	NO
1	SUB	/7500	TU4K	1	13	AUTOLINKR	
1	SUB	/7500	TU4K	2	14	SUB1	
1	OM	/7500	TU4K	3	15	AUTO21	NO
1	SUB	/7500	TU4K	2	16	SUB2	
1	OM	/7500	TU4K	3	17	AUTO22	NO
1	SUB	/7500	TU4K	3	18	SUB3	
1	OM	/7500	TU4K	4	19	AUTO23	NO

--- END OF SECTION ---

Der zweite Teil der Liste gibt die Pfadnamen an, die in den Operanden HIGH-PRIORITY-SCOPE, LOW-PRIORITY-SCOPE oder FORBIDDEN-SCOPE angegeben wurden.

PATH Fortlaufende Nummer des Pfads
 Auf diese Nummer wird im ersten Teil der Liste Bezug genommen.
 STR# Fortlaufende Nummer des Knotens im Strukturbaum des LLM
 Diese Nummer dient zur Bezugnahme auf den ersten Teil der Liste.

PATHNAME
 Pfadname, der im Operanden HIGH-PRIORITY-SCOPE, LOW-PRIORITY-SCOPE oder FORBIDDEN-SCOPE angegeben wurde.

BINDER V02.7A *SCOPE PATH INFORMATION* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 4
 LISTEN-BEISPIEL
 PATH STR# PATHNAME

PATH	STR#	PATHNAME
-----	-----	-----

--- END OF SECTION ---

Übersicht über die physische Struktur (6)

Diese Liste enthält folgende Informationen:

- SLICE** Kennzeichen für die Slice oder für die Region.
Auf dieses Kennzeichen wird in den anderen Listen in der Spalte SLICE Bezug genommen.
- ATTRIB** Attribute der Slice
Angabe abgekürzt entsprechend der Liste der Abkürzungen (siehe [Seite 144](#)).
- LENGTH** Länge der Slice
Die Länge ist bezogen auf die Länge der Textinformation (TXT). In der Liste werden auch alle Slices mit der Länge 0 protokolliert. Slices mit der Länge 0 werden beim Speichern des LLM jedoch nur übernommen, wenn sie vom Benutzer definiert wurden. Einzel-Slices mit der Länge 0 und Slices, die nach Attributen gebildet wurden, mit der Länge 0, werden nicht im LLM gespeichert.
- NAME** Name der Slice oder der Region (siehe [Seite 77](#)).

```

BINDER V02.7A   *PHYSICAL STRUCTURE*           LLM:  COMPLEX1       DATE=2016-02-03 10:43:51 PAGE    5
LISTEN-BEISPIEL
SLICE  ATTRIB  LENGTH  NAME
-----
 1 .A.....  00000800  PUU-ROU-RTU-RMU

```

--- END OF SECTION ---

Programmübersicht (7)

Die Programmübersicht beschreibt die Module, die das LLM aufbauen, mit ihrem Namen, ihrer Ladeadresse, ihrer Länge, ihren Attributen und ihrer Herkunft. Wahlfrei kann die Programmübersicht enthalten:

- Die Programmdefinitionen und COMMON-Bereiche in den einzelnen Modulen (Operand DEFINITIONS),
- Die Referenzen in den einzelnen Modulen (Operand REFERENCES),
- Eine Querverweisliste (Operand INVERTED-XREF-LIST).
Sie beschreibt die Liste von Externverweisen, die durch eine aktuelle Programmdefinition (CSECT, ENTRY,..) befriedigt wurden.

Im Einzelnen enthält die Programmübersicht folgende Informationen:

OBJ	Typ des Moduls oder des Symbols Angabe abgekürzt entsprechend der Liste der Abkürzungen (siehe Seite 144).
NAME	Name des Moduls oder des Symbols
ADDRESS	Ladeadresse bei CSECTs und ENTRYs
LENGTH	Länge der Textinformation (TXT) bei CSECTs.
ATTRIB	Attribute der CSECTs. Angabe abgekürzt entsprechend der Liste der Abkürzungen (siehe Seite 144). Zur Information auch für ENTRYs angegeben (relevant ist nur das Attribut INVISIBLE).
SLICE	Kennzeichen für die Slice Bezieht sich auf die Spalte SLICE in der Übersicht über die physische Struktur (siehe Seite 150).
TYPE	Nur für die Querverweisliste (INVERTED-XREF-LIST) von Bedeutung. Gibt den Typ der befriedigten Referenzen an. Angabe abgekürzt entsprechend der Liste der Abkürzungen (siehe Seite 144).
RESOLVED	Nur für Referenzen von Bedeutung. Adresse, die in die befriedigten Referenzen eingetragen ist. Falls eine Adresse eingetragen ist, die mit der Anweisung SET-EXTERN-RESOLUTION festgelegt wurde, enthält diese Spalte den Eintrag EXT-RES. In diesem Fall werden in einer Zusatzzeile der Name der Referenz und die eingetragene Adresse protokolliert.

IN MODULE

Nur für Referenzen und für die Querverweisliste (INVERTED-XREF-LIST) von Bedeutung.

Gibt an, in welchem Modul die unter OBJ und NAME angegebenen Externverweise befriedigt wurden.

FROM ELEMENT#

Nur für Bindemodule (OMs) von Bedeutung.

Enthält das Kennzeichen für das Element. Dieses Kennzeichen ist festgelegt in der Liste der Eingabeinformationen, Spalte ELEM# (siehe [Seite 157](#)).

OM#

Kennzeichen für das Bindemodul (OM).

Auf diese Kennzeichen wird in den anderen Listen Bezug genommen, indem das Kennzeichen dem Modulnamen vorangestellt wird.

SLICE

Nur für Referenzen und für die Querverweisliste (INVERTED-XREF-LIST) von Bedeutung.

Gibt die Slice an, die das Symbol enthält, das den unter OBJ angegebenen Externverweis befriedigt. Bezieht sich auf die Spalte SLICE in der Übersicht über die physische Struktur (siehe [Seite 150](#)).

SATIS

Gibt an, auf welche Art Externverweise im LLM behandelt wurden:

NOREF Der Externverweis wurde nicht befriedigt, da keine Relativierungsinformation vorhanden war.

UNRES Unbefriedigter Externverweis

SLICE Der Externverweis wurde in dieser Slice befriedigt.

ERREX Ein Externverweis ist vorhanden, für den mit der Anweisung SET-EXTERN-RESOLUTION eine Adresse eingetragen wurde.

Zusätzliche Werte für benutzerdefinierte Slices:

HIGH Das Symbol wurde in einer Slice befriedigt, die höher ist als die Slice, die das Symbol enthält.

LOW Das Symbol wurde in einer Slice befriedigt, die niedriger ist als die Slice, die das Symbol enthält.

OTHER Das Symbol wurde in einer Slice befriedigt, die in einer anderen Region liegt als die Slice, die das Symbol enthält.

CONC Das Symbol wurde in einer Slice befriedigt, die exklusiv zu der Slice ist, die das Symbol enthält.

BINDER V02.7A *PROGRAM MAP* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 6
 LISTEN-BEISPIEL

OBJ	NAME	ADDRESS	LENGTH	ATTRIB	SLICE	TYPE	RESOLVED	IN MODULE/FROM	ELEMENT#	OM#	SLICE	SATIS
OM	AUTOA				1			1		1		
SD	AUTOA	00000000	00000040	AA.....	1							
ER	AUTOB						FFFFFFFF					UNRES
WX	AUTOWX						FFFFFFFF					UNRES
VC	ENTRYB1						FFFFFFFF					UNRES
OM	AUTOC				1			1		2		
SD	AUTOC	00000040	00000018	AA.....	1							
VC	AUTOCOM						FFFFFFFF					UNRES
OM	AUTO2				1			1		3		
SD	AUTO2	00000058	0000000C	AA.....	1							
VC	AUTO21						00000110	AUTO21		8	1	SLICE
VC	AUTO22						00000118	AUTO22		9	1	SLICE
VC	AUTO23						00000168	AUTO23		10	1	SLICE
OM	AUTOA				1			1		4		
SD	AUTOA	00000068	00000040	AA.....	1							
ER	AUTOB						FFFFFFFF					UNRES
WX	AUTOWX						FFFFFFFF					UNRES
VC	ENTRYB1						FFFFFFFF					UNRES
OM	AUTOA				1			1		5		
SD	AUTOA	000000A8	00000040	AA.....	1							
ER	AUTOB						FFFFFFFF					UNRES
WX	AUTOWX						FFFFFFFF					UNRES
VC	ENTRYB1						FFFFFFFF					UNRES
OM	AUTOC				1			1		6		
SD	AUTOC	000000E8	00000018	AA.....	1							
VC	AUTOCOM						FFFFFFFF					UNRES
OM	AUTO2				1			1		7		
SD	AUTO2	00000100	0000000C	AA.....	1							
VC	AUTO21						00000110	AUTO21		8	1	SLICE
VC	AUTO22						00000118	AUTO22		9	1	SLICE
VC	AUTO23						00000168	AUTO23		10	1	SLICE
OM	AUTO21				1			2		8		
SD	AUTO21	00000110	00000006	AA.....	1							
						VC		AUTO2		3	1	SLICE
						VC		AUTO2		7	1	SLICE
OM	AUTO22				1			2		9		
SD	AUTO22	00000118	00000006	AA.....	1							
						VC		AUTO2		3	1	SLICE
						VC		AUTO2		7	1	SLICE
OM	AUTO23				1			2		10		
SD	AUTO23	00000168	00000006	AA.....	1							
						VC		AUTO2		3	1	SLICE
						VC		AUTO2		7	1	SLICE

--- END OF SECTION ---

Liste der COMMON-Bereiche

Diese Liste enthält folgende Informationen:

NAME Name des COMMON-Bereichs.

ADDRESS

Adresse der CSECT, mit der der COMMON-Bereich initialisiert wurde;
FFFFFFFF, wenn der COMMON-Bereich nicht initialisiert wurde.

LENGTH Länge des COMMON-Bereichs.

ATTRIB Attribute des COMMON-Bereichs

Angabe abgekürzt entsprechend der Liste der Abkürzungen (siehe [Seite 144](#)).

TYPE Nur für die Querverweisliste (INVERTED-XREF-LIST) von Bedeutung.

Gibt den Typ der befriedigten Referenzen an. Angabe abgekürzt entsprechend der Liste der Abkürzungen (siehe [Seite 144](#)).

IN MODULE

Name des Moduls, das den COMMON-Bereich enthält. *NOT PROMOTED bedeutet, dass der COMMON-Bereich nicht initialisiert wurde.

OM#

Auf diese Kennzeichen wird in den anderen Listen Bezug genommen, indem das Kennzeichen dem Symbol vorangestellt wird.

SLICE

Nur für die Querverweisliste (INVERTED-XREF-LIST) von Bedeutung.
Kennzeichen für die Slice. Bezieht sich auf die Spalte SLICE in der Übersicht über die physische Struktur (siehe [Seite 150](#)).

```

BINDER V02.7A *COMMON LIST* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 7
LISTEN-BEISPIEL
NAME ADDRESS LENGTH ATTRIB TYPE IN MODULE OM# SLICE SATIS
----
AUTOC2 FFFFFFFF 00000050 AA..... AUTOC 2 1
          *NOT-PROMOTED
AUTOC2 FFFFFFFF 00000050 AA..... AUTOC 6 1
          *NOT-PROMOTED
AUT022 00000118 00000050 AA..... AUT023 10 1
          AUT022 9 1
          --- END OF SECTION ---

```

Liste der unbefriedigten Externverweise (8)

Diese Liste enthält die Namen der unbefriedigten Externverweise, für die in der Programmübersicht in der Spalte RESOLVED die Adresse X'FFFFFFFF' und in der Spalte SATIS der Wert UNRES eingetragen ist (siehe [Seite 152](#)).

```

BINDER V02.7A *UNRESOLVED REFERENCES* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 8
LISTEN-BEISPIEL
VC AUTOCOM VC ENTRYB1 WX AUTOWX ER AUTOB
          --- END OF SECTION ---

```


Liste der Pseudoregister (11)

Diese Liste enthält folgende Informationen:

NAME Name des Pseudoregisters
 OFFSET Bezugspunkt des Pseudoregisters
 LENGTH Länge, die dem Pseudoregister zugewiesen wurde.
 REF LEN Länge des Pseudoregisters, die das Modul benutzt.
 IN MODULE
 Name des Moduls, das das Pseudoregister enthält.

Letzte Zeile: PSEUDO-REGISTER VECTOR LENGTH
 Gibt die Länge des Pseudoregistervektors an.

```
BINDER V02.7A *PSEUDO-REGISTERS*          LLM: COMPLEX1      DATE=2016-02-03 10:43:51 PAGE  11
LISTEN-BEISPIEL
PSEUDO-REGISTERS  NAME                      OFFSET  LENGTH  REF LEN  IN MODULE
-----
PSEUDO-REGISTER VECTOR LENGTH:           0
-----
--- END OF SECTION ---
```

Liste der unbenutzten Module (12)

Diese Liste enthält die Namen der Module, in denen keine Symbole zur Befriedigung von Externverweisen benutzt wurden.

Die Nummer gibt das Modulkennzeichen an, das in der Programmübersicht in der Spalte OM# für das Modul festgelegt ist (siehe [Seite 152](#)).

```
BINDER V02.7A *UNUSED MODULES*          LLM: COMPLEX1      DATE=2016-02-03 10:43:51 PAGE  12
LISTEN-BEISPIEL
UNUSED MODULES LIST
-----
  1 AUTOA                      2 AUTOC                      3 AUTO2
  4 AUTOA                      5 AUTOA                      6 AUTOC
  7 AUTO2
--- END OF SECTION ---
```

Liste der mehrfachen Programmdefinitionen (13)

Die Einträge haben die gleiche Bedeutung wie in der Programmübersicht (siehe [Seite 151](#)).

```

BINDER V02.7A  *DUPLICATE SYMBOLS*          LLM:  COMPLEX1      DATE=2016-02-03 10:43:51 PAGE  13
LISTEN-BEISPIEL
NAME           TYPE  IN MODULE          OM#  SLICE  TYPE  IN MODULE          OM#  SLICE
-----
AUTOA          SD   AUTOA              1    1
AUTOA          SD   AUTOA              4    1
AUTOA          SD   AUTOA              5    1
AUTOA          SD   AUTOA              2    1
AUTOA          SD   AUTOA              6    1
AUTOA          CM   AUTOA              2    1
AUTOA          CM   AUTOA              6    1
AUTOA          SD   AUTOA              3    1
AUTOA          SD   AUTOA              7    1
AUTOA          SD   AUTOA              9    1
                VC   AUTO2              3    1
                VC   AUTO2              7    1
AUTOA          CM   AUTO23             10    1

```

--- END OF SECTION ---

Liste der gemischten Module (14)

Gemischte Module (soweit vorhanden) erscheinen in allen Listen, in denen Symbole vorkommen. Die gemischten Module sind in Klammern dargestellt. Das daraus entstandene Modul ist ein Großmodul, das in den Listen mit GM bezeichnet ist. Ein Beispiel dazu ist auf [Seite 160ff](#) zu finden.

Liste mit Eingabeinformationen (15)

Die Liste besteht aus zwei Teilen:

Erster Teil der Liste

Für jedes eingefügte Modul enthält die Liste folgende Informationen:

NAME Elementname

TYPE Elementtyp in der Programmbibliothek

L LLM

R Bindemodul (OM)

VERSION Elementversion in der Programmbibliothek. Der Eintrag @ bedeutet den Standardwert für die höchste Version bei Programmbibliotheken (siehe Handbuch „LMS“ [3]).

DATE Erstellungsdatum

FILE ID Kennzeichen für die Programmbibliothek bzw. -datei. Stellt die Beziehung zum zweiten Teil der Liste her.

- BIND** Gibt an, wie das Modul (ELEMENT) eingefügt wurde.
- EXPL** Das Modul wurde mit der Anweisung INCLUDE-MODULES oder REPLACE-MODULES eingefügt.
- AUTO** Das Modul wurde mit Autolink eingefügt.
- IMPL** Das Modul wurde durch einen INCLUDE-Satz im Bindemodul eingefügt.
- ELEM#** Kennzeichen für das Element
Auf dieses Kennzeichen wird in anderen Listen Bezug genommen.
- PATH** Pfadname des Sub-LLM, auf das in der Anweisung INCLUDE-MODULES oder REPLACE-MODULES verzweigt wurde. Anschließend folgt eine Liste mit den Namen aller Bindemodule (OMs), aus denen das Sub-LLM besteht (siehe „Übersicht über die logische Struktur“ [Seite 148](#)).
Jedem Modulnamen ist eine Nummer für das Modul vorangestellt, die in der Programmübersicht in der Spalte OM# für das Modul festgelegt ist (siehe [Seite 151](#)).

```

BINDER V02.7A *INPUT INFORMATION*          LLM:  COMPLEX1      DATE=2016-02-03 10:43:51 PAGE  14
LISTEN-BEISPIEL
      NAME                                TYPE  VERSION                                DATE      FILE ID  BIND  ELEM#
-----
ELEMENT: AUTOLINKL                        L  @                                1991-02-01      1  EXPL  1
-----
      PATH: AUTOLINKL
      MODULES
-----
      1 AUTOA                                2 AUTOC                                3 AUTO2
      4 AUTOA                                5 AUTOA                                6 AUTOC
      7 AUTO2
ELEMENT: AUTOLINKR                        L  @                                1992-10-02      1  EXPL  2
-----
      PATH: AUTOLINKR
      MODULES
-----
      8 AUTO21                                9 AUTO22                                10 AUTO23
      ---- END OF SECTION ----

```

Zweiter Teil der Liste

Für jede Eingabequelle enthält die Liste folgende Informationen:

FILE ID Kennzeichen für die Eingabequelle. Stellt die Beziehung zum ersten Teil der Liste her.

LINKNAME

Dateikettungsname der Eingabequelle

FILE TYPE

Typ der Eingabequelle

PLAM Programmbibliothek (Elementtyp R oder L)

OML Bindemodulbibliothek

OMF EAM-Bindemoduldatei

FILENAME

Dateiname der Eingabequelle

```
BINDER V02.7A *LINKNAME CONVERSION* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 15
LISTEN-BEISPIEL
```

```
FILE ID LINKNAME FILE TYPE FILENAME
```

```
-----
1 PLAM :CTID:$USERID.BND.LLMLIB
```

```
--- END OF SECTION ---
```

Liste der BINDER-Anweisungen (16)

Diese Liste enthält die Liste der protokollierten BINDER-Anweisungen (siehe //START-STATEMENT-RECORDING).

```
BINDER V02.7A *BINDER STATEMENTS* LLM: COMPLEX1 DATE=2016-02-03 10:43:51 PAGE 17
LISTEN-BEISPIEL
```

```
START-LLM-CREATION INTERNAL-NAME=COMPLEX1,COPYRIGHT=*PARAMETERS,INCLUSION-DEFAULTS=*PARAMETERS
INCLUDE-MODULES MODULE-CONTAINER=*LIBRARY-ELEMENT(LIBRARY=BNDBSP.LIB,ELEMENT=(AUTOLINKL,AUTOLINKR),
TYPE>(*L,*R))
```

```
SAVE-LLM MODULE-CONTAINER=*LIBRARY-ELEMENT(LIBRARY=#OUT.PL,ELEMENT=COMPLEX1),MAP=*YES
```

```
--- END OF SECTION ---
```

Listenbeispiel für ein LLM mit gemischten Modulen

Das folgende Beispiel zeigt das Aussehen der BINDER-Listen, wenn gemischte Module im LLM vorhanden sind.

```

/start-binder
% BND0500 BINDER VERSION 'V02.7A00' STARTED
//modify-map-defaults help-information=no,-
//                                global-information=no,-
//                                logical-structure=yes,-
//                                physical-structure=no,-
//                                program-map=parameters(definitions=all,-
//                                inverted-xref-list=all,references=all),-
//                                unresolved-list=yes,-
//                                sorted-program-map=yes,-
//                                pseudo-register=no,-
//                                unused-module-list=no,-
//                                duplicate-list=yes(inv-xref-list=yes),-
//                                input-information=no,-
//                                output=*syslst _____ (1)
//start-llm-creation internal-name=complex2 _____ (2)
//begin-sub-llm-statements sub-llm-name=subllm _____ (3)
//include-modules library=bnd.llmlib,element=(auto2,auto21)
//end-sub-llm-statements
% BND1120 CURRENT LOGICAL POSITION: 'COMPLEX2'
//include-modules library=bnd.llmlib,element=auto22 _____ (4)
//show-map user-comment='LISTEN-BEISPIEL VOR MERGE',MERGED-MODULES=YES (5)
//merge-modules name=subllm,path-name=complex2 _____ (6)
% BND1112 '0' KEPT ENTRIES
//show-map user-comment='LISTEN-BEISPIEL (MERGED-MODULES=YES)',- _____ (7)
//                                merged-modules=yes
//show-map user-comment='LISTEN-BEISPIEL (MERGED-MODULES=NO)',- _____ (8)
//                                merged-modules=no
//save-llm library=bnd.llmlib,element=complex2,map=no _____ (9)
% BND3101 SOME EXTERNAL REFERENCES ARE UNRESOLVED
% BND1501 LLM FORMAT : '1'
//end
% BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'

```

- (1) Nach dem BINDER-Start werden die Standardwerte für die BINDER-Listen festgelegt. Es sollen nur die Listen ausgegeben werden, die relevante Informationen für das Mischen von Modulen enthalten.
- (2) Ein LLM mit dem internen Namen COMPLEX2 wird erzeugt.
- (3) Das LLM COMPLEX2 erhält ein Sub-LLM mit dem Namen SUBLLM, in das die Bindemodule AUTO2 und AUTO21 eingefügt werden. Die beiden Bindemodule werden aus der Programmbibliothek BND.LLMLIB geholt. Das Sub-LLM wird abgeschlossen.

- (4) In das LLM wird ein weiteres Bindemodul (AUTO22) eingefügt.
- (5) Vor dem Mischvorgang werden die BINDER-Listen angezeigt. Die Listen erhalten im Listenkopf den Kommentar 'LISTEN-BEISPIEL VOR MERGE'. MERGED-MODULES=YES bewirkt, dass schon vorhandene gemischte Module in den Listen in Klammern dargestellt werden. Im LLM COMPLEX2 ist das aber nicht der Fall, wie in den Listen zu erkennen ist.
- (6) Das Sub-LLM SUBLLM wird gemischt. Es ist über den Pfad COMPLEX2 und seinen Namen SUBLLM eindeutig als Objekt für das Mischen definiert.
- (7) Die BINDER-Listen werden mit den gemischten Modulen ausgegeben. Die Listen erhalten im Listenkopf den Kommentar 'LISTENBEISPIEL (MERGED-MODULES=YES)'. In den Listen sind die gemischten Module an der geklammerten Darstellung zu erkennen. Entstanden ist ein Großmodul mit dem Namen SUBLLM.
- (8) Die BINDER-Listen werden ohne die gemischten Module ausgegeben. Die Listen erhalten im Listenkopf den Kommentar 'LISTENBEISPIEL (MERGED-MODULES=NO)'. In den Listen ist das Mischen nur noch daran zu erkennen, dass das entstandene Großmodul SUBLLM nun auch als Modul in der Programmübersicht erscheint und mit der Abkürzung GM bezeichnet ist.
- (9) Das LLM COMPLEX2 wird in der Programmbibliothek BND.LLMLIB als Typ-L-Element unter dem Namen COMPLEX2 abgespeichert. Auf die Ausgabe einer weiteren Liste wird verzichtet. BINDER stellt fest, dass nicht alle Externverweise befriedigt sind und der LLM mit Format 1 gespeichert wird (siehe [Seite 44](#)).

Die folgenden Listen werden auf SYSLST ausgegeben. An der Kopfzeile ist zu erkennen, um welche Liste es sich handelt. Die Klammern in den Listen deuten auf die gemischten Module hin.

Listen vor dem Mischvorgang

```

BINDER V02.7A  *LOGICAL STRUCTURE*          LLM:  COMPLEX2      DATE=2016-02-03 14:13:30 PAGE    1
LISTEN-BEISPIEL VOR MERGE
SLICE TYPE  HSI  MMODE  LEVEL  STR#  NAME                                     T&D
-----
1 LLM  /7500  TU4K    0      1  COMPLEX2
1 SUB  /7500  TU4K    1      2  SUBLLM
1 OM   /7500  TU4K    2      3  AUTO2
1 OM   /7500  TU4K    2      4  AUTO21
1 OM   /7500  TU4K    1      5  AUTO22
                                                    NO
                                                    NO
                                                    NO
                                                    --- END OF SECTION ---
    
```

```

BINDER V02.7A  *SCOPE PATH INFORMATION*      LLM:  COMPLEX2      DATE=2016-02-03 14:13:30 PAGE    2
LISTEN-BEISPIEL VOR MERGE
PATH  STR#  PATHNAME
-----
                                                    --- END OF SECTION ---
    
```

```

BINDER V02.7A  *PROGRAM MAP*                LLM:  COMPLEX2      DATE=2016-02-03 14:13:30 PAGE    3
LISTEN-BEISPIEL VOR MERGE
OBJ  NAME  ADDRESS  LENGTH  ATTRIB  SLICE  TYPE  RESOLVED  IN  MODULE/FROM  ELEMENT#  OM#  SLICE  SATIS
-----
OM   AUTO2  00000000 00000000 AA..... 1
SD   AUTO2  00000000 00000000 AA..... 1
VC   AUTO21 00000010 AUTO21 2 1 SLICE
VC   AUTO22 00000018 AUTO22 3 1 SLICE
VC   AUTO23 FFFFFFFF 2 UNRES
OM   AUTO21 00000010 00000006 AA..... 1
SD   AUTO21 00000010 00000006 AA..... 1
VC   AUTO21 00000010 00000006 AA..... 1
OM   AUTO22 00000018 00000006 AA..... 1
SD   AUTO22 00000018 00000006 AA..... 1
VC   AUTO22 00000018 00000006 AA..... 1
VC   AUTO22 00000018 00000006 AA..... 1
                                                    1 1 SLICE
                                                    --- END OF SECTION ---
    
```

```

BINDER V02.7A  *COMMON LIST*                LLM:  COMPLEX2      DATE=2016-02-03 14:13:30 PAGE    4
LISTEN-BEISPIEL VOR MERGE
NAME  ADDRESS  LENGTH  ATTRIB  TYPE  IN MODULE  OM#  SLICE  SATIS
-----
                                                    --- END OF SECTION ---
    
```

```

BINDER V02.7A  *UNRESOLVED REFERENCES*      LLM:  COMPLEX2      DATE=2016-02-03 14:13:30 PAGE    5
LISTEN-BEISPIEL VOR MERGE
VC AUTO23
                                                    --- END OF SECTION ---
    
```

```

BINDER V02.7A  *UNRESOLVED LONG NAMES*      LLM:  COMPLEX2      DATE=2016-02-03 14:13:30 PAGE    6
LISTEN-BEISPIEL VOR MERGE
INDEX NAME
-----
                                                    --- END OF SECTION ---
    
```

```

BINDER V02.7A   *SORTED SYMBOLS*           LLM:  COMPLEX2   DATE=2016-02-03 14:13:30 PAGE   7
LISTEN-BEISPIEL VOR MERGE
NAME           TYPE IN MODULE           OM#  SLICE
-----
AUTO2          SD  AUTO2           1    1
AUTO21         SD  AUTO21          2    1
AUTO22         SD  AUTO22          3    1
    
```

--- END OF SECTION ---

```

BINDER V02.7A   *DUPLICATE SYMBOLS*        LLM:  COMPLEX2   DATE=2016-02-03 14:13:30 PAGE   8
LISTEN-BEISPIEL VOR MERGE
NAME           TYPE IN MODULE           OM#  SLICE TYPE IN MODULE           OM#  SLICE
-----
    
```

--- END OF SECTION ---

Listen nach dem Mischen mit Ausgabe der gemischten Module

```

BINDER V02.7A   *LOGICAL STRUCTURE*        LLM:  COMPLEX2   DATE=2016-02-03 14:13:30 PAGE   1
LISTEN-BEISPIEL (MERGED-MODULES=YES)
SLICE TYPE HSI  MMODE LEVEL STR# NAME           T&D
-----
 1 LLM /7500 TU4K 0 1 COMPLEX2
 1 SUB /7500 TU4K 1 2 SUBLLM
 1 GM /7500 TU4K 2 3 SUBLLM
 1(OM ) /7500 TU4K 3 4 (AUTO2)
 1(OM ) /7500 TU4K 3 5 (AUTO21)
 1 OM /7500 TU4K 1 6 AUTO22
    
```

--- END OF SECTION ---

```

BINDER V02.7A   *SCOPE PATH INFORMATION*    LLM:  COMPLEX2   DATE=2016-02-03 14:13:30 PAGE   2
LISTEN-BEISPIEL (MERGED-MODULES=YES)
PATH STR# PATHNAME
-----
    
```

--- END OF SECTION ---

```

BINDER V02.7A   *PROGRAM MAP*             LLM:  COMPLEX2   DATE=2016-02-03 14:13:30 PAGE   3
LISTEN-BEISPIEL (MERGED-MODULES=YES)
OBJ  NAME      ADDRESS  LENGTH  ATTRIB  SLICE  TYPE  RESOLVED  IN MODULE/FROM ELEMENT#  OM#  SLICE  SATIS
-----
GM   SUBLLM      00000000 00000016 AA..... 1
SD   SUBLLM      00000000 00000016 AA..... 1
(OM) (AUTO2)    1
(SD)(AUTO2)    00000000
(VC)(AUTO21)   00000010 AUTO21 3 1 SLICE
VC   AUTO22     00000018 AUTO22 4 1 SLICE
VC   AUTO23     FFFFFFFF
(OM) (AUTO21)  1
(SD)(AUTO21)  00000010
(VC)          AUTO2 2 1 SLICE
OM   AUTO22     1
SD   AUTO22     00000018 00000006 AA..... 1
VC          AUTO2 2 1 SLICE
    
```

--- END OF SECTION ---

```

BINDER V02.7A *COMMON LIST*                LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    4
LISTEN-BEISPIEL (MERGED-MODULES=YES)
NAME          ADDRESS    LENGTH    ATTRIB    TYPE IN MODULE                OM#    SLICE Satis
-----
--- END OF SECTION ---
    
```

```

BINDER V02.7A *UNRESOLVED REFERENCES*       LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    5
LISTEN-BEISPIEL (MERGED-MODULES=YES)
VC AUTO23
--- END OF SECTION ---
    
```

```

BINDER V02.7A *UNRESOLVED LONG NAMES*       LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    6
LISTEN-BEISPIEL (MERGED-MODULES=YES)
INDEX NAME
-----
--- END OF SECTION ---
    
```

```

BINDER V02.7A *SORTED SYMBOLS*              LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    7
LISTEN-BEISPIEL (MERGED-MODULES=YES)
NAME          TYPE IN MODULE                OM#    SLICE
-----
(AUTO2)      (SD)(AUTO2)                2      1
(AUTO21)     (SD)(AUTO21)              3      1
AUTO22      SD AUTO22                  4      1
SUBLLM      SD SUBLLM                  1      1
--- END OF SECTION ---
    
```

```

BINDER V02.7A *DUPLICATE SYMBOLS*           LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    8
LISTEN-BEISPIEL (MERGED-MODULES=YES)
NAME          TYPE IN MODULE                OM#    SLICE TYPE IN MODULE                OM#    SLICE
-----
--- END OF SECTION ---
    
```

Listen nach dem Mischen ohne gemischte Module

```

BINDER V02.7A *LOGICAL STRUCTURE*           LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    1
LISTEN-BEISPIEL (MERGED-MODULES=NO)
SLICE TYPE  HSI    MMODE LEVEL STR# NAME                T&D
-----
1 LLM /7500 TU4K    0    1 COMPLEX2
1 SUB /7500 TU4K    1    2 SUBLLM
1 GM /7500 TU4K    2    3 SUBLLM
1 OM /7500 TU4K    1    6 AUTO22
--- END OF SECTION ---
    
```

```

BINDER V02.7A *SCOPE PATH INFORMATION*       LLM:  COMPLEX2    DATE=2016-02-03 14:13:30 PAGE    2
LISTEN-BEISPIEL (MERGED-MODULES=NO)
PATH STR# PATHNAME
-----
--- END OF SECTION ---
    
```

BINDER V02.7A *PROGRAM MAP* LLM: COMPLEX2 DATE=2016-02-03 14:13:30 PAGE 3
 LISTEN-BEISPIEL (MERGED-MODULES=NO)

OBJ	NAME	ADDRESS	LENGTH	ATTRIB	SLICE	TYPE	RESOLVED	IN MODULE/FROM	ELEMENT#	OM#	SLICE	SATIS
GM	SUBLLM				1					1		
SD	SUBLLM	00000000	00000016	AA.....	1							
VC	AUTO22						00000018	AUTO22		2	1	SLICE
VC	AUTO23						FFFFFFFF					UNRES
OM	AUTO22				1					2		
SD	AUTO22	00000018	00000006	AA.....	1							
						VC		SUBLLM			1	1 SLICE

--- END OF SECTION ---

BINDER V02.7A *COMMON LIST* LLM: COMPLEX2 DATE=2016-02-03 14:13:30 PAGE 4
 LISTEN-BEISPIEL (MERGED-MODULES=NO)

NAME	ADDRESS	LENGTH	ATTRIB	TYPE	IN MODULE	OM#	SLICE	SATIS
----	-----	-----	-----	----	-----	----	-----	-----

--- END OF SECTION ---

BINDER V02.7A *UNRESOLVED REFERENCES* LLM: COMPLEX2 DATE=2016-02-03 14:13:30 PAGE 5
 LISTEN-BEISPIEL (MERGED-MODULES=NO)
 VC AUTO23

END OF SECTION ---

BINDER V02.7A *UNRESOLVED LONG NAMES* LLM: COMPLEX2 DATE=2016-02-03 14:13:30 PAGE 6
 LISTEN-BEISPIEL (MERGED-MODULES=NO)
 INDEX NAME

--- END OF SECTION ---

BINDER V02.7A *SORTED SYMBOLS* LLM: COMPLEX2 DATE=2016-02-03 14:13:30 PAGE 7
 LISTEN-BEISPIEL (MERGED-MODULES=NO)

NAME	TYPE	IN MODULE	OM#	SLICE
----	----	-----	----	-----
AUTO22		SD AUTO22	2	1
SUBLLM		SD SUBLLM	1	1

--- END OF SECTION ---

BINDER V02.7A *DUPLICATE SYMBOLS* LLM: COMPLEX2 DATE=2016-02-03 14:13:30 PAGE 8
 LISTEN-BEISPIEL (MERGED-MODULES=NO)

NAME	TYPE	IN MODULE	OM#	SLICE	TYPE	IN MODULE	OM#	SLICE
----	----	-----	----	-----	----	-----	----	-----
NAME	TYPE	IN MODULE	OM#	SLICE	NAME	TYPE	IN MODULE	OM#
NAME	TYPE	IN MODULE	OM#	SLICE	NAME	TYPE	IN MODULE	OM#

--- END OF SECTION ---

Listenbeispiel: Einfügen von Modulen mit Hilfe einer SYSPLAMALT-Variable

Das folgende Beispiel zeigt, wie der INPUT INFORMATION-Abschnitt aussieht, wenn Module von einer SYSPLAMALT-Variable eingefügt sind.

Die aktuelle Bibliothek, aus der die Module eingefügt sind, kann mit Hilfe von FILE ID-Spalten in beiden Teilen des Abschnitts INPUT INFORMATION festgelegt werden.

```
/sysplamalt-bnd1=
'(bnd.bnd.pl.in.llmplamalt1,bnd.bnd.pl.in.llmplamalt2,bnd.bnd.pl.in.llmplamalt3)'
...
/start-binder
//start-llm-creation internal-name=llmplamalt
//include-modules library=sysplamalt-bnd1,element=module11
//include-modules element=module21
//include-modules element=module22
//include-modules library=bnd.pl.in,element=om5
//save-llm library=llmplamalt.pl,map=yes
//end
```

Erster Teil des Abschnitts INPUT INFORMATION

BINDER	V02.7A	*INPUT INFORMATION*	LLM:	LLMPLAMALT	DATE=2016-02-04 11:41:39	PAGE	9	
	NAME		TYPE	VERSION	DATE	FILE ID	BIND	ELEM#
ELEMENT:	MODULE11		R	001	1989-07-25	1.1	EXPL	1
MODULES								
1	MODULE11		R	001	1989-07-25	1.2	EXPL	2
ELEMENT:	MODULE21							
MODULES								
2	MODULE21		R	001	1989-07-25	1.2	EXPL	3
ELEMENT:	MODULE22							
MODULES								
3	MODULE22		R	@	2008-09-30	2	EXPL	4
ELEMENT:	OM5							
MODULES								
4	OM5							

--- END OF SECTION ---

FILE ID:

Kennzeichen für die Bibliothek, von der das Element eingefügt wurde.

Wenn das Kennzeichen das Format 'x.y' hat, verweist es auf eine der Bibliotheken, die in der SYSPLAMALT-Variable spezifiziert sind.

Es stellt die Verknüpfung mit dem zweiten Teil des Abschnitts bereit.

Zweiter Teil des Abschnitts INPUT INFORMATION

```

BINDER V02.7A  *LINKNAME  CONVERSION*      LLM:  LLMPLAMALT  DATE=2016-02-04 11:41:39 PAGE  10
FILE ID        LINKNAME      FILE TYPE      FILENAME
-----
      1                PLAM          LLMPLAMALT-BND1
      1.1              :IOSN:$BINDER.BND.BND.PL.IN.LLMPLAMALT1
      1.2              :IOSN:$BINDER.BND.BND.PL.IN.LLMPLAMALT2
      2                PLAM          :IOSN:$BINDER.BND.PL.IN

```

--- END OF SECTION ---

FILE ID:

Kennzeichen für die Inputquelle.

Es stellt die Verknüpfung mit dem ersten Teil des Abschnitts bereit.

Wenn das Kennzeichen das Format 'x.y' hat, verweist es auf eine der Bibliotheken, die in der SYSPLAMALT-Variable spezifiziert sind.

Die dazugehörige SYSPLAMALT-Variable kann einige Zeilen vorher anhand der FILE ID 'x' identifiziert werden.

5 BINDER-Lauf

Ein **BINDER-Lauf** ist eine Folge von Anweisungen, die nach dem Ladeaufruf für den BINDER beginnt und mit der END-Anweisung endet.

5.1 Aufrufen und Beenden des BINDER

Der BINDER wird mit folgendem Kommando geladen und gestartet:

START-BINDER
VERSION = <u>*STD</u> ,MONJV = <u>*NONE</u> / <filename 1..54 without-gen-vers> ,CPU-LIMIT = <u>*JOB-REST</u> / <integer 1..32767 seconds>

VERSION =

Produktversion des BINDER, die gestartet werden soll.
Derzeit wird nur die Angabe des Werts ***STD** unterstützt.

VERSION = *STD

Keine explizite Angabe der Produktversion. Die Produktversion wird folgendermaßen ausgewählt:

1. Die mit dem Kommando /SELECT-PRODUCT-VERSION vorgegebene Version.
2. Die höchste mit IMON installierte BINDER-Version.

MONJV =

Angabe einer Monitor-Jobvariablen zur Überwachung des BINDER-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.

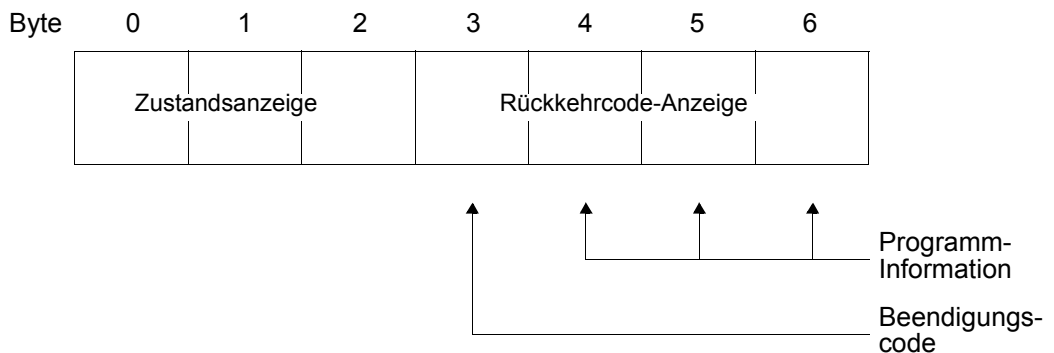
Der BINDER erwartet nach der Programmlademeldung die Eingabe von Anweisungen aus der Systemdatei SYSDTA (siehe [Seite 138ff](#)).

Die Anweisung END beendet den BINDER-Lauf.

5.2 Überwachung des BINDER-Laufs mit Jobvariablen

Für den Ablauf des BINDER kann der Benutzer eine programmüberwachende Jobvariable (JV) verwenden (siehe Handbuch „Jobvariablen“ [6]). Voraussetzung ist, dass das Softwareprodukt JV installiert ist.

Die programmüberwachende Jobvariable ist 7 Byte lang und hat folgenden Aufbau:



Die ersten 3 Byte der Jobvariablen (Byte 0-2) enthalten die **Zustandsanzeige**. Sie gibt den aktuellen Zustand des BINDER-Laufs wieder und kann folgende Werte annehmen:

- \$R BINDER läuft (START-BINDER)
- \$T BINDER-Lauf wurde erfolgreich beendet
- \$A BINDER-Lauf wurde fehlerhaft beendet

Die folgenden 4 Byte (Byte 3-6) enthalten die **Rückkehrcode-Anzeige**. Die Rückkehrcode-Anzeige besteht aus dem **Beendigungscode** (Byte 3) und der **Programminformation** (Byte 4-6).

Beendigungscode

- 0 der BINDER-Lauf wurde ohne Fehler beendet.
- 1 der BINDER-Lauf wurde ohne Fehler beendet. Warnungsmeldungen wurden ausgegeben.
- 2 der BINDER-Lauf wurde mit Fehlern beendet. Das LLM enthält Fehler oder ist unvollständig gebunden.
- 3 der BINDER-Lauf wurde wegen eines schwerwiegenden Fehlers beendet.

Programminformation

- 000 der BINDER-Lauf wurde ohne Fehler beendet. Das LLM wurde fehlerfrei gebunden.
- 001 der BINDER-Lauf wurde ohne Fehler beendet. Warnungsmeldungen wurden ausgegeben.
- 002 der BINDER-Lauf wurde ohne Fehler beendet. Einige unbefriedigte Externverweise wurden nicht aufgelöst.
- 003 der BINDER-Lauf wurde mit Fehlern beendet. Fehler bei der Syntaxprüfung der Anweisungen wurden erkannt.
- 004 der BINDER-Lauf wurde mit Fehlern beendet. Einige behebbare Fehler wurden erkannt.
- 005 der BINDER-Lauf wurde mit Fehlern beendet. Fehler in der Eingabe oder Anwenderfehler wurden erkannt. Kein LLM wurde gebunden bzw. das gebundene LLM darf nicht verwendet werden.
- 006 der BINDER-Lauf wurde mit Fehlern beendet (interner BINDER-Fehler). Kein LLM wurde gebunden bzw. das gebundene LLM darf nicht verwendet werden.

Die folgende Tabelle zeigt den Zusammenhang zwischen den verschiedenen Werten der Jobvariablen und den Fehlerklassen. Der Wert der Jobvariablen entspricht der *höchsten* Fehlerklasse, die während des BINDER-Laufs aufgetreten ist.

Zustands- anzeige	Beendigungs- code	Programm- information	Fehlerklasse
\$T	0	000	INFORMATION
\$T	1	001	WARNING
\$T	1	002	UNRESOLVED EXTERNS
\$A	2	003	SYNTAX ERROR
\$A	2	004	RECOVERABLE ERROR
\$A	3	005	FATAL ERROR
\$A	3	006	INTERNAL ERROR

6 Unterprogrammchnittstelle

6.1 Makroaufruf BINDER

Der BINDER wird mit dem Makroaufruf BINDER von einem Hauptprogramm als Unterprogramm aufgerufen. Die Anweisungen an den BINDER können dabei auf folgende Arten übergeben werden:

- Übergabe von SYSDTA
Der BINDER fordert nach dem Aufruf die Anweisungen nacheinander über SYSDTA an. Er verzweigt ins Hauptprogramm zurück, wenn die Anweisung END eingegeben wird.
- Übergabe vom Hauptprogramm
Bei jedem Aufruf des BINDER übergibt das Hauptprogramm *eine* Anweisung. Für jede Anweisung ist ein Aufruf notwendig.

Format und Operandenbeschreibung

Operation	Operanden
BINDER	[BNDSTMT = <u>NULL</u> / adr] [,MF = <u>S</u> / C / D / E / L / M] [,PARAM = opadr / (r)] [,PREFIX = <u>P</u> / x] [,MACID = <u>BND</u> / yyy]

BNDSTMT Legt fest, ob die Anweisungen von SYSDTA oder vom Hauptprogramm an den BINDER übergeben werden.

=NULL Die Anweisungen werden von SYSDTA übergeben.

=adr Die Anweisungen werden vom Hauptprogramm übergeben. „adr“ bezeichnet die symbolische Adresse (Name) eines Parameterbereichs, in dem jeweils eine Anweisung übergeben wird (siehe nächste Seite).

MF	Steuert die Form der Makroauflösung (siehe Handbuch „Makroaufrufe an den Ablaufteil“ [7]).
=S	S-Form
=C	C-Form (nur CSECT)
=D	D-Form (nur DSECT)
=E	E-Form (nur Befehlscode) Erzeugt die notwendigen Befehle für die Ausführung des Makros.
=L	L-Form (nur Daten)
=M	M-Form
PARAM	Nur für E-Form und M-Form.
=opadr	Operandenliste, deren Adresse mit „opadr“ anzugeben ist, und die zuvor mit dem Steueroperanden MF=L erzeugt wurde.
=(r)	Operandenliste, deren Adresse im Register (r) anzugeben ist, und die zuvor mit dem Steueroperanden MF=L erzeugt wurde.
PREFIX=x	Steuert die Generierung der Namen (nur für C-Form, D-FORM und M-Form). Für „x“ ist <i>ein</i> Buchstabe anzugeben. Er wird als erster Buchstabe aller symbolischen Namen verwendet. Die restlichen Zeichen des Namens werden nicht geändert. Standardwert ist „P“.
MACID=yyy	Steuert die Generierung der Namen (nur für C-Form und M-Form). Für „yyy“ sind <i>drei</i> Buchstaben anzugeben. Sie bestimmen das zweite bis vierte Zeichen der symbolischen Namen. Standardwert ist „BND“.

Hinweise

- Der *Makro* BINDER wird in der Makrobibliothek \$.SYSLIB.BINDER.027 ausgeliefert.
- Das *Modul* BINDER wird in der Programmbibliothek \$.SYSLNK.BINDER.027 ausgeliefert. Da das Modul BINDER in dieser Bibliothek gewartet wird, ist es vorteilhafter, im Hauptprogramm dieses Modul dynamisch mit dem Makro BIND nachzuladen, da dann das aufrufende Programm immer das aktuelle Modul BINDER benutzen kann (siehe nachfolgende Beispiele). Die S-Form des Makroaufrufs BINDER sollte nicht verwendet werden, da sie eine V-Konstante V(BINDER) generiert.
- Werden die Anweisungen vom aufrufenden Programm übergeben, muss immer dieselbe Parameterliste benutzt werden. Der BINDER speichert Informationen in der Parameterliste, die für den nächsten Aufruf benutzt werden. Die Parameterliste sollte daher beim ersten Aufruf des BINDER mit MF=L angelegt und bei den folgenden Aufrufen mit MF=M geändert werden (siehe Beispiel 2 [Seite 182](#)).
- Der BINDER benutzt Felder in der Parameterliste für interne Informationen. Die Werte in der Parameterliste sollten daher nicht direkt, sondern implizit durch den BINDER-Makro mit MF=M geändert werden.

Aufbau des Parameterbereichs

Der Parameterbereich für die Übergabe einer Anweisung hat folgenden Aufbau:

Byte	Länge	Bedeutung
0-1	2	Länge der Anweisung einschließlich 4 Byte Satzlängenfeld (n+4)
2-3	2	Reserviert (wird ignoriert)
4-n	beliebig	Anweisung der Länge n. Die Anweisung wird ohne Syntaxprüfung im angegebenen Format an SDF übergeben. Der Benutzer muss daher selbst entscheiden, ob Kleinbuchstaben zulässig sind oder nicht.

Registerkonventionen

Beim Aufruf des BINDER als Unterprogramm sind folgende Registerkonventionen zu beachten:

- Register 1: Adresse der Parameterliste des BINDER-Makros
- Register 14: Rücksprungadresse in das Hauptprogramm
- Register 15: Adresse der Einsprungstelle in den BINDER

Rückinformation und Fehleranzeigen

Der BINDER übergibt bei jedem Makroaufruf einen **Returncode (RC)**, der Informationen über die Ausführung des Makros enthält. Der Returncode wird im *niedrigstwertigen* Byte des Feldes MAINCODE im Standardheader übergeben (siehe Handbuch „Makroaufrufe an den Ablaufteil“ [7]). Die Werte bezeichnen Sedezimalkonstanten. Der übergebene Returncode entspricht der höchsten Fehlerklasse, die beim Aufruf des BINDER aufgetreten ist.

Zusätzlich zeigt Subcode1 an, ob der Makro BINDER normal oder abnormal beendet wurde.

Subcode2 enthält das maximal erlaubte Fehlergewicht, falls die abnormale Beendigung durch das Überschreiten dieses Fehlergewichts verursacht wurde. Andernfalls enthält Subcode2 den Wert X'FF'.

Die folgenden Tabellen zeigen die Werte und Bedeutungen des Returncodes und des Subcode1. Zusätzlich sind noch die zugehörigen symbolischen Namen angegeben. Der Benutzer kann mit dem Operanden PREFIX=x das erste Zeichen und mit dem Operanden MACID=yyy das zweite bis vierte Zeichen in den symbolischen Namen festlegen. Standardwerte sind PREFIX=P und MACID=BND.

Returncode	symbolischer Name xyyy...	Bedeutung
X'00'	OK	Kein Fehler
X'01'	INFO	Fehler der Fehlerklasse INFORMATION
X'02'	WARN	Fehler der Fehlerklasse WARNING
X'03'	UNRE	Fehler der Fehlerklasse UNRESOLVED EXTERNS
X'04'	SYNT	Fehler der Fehlerklasse SYNTAX ERROR
X'05'	RECO	Fehler der Fehlerklasse RECOVERABLE ERROR
X'06'	FATA	Fehler der Fehlerklasse FATAL ERROR
X'07'	INTE	Fehler der Fehlerklasse INTERNAL ERROR

Subcode1	symbolischer Name xyyy...	Bedeutung
X'FF'	NORM	Normale Beendigung
X'40'	ABN	Abnormale Beendigung

6.2 Beispiele

Beispiel 1

Eine Routine BNDCALL1 lädt den BINDER mit dem Makroaufruf BIND dynamisch aus der Programmbibliothek \$.SYSLNK.BINDER.027. Mit dem Makroaufruf BINDER wird der BINDER aufgerufen und fordert die BINDER-Anweisungen über SYSDTA an. Die Routine BNDCALL1 wird von einem Hauptprogramm PROG1 als Unterprogramm aufgerufen.

Ausdruck des Quellprogramms von BNDCALL1

```

BNDCALL1 CSECT
        USING BNDCALL1,15
        STM 0,15,SAVEREG ----- (1)
        DROP 15
        LR 10,15
        USING BNDCALL1,10
        L 1,BINDER@ ----- (2)
        LTR 1,1
        BNZ NOTICALL ----- (3)
        BIND MF=E,PARAM=BINDPL ----- (4)
        LA 2,BINDPL
        USING BINDDS,2 ----- (5)
        CLI XBINSR2,XBINPART ----- (6)
        BNL BINDERR ----- (7)
NOTICALL DS OH
        USING BNDDS,1 ----- (8)
        LA 1,BNDPL ----- (9)
        L 15,BINDER@ ----- (10)
        BALR 14,15 ----- (11)
        CLI YBNDMRET+1,YBNDSYNT ----- (12)
        BNL BNDERROR ----- (13)
RETURN  DS OH
        LM 0,15,SAVEREG ----- (14)
        BR 14 ----- (15)
BINDERR DS OH
        B RETURN
BNDERROR DS OH
        B RETURN
SAVEREG DS 16F
BINDER@ DC A(0)
BNDPL  BINDER MF=L ----- (16)
MODNAM DC CL32'BINDER '
MODLIB DC CL54'$.SYSLNK.BINDER.027 '
BINDPL BIND MF=L,SYMBOL@=MODNAM,LIBNAM@=MODLIB,SYMBLAD=BINDER@ ----- (17)
        LTORG
BNDDS  BINDER MF=D,PREFIX=Y ----- (18)

```

```
BINDDS  BIND  MF=D,PREFIX=X ----- (19)
        END
```

Ausdruck des Quellprogramms von PROGI

```
PROGI   START
        BALR  3,0
        USING *,3
        L     15,=V(BNDCALL1) ----- (20)
        BALR  14,15
        WROUT AUS,FEHLER ----- (21)
FEHLER  TERM
AUS     DC    Y(AUSE-AUS)
        DS    CL3
        DC    C'BINDER CALL TERMINATED'
AUSE    EQU   *
        END
```

- (1) Alle Register werden im Sicherstellungsbereich von BNDCALL1 sichergestellt.
- (2) Das Feld BINDER@ wird überprüft. In ihm übergibt der BIND-Makro die Startadresse des nachgeladenen Moduls BINDER (siehe (4)).
- (3) Falls eine Adresse im Feld BINDER@ übergeben wurde, ist das Modul BINDER bereits geladen und der BIND-Aufruf wird übergangen.
- (4) Der Makro BIND wird in seiner E-Form aufgerufen. An dieser Stelle im Programm wird daher nur der Befehlssteil erzeugt. Die zugehörige Parameterliste wird an der symbolischen Adresse BINDPL (siehe (17)) durch einen BIND-Aufruf mit MF=L angelegt. Die dort angegebenen Operandenwerte veranlassen den BIND-Makro bei der Programmausführung,
 - das Modul BINDER (SYMBOL@=MODNAM) aus der Bibliothek \$.SYSLNK.BINDER.027 (LIBNAM@=MODLIB) nachzuladen,
 - die Startadresse des Moduls BINDER im Feld BINDER@ zu übergeben (SYMBLAD=BINDER@),
 - nach dem Laden des Moduls BINDER in der aufrufenden Routine BNDCALL1 mit dem dem Makroaufruf BIND folgenden Befehl fortzusetzen (Standardwert BRANCH=NO).
- (5) Register 2 wird dem Assembler als Basisadressregister zur Adressierung der DSECT für die Parameterliste des BIND-Makros zugewiesen, die an der symbolischen Adresse BINDDS durch einen BIND-Aufruf mit MF=D erzeugt wird.

- (6) Nach der Ausführung des BIND-Makros wird das Feld XBINSR2 des Standardheaders, das den SUBCODE2 enthält, mit dem SUBCODE2 XBINPART des Standardheaders, der eine fehlerfreie Ausführung des Makros BIND festlegt, verglichen. Die Namen XBINSR2 und XBINPART stammen aus der DSECT, die unter der symbolischen Adresse BINDDS durch einen BIND-Aufruf mit MF=D und PREFIX=X erzeugt wurde (siehe (17)). Diese DSECT beschreibt den Aufbau der Parameterliste des BIND-Makros. Die symbolischen Namen der DSECT können zur Adressierung innerhalb der Parameterliste verwendet werden, nachdem das zugeordnete Basisadressregister (hier Register 2) mit der Anfangsadresse der Parameterliste (hier BINDPL) geladen worden ist.
- (7) Verzweigung zum Fehlerausgang BINDERR, falls der BIND-Makro fehlerhaft oder nicht ausgeführt wird.
- (8) Register 1 wird dem Assembler als Basisadressregister zur Adressierung der DSECT für die Parameterliste des BINDER-Makros zugewiesen, die an der symbolischen Adresse BNDDS (siehe (18)) durch einen BINDER-Aufruf mit MF=D erzeugt wird.
- (9) Register 1 wird mit der Adresse der Parameterliste des BINDER-Makros geladen. Die Parameterliste wird an der symbolischen Adresse BNDPL (siehe (16)) durch einen BINDER-Aufruf mit MF=L erzeugt. Der Standardwert BNDSTMT=NULL legt fest, dass die BINDER-Anweisungen vom BINDER über SYSDTA angefordert werden.
- (10) Register 15 wird mit der Startadresse des Moduls BINDER geladen, die vom Makro BIND im Feld BINDER@ übergeben wurde.
- (11) Aufruf des Moduls BINDER, der die BINDER-Anweisungen über SYSDTA anfordert. Nach Eingabe der Anweisung END wird der BINDER-Lauf beendet und zurück in die Routine BNDCALL1 verzweigt.
- (12) Nach der Ausführung des BINDER-Makros wird das Feld YBNDMRET+1 des Standardheaders, das das niedrigstwertige Byte des MAINCODE bezeichnet, mit dem Returncode YBNDSYNT verglichen, der die Fehlerklasse SYNTAX ERROR angibt. Die Namen YBNDMRET und YBNDSYNT stammen aus der DSECT, die unter der symbolischen Adresse BNDDS durch einen BINDER-Aufruf mit MF=D und PREFIX=Y erzeugt wurde (siehe (16)). Diese DSECT beschreibt den Aufbau der Parameterliste des BINDER-Makros. Die symbolischen Namen der DSECT können zur Adressierung innerhalb der Parameterliste verwendet werden, nachdem das zugeordnete Basisadressregister (hier Register 1) mit der Anfangsadresse der Parameterliste (hier BNDPL) geladen worden ist.
- (13) Verzweigung zum Fehlerausgang BNDERROR, falls bei Ausführung des BINDER-Makros Fehler ab der Fehlerklasse SYNTAX ERROR aufgetreten sind.
- (14) Laden aller sichergestellten Register mit den ursprünglichen Inhalten.

- (15) Rücksprung in das Anwenderprogramm, das die Routine BNDCALL1 aufgerufen hat.
- (16) Mit dem Aufruf des Makros BINDER in der L-Form wird die Parameterliste für den Aufruf des Moduls BINDER (siehe (9)) angelegt und mit Werten versorgt.
- (17) Mit der L-Form des Makros BIND wird die Parameterliste für den Makro BIND angelegt (siehe (4)).
- (18) Der Makroaufruf BINDER mit MF=D erzeugt eine DSECT, die den Aufbau der Parameterliste des BINDER-Makros beschreibt. Der Operand PREFIX=Y bewirkt, dass alle symbolischen Namen in dieser DSECT (Feldnamen und Equates) mit dem Buchstaben Y beginnen.
- (19) Der Makroaufruf BIND mit MF=D erzeugt eine DSECT, die den Aufbau der Parameterliste des BIND-Makros beschreibt. Der Operand PREFIX=X bewirkt, dass alle symbolischen Namen in dieser DSECT (Feldnamen und Equates) mit dem Buchstaben X beginnen.
- (20) Die Routine BNDCALL1 wird vom Anwenderprogramm PROG1 als Unterprogramm aufgerufen.
- (21) Eine Meldung nach SYSOUT informiert darüber, dass die Routine BNDCALL1 beendet wurde.

Ablaufprotokoll

```

/set-file-link link-name=altlib,file-name=$.syslib.binder.027 _____ (1)
/start-assembly
% BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2C00' OF '2002-03-24' LOADED
% BLS0552 COPYRIGHT (C) 2016 FUJITSU TECHNOLOGY SOLUTIONS GMBH.
  ALL RIGHTS RESERVED
% ASS6010 V01.2C00 OF BS2000 ASSEMBH  READY
///compile source=src.bndcall1,macro-library=*link(link-name=altlib),-
///module-library=bndlib _____ (2)
% ASS6011 ASSEMBLY TIME: 1458 MSEC
% ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
% ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
% ASS6006 LISTING GENERATOR TIME: 867 MSEC
///compile source=src.prog1,module-library=*omf _____ (3)
% ASS6011 ASSEMBLY TIME: 697 MSEC
% ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
% ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
% ASS6006 LISTING GENERATOR TIME: 260 MSEC
///end
% ASS6012 END OF ASSEMBH

```

```

/set-file-link link-name=b1slib01,file-name=bndlib _____ (4)
/start-program from-file=*module(library=*omf,element=prog1, -
/          run-mode=advanced(alternate-libraries=yes)) _____ (5)
% BLS0517 MODULE 'PROG1' LOADED
%//start-llm-creation internal-name=llm1 _____ (6)
%//include-modules library=bndlib,element=(mod1,mod2)
%//save-llm library=bndlib
% BND1501 LLM FORMAT : '1'
%//end
% BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
BINDER CALL TERMINATED _____ (7)

```

- (1) Dem Assembler wird als ALTLIB die Makrobibliothek \$.SYSLIB.BINDER.027 zugewiesen, die den Makro BINDER enthält.
- (2) Die Routine BNDCALL1 wird übersetzt und das Bindemodul BNDCALL1 wird in der Programmbibliothek BNDLIB abgespeichert.
- (3) Das Anwenderprogramm PROG1 wird übersetzt.
- (4) Die Programmbibliothek BNDLIB, die das Modul BNDCALL1 enthält, wird als alternative Bibliothek zugewiesen.
- (5) Der DBL wird aufgerufen, um das Modul PROG1 zu laden und zu starten. Das Modul PROG1 wird aus der EAM-Bindemodulbibliothek geholt. Externverweise werden mit Symbolen aus der alternativen Bibliothek befriedigt.
- (6) Das Modul BINDER wird von BNDCALL1 aufgerufen und fordert die BINDER-Anweisungen über SYSDTA an. Die Anweisung END beendet den BINDER-Lauf.
- (7) Das Programm PROG1 gibt eine Meldung aus, dass die Routine BNDCALL1 beendet wurde.

Beispiel 2

Eine Routine BNDCALL2 lädt den BINDER mit dem Makroaufruf BIND dynamisch aus der Programmbibliothek \$.SYSLNK.BINDER.027. Mit dem Makroaufruf BINDER wird der BINDER aufgerufen. Die ersten beiden BINDER-Anweisungen werden von der Routine BNDCALL2 übergeben. Anschließend fordert der BINDER weitere BINDER-Anweisungen über SYSDTA an. Die Routine BNDCALL2 wird von einem Hauptprogramm PROG2 als Unterprogramm aufgerufen.

Ausdruck des Quellprogramms von BNDCALL2

```

BNDCALL2 CSECT
        USING BNDCALL2,15
        STM 0,15,SAVEREG _____ (1)
        DROP 15
        LR 10,15
        USING BNDCALL2,10
        L 1,BINDER@ _____ (2)
        LTR 1,1
        BNZ NOT2CALL _____ (3)
        BIND MF=E,PARAM=BINDPL _____ (4)
        LA 2,BINDPL
        USING BINDDS,2 _____ (5)
        CLI XBINSR2,XBINPART _____ (6)
        BNL BINDERR _____ (7)
NOT2CALL DS 0H
        LA 1,BNDPL _____ (8)
        USING BNDDS,1 _____ (9)
        L 15,BINDER@ _____ (10)
        BALR 14,15 _____ (11)
        CLI YBNDMRET+1,YBNDSYNT _____ (12)
        BL CAL2STMT _____ (13)
        CLI YBNDMRET+1,YBNDFATA _____ (14)
        BNL RETURN _____ (15)
CAL2STMT BINDER MF=M,BNDSTMT=STMT2LG,PREFIX=Y _____ (16)
        BALR 14,15 _____ (17)
        BINDER MF=M,BNDSTMT=NULL,PREFIX=Y _____ (18)
        BALR 14,15 _____ (19)
        CLI YBNDMRET+1,YBNDSYNT _____ (20)
        BNL BNDERROR _____ (21)
RETURN DS 0H
        LM 0,15,SAVEREG _____ (22)
        BR 14 _____ (23)
BINDERR DS 0H
        B RETURN
BNDERROR DS 0H
        B RETURN
SAVEREG DS 16F

```

```

BINDER@ DC      A(0)
STMT1LG DC      H'40' _____ (24)
STMT1UN DC      H'0'
STMT1   DC      CL36'START-LLM-CREATION INTERNAL-NAME=LLM'
STMT2LG DC      H'56' _____ (25)
STMT2UN DC      H'0'
STMT2   DC      CL52'INCLUDE-MODULES LIBRARY=BNDLIB,ELEMENT=(MOD1,MOD2)'
BNDPL   BINDER MF=L,BNDSTMT=STMT1LG _____ (26)
MODNAM  DC      CL32'BINDER '
MODLIB  DC      CL54'$.SYSLNK.BINDER.027 '
BINDPL  BIND    MF=L,SYMBOL@=MODNAM,LIBNAM@=MODLIB,SYMBLAD=BINDER@ _____ (27)
          LTOrg
BNDDS   BINDER MF=D,PREFIX=Y _____ (28)
BINDDS  BIND    MF=D,PREFIX=X _____ (29)
          END

```

Ausdruck des Quellprogramms von PROG2

```

PROG2   START
          BALR  3,0
          USING *,3
          L     15,=V(BNDCALL2) _____ (30)
          BALR  14,15
          WROUT AUS,FEHLER _____ (31)
FEHLER  TERM
AUS     DC      Y(AUSE-AUS)
          DS     CL3
          DC     C'BINDER CALL TERMINATED'
AUSE    EQU     *
          END

```

- (1) Alle Register werden im Sicherstellungsbereich von BNDCALL2 sichergestellt.
- (2) Das Feld BINDER@ wird überprüft. In ihm übergibt der BIND-Makro die Startadresse des nachgeladenen Moduls BINDER (siehe 04).
- (3) Falls eine Adresse im Feld BINDER@ übergeben wurde, ist das Modul BINDER bereits geladen und der BIND-Aufruf wird übergangen.

- (4) Der Makro BIND wird in seiner E-Form aufgerufen. An dieser Stelle im Programm wird daher nur der Befehlssteil erzeugt. Die zugehörige Parameterliste wird an der symbolischen Adresse BINDPL (siehe (27)) durch einen BIND-Aufruf mit MF=L angelegt. Die dort angegebenen Operandenwerte veranlassen den BIND-Makro bei der Programmausführung:
 - Das Modul BINDER (SYMBOL@=MODNAM) aus der Bibliothek \$.SYSLNK.BINDER.027 (LIBNAM@=MODLIB) nachzuladen,
 - die Startadresse des Moduls BINDER im Feld BINDER@ zu übergeben (SYMBLAD=BINDER@),
 - nach dem Laden des Moduls BINDER in der aufrufenden Routine BNDCALL2 mit dem dem Makroaufruf BIND folgenden Befehl fortzusetzen (Standardwert BRANCH=NO).
- (5) Register 2 wird dem Assembler als Basisadressregister zur Adressierung der DSECT für die Parameterliste des BIND-Makros zugewiesen, die an der symbolischen Adresse BINDDS durch einen BIND-Aufruf mit MF=D erzeugt wird.
- (6) Nach der Ausführung des BIND-Makros wird das Feld XBINSR2 des Standardheaders, das den SUBCODE2 enthält, mit dem SUBCODE2 XBINPART des Standardheaders, der eine fehlerfreie Ausführung des Makros BIND festlegt, verglichen. Die Namen XBINSR2 und XBINPART stammen aus der DSECT, die unter der symbolischen Adresse BINDDS durch einen BIND-Aufruf mit MF=D und PREFIX=X erzeugt wurde (siehe (27)). Diese DSECT beschreibt den Aufbau der Parameterliste des BIND-Makros. Die symbolischen Namen der DSECT können zur Adressierung innerhalb der Parameterliste verwendet werden, nachdem das zugeordnete Basisadressregister (hier Register 2) mit der Anfangsadresse der Parameterliste (hier BINDPL) geladen worden ist.
- (7) Verzweigung zum Fehlerausgang BINDERR, falls der BIND-Makro fehlerhaft oder nicht ausgeführt wird.
- (8) Register 1 wird mit der Adresse der Parameterliste des BINDER-Makros geladen. Die Parameterliste wird an der symbolischen Adresse BNDPL (siehe (26)) durch einen BINDER-Aufruf mit MF=L erzeugt. Der Wert STMT1LG des Operanden BNDSTMT legt fest, dass die BINDER-Anweisung START-LLM-CREATION übergeben wird, die im Parameterbereich ab der Adresse STMT1LG aufgebaut ist.
- (9) Register 1 wird dem Assembler als Basisadressregister zur Adressierung der DSECT für die Parameterliste des BINDER-Makros zugewiesen, die an der symbolischen Adresse BNDDS durch einen BINDER-Aufruf mit MF=D erzeugt wird.
- (10) Register 15 wird mit der Startadresse des Moduls BINDER geladen, die vom Makro BIND im Feld BINDER@ übergeben wurde.
- (11) Aufruf des Moduls BINDER, der die erste BINDER-Anweisung aus dem Parameterbereich ab der Adresse STMT1LG übernimmt.

- (12) Nach der Ausführung des BINDER-Makros wird das Feld YBNDMRET+1 des Standardheaders, das das niedrigstwertige Byte des MAINCODE bezeichnet, mit dem Returncode YBNDSYNT verglichen, der die Fehlerklasse SYNTAX ERROR angibt. Die Namen YBNDMRET und YBNDSYNT stammen aus der DSECT, die unter der symbolischen Adresse BNDDS durch einen BINDER-Aufruf mit MF=D und PREFIX=Y erzeugt wurde (siehe (26)). Diese DSECT beschreibt den Aufbau der Parameterliste des BINDER-Makros. Die symbolischen Namen der DSECT können zur Adressierung innerhalb der Parameterliste verwendet werden, nachdem das zugeordnete Basisadressregister (hier Register 1) mit der Anfangsadresse der Parameterliste (hier BNDPL) geladen worden ist.
- (13) Verzweigung zum nächsten BINDER-Aufruf, falls bei Ausführung des BINDER-Makros keine Fehler ab der Fehlerklasse SYNTAX ERROR aufgetreten sind.
- (14) Das Feld YBNDMRET+1 des Standardheaders, das das niedrigstwertige Byte des MAINCODE bezeichnet, wird verglichen mit dem Returncode YBNDFATA, der die Fehlerklasse FATAL ERROR angibt.
- (15) Falls bei der Ausführung des BINDER-Makros Fehler ab der Fehlerklasse FATAL ERROR aufgetreten sind, verzweigt die Routine BNDCALL2 in das Anwenderprogramm PROG2, von dem sie aufgerufen wurde.
- (16) Der Makroaufruf BINDER mit MF=M modifiziert die Parameterliste, die mit dem vorhergehenden Makroaufruf BINDER mit MF=L angelegt wurde. Der Wert STMT2LG des Operanden BNDSTMT legt fest, dass die BINDER-Anweisung INCLUDE-MODULES übernommen wird, die im Parameterbereich ab der Adresse STMT2LG aufgebaut ist.
- (17) Aufruf des Moduls BINDER, der die BINDER-Anweisung aus dem Parameterbereich ab der Adresse STMT2LG übernimmt.
- (18) Der Makroaufruf BINDER mit MF=M modifiziert die Parameterliste, die mit MF=L angelegt wurde. Der Wert NULL des Operanden BNDSTMT legt fest, dass die folgenden BINDER-Anweisungen über SYSDTA angefordert werden.
- (19) Aufruf des Moduls BINDER, der die BINDER-Anweisungen über SYSDTA anfordert. Nach Eingabe der Anweisung END wird der BINDER-Lauf beendet und zurück in die Routine BNDCALL2 verzweigt.
- (20) Nach der Ausführung des BINDER-Makros wird das Feld YBNDMRET+1 des Standardheaders, das das niedrigstwertige Byte des MAINCODE bezeichnet, mit dem Returncode YBNDSYNT verglichen, der die Fehlerklasse SYNTAX ERROR angibt.
- (21) Verzweigung zum Fehlerausgang BINDERERR, falls bei Ausführung des BINDER-Makros Fehler ab der Fehlerklasse SYNTAX ERROR aufgetreten sind.
- (22) Laden aller sichergestellten Register mit den ursprünglichen Inhalten.

- (23) Rücksprung in das Anwenderprogramm, das die Routine BNDCALL2 aufgerufen hat.
- (24) Parameterbereich, in dem die Anweisung START-LLM-CREATION aufgebaut ist.
- (25) Parameterbereich, in dem die Anweisung INCLUDE-MODULES aufgebaut ist.
- (26) Der Makroaufruf BINDER mit MF=L erzeugt die Parameterliste für den Aufruf des Moduls BINDER (siehe (11)) und versorgt sie mit Werten, z.B. die auszuführende Anweisung.
- (27) Der Makroaufruf BIND mit MF=L erzeugt die Parameterliste für den Makro BIND (siehe (4)).
- (28) Der Makroaufruf BINDER mit MF=D erzeugt eine DSECT, die den Aufbau der Parameterliste des BINDER-Makros beschreibt. Der Operand PREFIX=Y bewirkt, dass alle symbolischen Namen in dieser DSECT (Feldnamen und Equates) mit dem Buchstaben Y beginnen.
- (29) Der Makroaufruf BIND mit MF=D erzeugt eine DSECT, die den Aufbau der Parameterliste des BIND-Makros beschreibt. Der Operand PREFIX=X bewirkt, dass alle symbolischen Namen in dieser DSECT (Feldnamen und Equates) mit dem Buchstaben X beginnen.
- (30) Die Routine BNDCALL2 wird vom Anwenderprogramm PROG2 als Unterprogramm aufgerufen.
- (31) Eine Meldung nach SYSOUT informiert darüber, dass die Routine BNDCALL2 beendet wurde.

Ablaufprotokoll

```

/set-file-link link-name=altlib,file-name=$.syslib.binder.027 _____ (1)
/start-assembly
% BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2C00' OF '2002-03-24' LOADED
% BLS0552 COPYRIGHT (C) 2016 FUJITSU TECHNOLOGY SOLUTIONS GMBH.
  ALL RIGHTS RESERVED
% ASS6010 V01.2C00 OF BS2000 ASSEMBH  READY
///compile source=src.bndcall2,macro-library=*link(link-name=altlib),-
///  module-library=bndlib _____ (2)
% ASS6011 ASSEMBLY TIME: 1486 MSEC
% ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
% ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
% ASS6006 LISTING GENERATOR TIME: 855 MSEC
///compile source=src.prog2,module-library=*omf _____ (3)
% ASS6011 ASSEMBLY TIME: 688 MSEC
% ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
% ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
% ASS6006 LISTING GENERATOR TIME: 255 MSEC
///end

```

```
% ASS6012 END OF ASSEMBH
/set-file-link link-name=b1slib01,file-name=bndlib _____ (4)
/start-program from-file=*module(library=*omf,element=prog2,-
/
run-mode=advanced(alternate-libraries=yes)) _____ (5)
% BLS0517 MODULE 'PROG2' LOADED
///include-modules element=mod3,type=(1,r) _____ (6)
///show-map help-information=no,global-information=no, - _____ (7)
///physical-structure=no,program-map=no,unresolved-list=no,-
///input-information=no,output=*by-show-file
% BND1601 'FILE' MACRO PERFORMED ON 'BNDMAP.2016-02-17138.173535.2AB0'
```

```
15000000BINDER V02.7A *LOGICAL STRUCTURE* LLM: X (8)
15000001
15000002SLICE TYPE HSI MMODE LEVEL STR# NAME
15000003-----
15100000 1 LLM /7500 TU4K 0 1 LLMX
15100001 1 GM /7500 TU4K 1 2 MOD1
15100002 1 GM /7500 TU4K 1 2 MOD2
15100003 1 GM /7500 TU4K 1 2 MOD3
15900000
17000000BINDER V02.0B *SCOPE PATH INFORMATION* LLM: X
17000001
17000002PATH STR# PATHNAME
17000003-----
17900000

% SH00301 WARNING: END OF FILE REACHED
e I*SOF+ 1( 1)
```

```
save-llm library=bndlib _____ (9)
///end _____ (10)
% BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
BINDER CALL TERMINATED _____ (11)
```

- (1) Dem Assembler wird als ALTLIB die Makrobibliothek \$.SYSLIB.BINDER.027 zugewiesen, die den Makro BINDER enthält.
- (2) Die Routine BNDCALL2 wird übersetzt und das Bindemodul BNDCALL2 in der Programmbibliothek BNDLIB abgespeichert.
- (3) Das Anwenderprogramm PROG2 wird übersetzt und das Bindemodul PROG2 wird in der EAM-Bindemoduldatei abgelegt.
- (4) Die Programmbibliothek BNDLIB, die das Bindemodul BNDCALL2 enthält, wird als alternative Bibliothek zugewiesen.

- (5) Der DBL wird aufgerufen, um das Modul PROG2 zu laden und zu starten. Das Modul PROG2 wird aus der EAM-Bindemodulbibliothek geholt. Nachdem die ersten beiden Anweisungen START-LLM-CREATION und INCLUDE-MODULES abgearbeitet wurden, die von BNDCALL2 übergeben werden, fordert das Modul BINDER weitere BINDER-Anweisungen über SYSDTA an.
- (6) Die Anweisung INCLUDE-MODULES wird über SYSDTA eingegeben. Sie holt das Bindemodul MOD3 aus der aktuellen Bibliothek BLSLIB01 und fügt es in das LLM ein.
- (7) Die Anweisung SHOW-MAP wird über SYSDTA eingegeben. Es soll nur die Übersicht über die logische Struktur des LLM ausgegeben werden.
- (8) Liste über die logische Struktur des erzeugten LLM. In den ersten acht Spalten sind die ISAM-Schlüssel zu erkennen.
- (9) Die Anweisung SAVE-LLM speichert das erzeugte LLM in der Programmbibliothek BNDLIB.
- (10) Die Anweisung END beendet den BINDER-Lauf.
- (11) Das Programm PROG2 gibt eine Meldung aus, dass die Routine BNDCALL2 beendet wurde.

7 BINDER-Anweisungen

In diesem Kapitel sind alle Anweisungen für den BINDER beschrieben.

7.1 Einteilung der Anweisungen nach Funktionen

Die Anweisungen für den BINDER sind entsprechend ihrer Funktion in folgende Gruppen eingeteilt:

Erzeugen, Ändern und Speichern eines LLM

START-LLM-CREATION	Erzeugt ein aktuelles LLM im BINDER-Arbeitsbereich.
START-LLM-UPDATE	Ändert ein LLM, das in einer Programmbibliothek gespeichert ist.
SAVE-LLM	Speichert das aktuelle LLM aus dem BINDER-Arbeitsbereich in einer Programmbibliothek.

Einfügen, Entfernen und Ersetzen von Modulen

INCLUDE-MODULES	Fügt ein oder mehrere Module in das aktuelle LLM ein.
REMOVE-MODULES	Entfernt ein oder mehrere Module im aktuellen LLM.
REPLACE-MODULES	Ersetzt ein oder mehrere Module im aktuellen LLM durch neue Module.

Mischen von Modulen

MERGE-MODULES	Mischen aller Module eines (Sub-)LLMs zu einem LLM, das nur noch ein Großmodul mit einer einzigen CSECT enthält.
---------------	--

Erzeugen der logischen Struktur eines LLM

BEGIN-SUB-LLM-STATEMENTS	Legt den Beginn eines Sub-LLM innerhalb eines LLM oder eines Sub-LLM fest.
END-SUB-LLM-STATEMENTS	Legt das Ende eines Sub-LLM innerhalb eines LLM oder Sub-LLM fest.

Erzeugen der physischen Struktur eines LLM

SET-USER-SLICE-POSITION	Legt die Lage einer Slice in der physischen Struktur eines LLM fest.
-------------------------	--

Ändern der Merkmale von LLMs und Modulen

MODIFY-LLM-ATTRIBUTES	Ändert die Merkmale eines LLMs.
MODIFY-MODULE-ATTRIBUTES	Ändert Merkmale der Module im aktuellen LLM und kann z.B. die logische Struktur des LLMs modifizieren.

Befriedigen von Externverweisen durch Autolink

RESOLVE-BY-AUTOLINK	Fügt automatisch Module in das aktuelle LLM ein, die unbefriedigte Externverweise befriedigen.
SET-EXTERN-RESOLUTION	Vereinbart die Behandlung unbefriedigter Externverweise, die nicht befriedigt werden können.

Behandlung von Symbolen

RENAME-SYMBOLS	Ersetzt die Namen von Symbolen im aktuellen LLM durch neue Namen.
MODIFY-SYMBOL-ATTRIBUTES	Ändert die Attribute von CSECTs und COMMON-Bereichen im aktuellen LLM.
MODIFY-SYMBOL-TYPE	Ändert den Typ von Symbolen (EXTRN, WXTRN und VCON).
MODIFY-SYMBOL-VISIBILITY	Legt fest, in welchem Umfang Programmdefinitionen (CSECTs) und Einsprungstellen (ENTRYs) im aktuellen LLM sichtbar bleiben oder maskiert werden.

Anzeigefunktionen

SHOW-DEFAULTS	Zeigt globale Standardwerte für einen BINDER-Lauf an.
SHOW-LIBRARY-ELEMENTS	Anzeigen und Prüfen von Bibliothekselementen.
SHOW-SYMBOL-INFORMATION	Zeigt ausgewählte Informationen über Symbole an.

Steuern der Listenausgabe, Fehlerbehandlung und Standardwerte

SHOW-MAP	Gibt Listen mit Informationen über das aktuelle LLM aus.
MODIFY-MAP-DEFAULTS	Ändert die Standardwerte für die Listenausgabe.
MODIFY-ERROR-PROCESSING	Legt fest, dass der BINDER-Lauf beim Auftreten von Fehlern bestimmter Fehlerklassen beendet wird, und steuert die Meldungsausgabe.
MODIFY-STD-DEFAULTS	Ändert globale Standardwerte für einen BINDER-Lauf.
START-STATEMENT-RECORDING	BINDER-Anweisungen protokollieren
STOP-STATEMENT-RECORDING	Protokollierung von BINDER-Anweisungen beenden

Beenden des BINDER-Laufs

END	Beendet den BINDER-Lauf.
-----	--------------------------

Die Standardanweisungen von SDF können zusätzlich angegeben werden. Sie werden (mit Ausnahme von END) nicht in diesem Handbuch beschrieben. Sie finden die Beschreibung dieser Anweisungen im Handbuch „Einführung in die Dialogschnittstelle SDF“ [13].

7.2 Hinweise zur SDF-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“ [13]).

Abkürzungsmöglichkeiten von Namen

SDF ermöglicht, Eingaben sowohl im Dialog- und Stapelbetrieb als auch in Prozeduren abzukürzen. Diese Abkürzungen müssen jedoch in der zugehörigen Syntax-Umgebung eindeutig sein. Besonders bei Funktionserweiterungen kann eine heute bestehende Eindeutigkeit wieder aufgehoben werden. Es wird deshalb empfohlen, gerade 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 Operandenwert 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.

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.

7.2.1 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 4: 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 4 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 4 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 4 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.	GUIDANCE-MODE = *YES
=	Das Gleichheitszeichen verbindet einen Operandennamen mit den dazugehörigen Operandenwerten.	GUIDANCE-MODE = *NO
< >	Spitze Klammern kennzeichnen Variablen, deren Wertevorrat durch Datentypen und ihre Zusätze beschrieben wird (siehe Tabellen 2 und 4).	SYNTAX-FILE = <filename 1..54>
<u>Unterstreichung</u>	Der Unterstrich kennzeichnet den Default-Wert eines Operanden.	GUIDANCE-MODE = *NO
/	Der Schrägstrich trennt alternative Operandenwerte.	NEXT-FIELD = *NO / *YES
(...)	Runde Klammern kennzeichnen Operandenwerte, die eine Struktur einleiten.	,UNGUIDED-DIALOG = *YES (...)/ *NO
[]	Eckige Klammern kennzeichnen struktureinleitende Operandenwerte, deren Angabe optional ist. Die nachfolgende Struktur kann ohne den einleitenden Operandenwert angegeben werden.	SELECT = [*BY-ATTRIBUTES](...)
Einrückung	Die Einrückung kennzeichnet die Abhängigkeit zu dem jeweils übergeordneten Operanden.	GUIDED-DIALOG = *YES (...) *YES(...) SCREEN-STEPS = *NO / *YES

Tabelle 1: Metasyntax (Teil 1 von 2)

Kennzeichnung	Bedeutung	Beispiele
<p style="text-align: center;"> </p> <p>,</p> <p>list-poss(n):</p>	<p>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 Struktur-tiefe.</p> <p>Das Komma steht vor weiteren Operanden der gleichen Struktur-stufe.</p> <p>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.</p>	<p>SUPPORT = *TAPE(...)</p> <pre> *TAPE(...) VOLUME = *ANY(...) ANY(...) ... </pre> <p>GUIDANCE-MODE = *NO / *YES</p> <p>,SDF-COMMANDS = *NO / *YES</p> <p>list-poss: *SAM / *ISAM</p> <p>list-poss(40): <structured-name 1..30></p> <p>list-poss(256): *OMF / *SYSLST(...) / <filename 1..54></p>
<p>Kurzname:</p>	<p>Der darauf folgende Name ist ein garantierter Aliasname des Kommando- bzw. Anweisungsnamens.</p>	<p>HELP-SDF Kurzname: HPSDF</p>

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	<p>Eingabeformat:</p> $[:cat:][\$user.] \left\{ \begin{array}{l} \text{datei} \\ \text{datei(nr)} \\ \text{gruppe} \\ \text{gruppe} \left\{ \begin{array}{l} (*abs) \\ (+rel) \\ (-rel) \end{array} \right\} \end{array} \right\}$ <p>: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.</p> <p>\$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.</p> <p>\$. (Sonderfall) System-Standardkennung</p> <p>datei Datei- oder Jobvariablenname; kann durch Punkt in mehrere Teilnamen gegliedert sein: name₁[.name₂[...]] 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.</p>

Tabelle 2: Datentypen (Teil 2 von 6)

Datentyp	Zeichenvorrat	Besonderheiten
filename (Forts.)		<p>#datei (Sonderfall) @datei (Sonderfall) # oder @ als erstes Zeichen kennzeichnet je nach Systemparameter temporäre Dateien und Jobvariablen.</p> <p>datei(nr) Banddateiname nr: Versionsnummer; Zeichenvorrat ist A...Z, 0...9, \$, #, @. Klammern müssen angegeben werden.</p> <p>gruppe Name einer Dateigenerationsgruppe (Zeichenvorrat siehe unter „datei“)</p> <p>gruppe $\left\{ \begin{array}{l} (*abs) \\ (+rel) \\ (-rel) \end{array} \right\}$</p> <p>(*abs) absolute Generationsnummer (1..9999); * und Klammern müssen angegeben werden.</p> <p>(+rel) (-rel) relative Generationsnummer (0..99); Vorzeichen und Klammern müssen angegeben werden.</p>
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: $[[C]][V][m]m.naso[']$ <div style="margin-left: 150px;"> Korrekturstand Freigabestand </div> <p>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 4, 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.</p>
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: $\left. \begin{array}{l} hh:mm:ss \\ hh:mm \\ hh \end{array} \right\}$ 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)

Sonderdatentypen

Datentyp	Zeichenvorrat	Besonderheiten
element-name	A...Z 0...9 \$,#,@ Bindestrich Punkt Unterstreichung	Die Zeichen Bindestrich, Unterstreichung und Punkt dürfen nicht erstes oder letztes Zeichen sein, und zwei gleiche der oben genannten Sonderzeichen dürfen nicht unmittelbar nebeneinander stehen. Der Bindestrich darf nicht unmittelbar rechts neben einem der Zeichen \$, @, #, Unterstreichung und Punkt stehen. Der Elementname muss mindestens einen Buchstaben oder eines der Sonderzeichen \$, #, @ enthalten.
element-version	A...Z 0...9 Bindestrich Punkt	Die Sonderzeichen Punkt und Bindestrich dürfen nicht erstes oder letztes Zeichen sein. Gleiche Sonderzeichen dürfen nicht unmittelbar nebeneinander stehen. Der Bindestrich darf nicht unmittelbar rechts von einem Punkt stehen.
path-name	beliebig	Eingabeformat siehe Seite 25
symbol	A...Z 0...9 \$,#,@,&,% Bindestrich Unterstreichung	erstes Zeichen: A...Z oder \$,#,@ Darüberhinaus sind auch Namen erlaubt, die aus 8 Leerzeichen bestehen.
symbol-with-wild	EBCDIC-Zeichen	ist in Hochkommata einzuschließen; Teile eines Namens dürfen durch Platzhalter ersetzt werden (siehe Tabelle 4).

Tabelle 3: Sonderdatentypen

Zusätze zu Datentypen

Zusatz	Bedeutung												
<i>x..y unit</i>	<p>beim Datentyp integer: Intervallangabe</p> <p><i>x</i> Mindestwert, der für integer erlaubt ist. <i>x</i> ist eine ganze Zahl, die mit einem Vorzeichen versehen werden darf.</p> <p><i>y</i> Maximalwert, der für integer erlaubt ist. <i>y</i> ist eine ganze Zahl, die mit einem Vorzeichen versehen werden darf.</p> <p><i>unit</i> Dimension. Folgende Angaben werden verwendet:</p> <table> <tr> <td><i>days</i></td> <td><i>byte</i></td> </tr> <tr> <td><i>hours</i></td> <td><i>2Kbyte</i></td> </tr> <tr> <td><i>minutes</i></td> <td><i>4Kbyte</i></td> </tr> <tr> <td><i>seconds</i></td> <td><i>Mbyte</i></td> </tr> <tr> <td><i>milliseconds</i></td> <td></td> </tr> </table>	<i>days</i>	<i>byte</i>	<i>hours</i>	<i>2Kbyte</i>	<i>minutes</i>	<i>4Kbyte</i>	<i>seconds</i>	<i>Mbyte</i>	<i>milliseconds</i>			
<i>days</i>	<i>byte</i>												
<i>hours</i>	<i>2Kbyte</i>												
<i>minutes</i>	<i>4Kbyte</i>												
<i>seconds</i>	<i>Mbyte</i>												
<i>milliseconds</i>													
<i>x..y special</i>	<p>bei den übrigen Datentypen: Längenangabe</p> <p>Bei den Datentypen <i>catid</i>, <i>date</i>, <i>device</i>, <i>product-version</i>, <i>time</i> und <i>vsn</i> wird die Längenangabe nicht angezeigt.</p> <p><i>x</i> Mindestlänge für den Operandenwert; <i>x</i> ist eine ganze Zahl.</p> <p><i>y</i> Maximallänge für den Operandenwert; <i>y</i> ist eine ganze Zahl.</p> <p><i>x=y</i> Der Operandenwert muss genau die Länge <i>x</i> haben.</p> <p><i>special</i> Zusatzangabe zur Beschreibung eines Sonderdatentyps, der durch die Implementierung geprüft wird. Vor <i>special</i> können weitere Zusätze stehen. Folgende Angaben werden verwendet:</p> <table> <tr> <td><i>arithm-expr</i></td> <td>arithmetischer Ausdruck (SDF-P)</td> </tr> <tr> <td><i>bool-expr</i></td> <td>logischer Ausdruck (SDF-P)</td> </tr> <tr> <td><i>string-expr</i></td> <td>String-Ausdruck (SDF-P)</td> </tr> <tr> <td><i>expr</i></td> <td>beliebiger Ausdruck (SDF-P)</td> </tr> <tr> <td><i>cond-expr</i></td> <td>bedingter Ausdruck (JV)</td> </tr> <tr> <td><i>symbol</i></td> <td>CSECT- oder Entry-Name (BLS)</td> </tr> </table>	<i>arithm-expr</i>	arithmetischer Ausdruck (SDF-P)	<i>bool-expr</i>	logischer Ausdruck (SDF-P)	<i>string-expr</i>	String-Ausdruck (SDF-P)	<i>expr</i>	beliebiger Ausdruck (SDF-P)	<i>cond-expr</i>	bedingter Ausdruck (JV)	<i>symbol</i>	CSECT- oder Entry-Name (BLS)
<i>arithm-expr</i>	arithmetischer Ausdruck (SDF-P)												
<i>bool-expr</i>	logischer Ausdruck (SDF-P)												
<i>string-expr</i>	String-Ausdruck (SDF-P)												
<i>expr</i>	beliebiger Ausdruck (SDF-P)												
<i>cond-expr</i>	bedingter Ausdruck (JV)												
<i>symbol</i>	CSECT- oder Entry-Name (BLS)												
<i>with</i>	Erweitert die Angabemöglichkeiten für einen Datentyp.												
<i>-compl</i>	Bei Angaben zu dem Datentyp <i>date</i> ergänzt SDF zweistellige Jahresangaben der Form <i>jj-mm-tt</i> zu:												
	<table> <tr> <td>20<i>jj</i>-<i>mm</i>-<i>tt</i></td> <td>falls <i>jj</i> < 60</td> </tr> <tr> <td>19<i>jj</i>-<i>mm</i>-<i>tt</i></td> <td>falls <i>jj</i> ≥ 60</td> </tr> </table>	20 <i>jj</i> - <i>mm</i> - <i>tt</i>	falls <i>jj</i> < 60	19 <i>jj</i> - <i>mm</i> - <i>tt</i>	falls <i>jj</i> ≥ 60								
20 <i>jj</i> - <i>mm</i> - <i>tt</i>	falls <i>jj</i> < 60												
19 <i>jj</i> - <i>mm</i> - <i>tt</i>	falls <i>jj</i> ≥ 60												
<i>-low</i>	Groß- und Kleinschreibung wird unterschieden.												
<i>-path-compl</i>	Bei Angaben zu dem Datentyp <i>filename</i> ergänzt SDF die Katalog- und/oder die Benutzerkennung, falls diese nicht angegeben werden.												
<i>-under</i>	Erlaubt Unterstriche ' _ ' bei den Datentypen <i>name</i> und <i>composed-name</i> .												

Tabelle 4: Zusätze zu Datentypen (Teil 1 von 7)

Zusatz	Bedeutung
with (Forts.) -wild(n)	<p>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.</p> <p>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.</p>
BS2000-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.
Punkt am Ende	Teilqualifizierte Angabe eines Namens. Entspricht implizit der Zeichenfolge „/*“, d.h. nach dem Punkt folgt mindestens ein beliebiges Zeichen.
/	Ersetzt genau ein beliebiges Zeichen.
<s _x :s _y >	<p>Ersetzt eine Zeichenfolge, für die gilt:</p> <ul style="list-style-type: none"> – 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')

Tabelle 4: Zusätze zu Datentypen (Teil 2 von 7)

Zusatz	Bedeutung	
with-wild(n) (Forts.)	<s ₁ ,...> -s	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 204). 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 ₁ c _x -c _y s 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 ₁ c _x -c _y s 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 ₁ c _x -c _y s ₂]	

Tabelle 4: Zusätze zu Datentypen (Teil 3 von 7)

Zusatz	Bedeutung										
with (Forts.) -wild- constr(n)	<p>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.</p> <p>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.</p> <p>Folgende Platzhalter können zur Konstruktionsangabe verwendet werden:</p> <table border="1"> <thead> <tr> <th>Platzhalter</th> <th>Bedeutung</th> </tr> </thead> <tbody> <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> </tbody> </table> <p>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:</p> <ol style="list-style-type: none"> 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. 	Platzhalter	Bedeutung	*	Entspricht der Zeichenfolge, die durch den Platzhalter * in der Auswahlzeichenfolge ausgewählt wird.	Punkt am Ende	Entspricht der teilqualifizierten Angabe eines Namens in der Auswahlzeichenfolge. Entspricht der Zeichenfolge, die durch den Punkt am Ende der Auswahlzeichenfolge ausgewählt wird.	/ oder ?	Entspricht dem Zeichen, das durch den Platzhalter / oder ? in der Auswahlzeichenfolge ausgewählt wird.	<n>	Entspricht der Zeichenfolge, die durch den n-ten Platzhalter in der Auswahlzeichenfolge ausgewählt wird; n = <integer>
Platzhalter	Bedeutung										
*	Entspricht der Zeichenfolge, die durch den Platzhalter * in der Auswahlzeichenfolge ausgewählt wird.										
Punkt am Ende	Entspricht der teilqualifizierten Angabe eines Namens in der Auswahlzeichenfolge. Entspricht der Zeichenfolge, die durch den Punkt am Ende der Auswahlzeichenfolge ausgewählt wird.										
/ oder ?	Entspricht dem Zeichen, das durch den Platzhalter / oder ? in der Auswahlzeichenfolge ausgewählt wird.										
<n>	Entspricht der Zeichenfolge, die durch den n-ten Platzhalter in der Auswahlzeichenfolge ausgewählt wird; n = <integer>										

Tabelle 4: Zusätze zu Datentypen (Teil 4 von 7)

Zusatz	Bedeutung
with-wild-constr(n) (Forts.)	<p>Bei Konstruktionsangaben sind folgende Regeln zu beachten:</p> <ul style="list-style-type: none"> – 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 4: Zusätze zu Datentypen (Teil 5 von 7)

Zusatz	Bedeutung																				
with-wild- constr(n) (Forts.)	Beispiele:																				
	<table border="1"> <thead> <tr> <th>Auswahlmuster</th> <th>Auswahl</th> <th>Konstruktionsmuster</th> <th>neuer Name</th> </tr> </thead> <tbody> <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> </tbody> </table>	Auswahlmuster	Auswahl	Konstruktionsmuster	neuer Name	A/*	AB1 AB2 A.B.C	D<3><2>	D1 D2 D.CB	C.<A:C>/<D,F>	C.AAD C.ABD C.BAF C.BBF	G.<1>.<3>.XY<2>	G.A.D.XYA G.A.D.XYB G.B.F.XYA G.B.F.XYB	C.<A:C>/<D,F>	C.AAD C.ABD C.BAF C.BBF	G.<1>.<2>.XY<2>	G.A.A.XYA G.A.B.XYB G.B.A.XYA G.B.B.XYB	A//B	ACDB ACEB AC.B A.CB	G/XY/	GCXYD GCXYE GCXY. ¹ G.XYC
	Auswahlmuster	Auswahl	Konstruktionsmuster	neuer Name																	
	A/*	AB1 AB2 A.B.C	D<3><2>	D1 D2 D.CB																	
	C.<A:C>/<D,F>	C.AAD C.ABD C.BAF C.BBF	G.<1>.<3>.XY<2>	G.A.D.XYA G.A.D.XYB G.B.F.XYA G.B.F.XYB																	
C.<A:C>/<D,F>	C.AAD C.ABD C.BAF C.BBF	G.<1>.<2>.XY<2>	G.A.A.XYA G.A.B.XYB G.B.A.XYA G.B.B.XYB																		
A//B	ACDB ACEB AC.B A.CB	G/XY/	GCXYD GCXYE GCXY. ¹ G.XYC																		
¹ Punkt am Ende des Namens kann Namenskonvention widersprechen (z.B bei vollqualifizierten Dateinamen)																					
without	Schränkt die Angabemöglichkeiten für einen Datentyp ein.																				
-cat	Die Angabe einer Katalogkennung ist nicht erlaubt.																				
-corr	Eingabeformat: [[C]'] [V][m]m.na['] Angaben zum Datentyp product-version dürfen den Korrekturstand nicht enthalten.																				
-gen	Die Angabe einer Dateigeneration oder Dateigenerationsgruppe ist nicht erlaubt.																				
-man	Eingabeformat: [[C]'] [V][m]m.n['] Angaben zum Datentyp product-version dürfen weder Freigabe- noch Korrekturstand enthalten.																				
-odd	Der Datentyp x-text erlaubt nur eine gerade Anzahl von Zeichen.																				
-sep	Beim Datentyp text ist die Angabe der folgenden Trennzeichen nicht erlaubt: ; = () < > _ (also Strichpunkt, Gleichheitszeichen, runde Klammer auf und zu, Größerzeichen, Kleinerzeichen und Leerzeichen)																				
-temp- file	Die Angabe einer temporären Datei ist nicht erlaubt (siehe #datei bzw. @datei bei filename).																				

Tabelle 4: Zusätze zu Datentypen (Teil 6 von 7)

Zusatz	Bedeutung
without (Forts.)	
-user	Die Angabe einer Benutzererkennung 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: <code>[[C]][V][m]m.naso[']</code> Angaben zum Datentyp product-version müssen den Korrekturstand (und damit auch den Freigabestand) enthalten.
-man	Eingabeformat: <code>[[C]][V][m]m.na[so][']</code> 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 4: Zusätze zu Datentypen (Teil 7 von 7)

7.3 Beschreibung der Anweisungen

Die Anweisungen sind in der alphabetischen Reihenfolge ihrer Namen geordnet. Die Beschreibung einer Anweisung ist gegliedert in:

- Anweisungsname und Funktion,
- Beschreibung der Anweisungsfunktion,
- Darstellung des Anweisungsformates,
- Beschreibung der Anweisungsoperanden.

Übersicht

Anweisungsname	Funktion
BEGIN-SUB-LLM-STATEMENTS	Beginn eines Sub-LLM festlegen
END	Beenden des BINDER-Laufs
END-SUB-LLM-STATEMENTS	Ende eines Sub-LLM festlegen
INCLUDE-MODULES	Einfügen von Modulen
MERGE-MODULES	Mischen von Modulen eines (Sub-)LLMs
MODIFY-ERROR-PROCESSING	Steuern der Fehlerbehandlung
MODIFY-LLM-ATTRIBUTES	Ändern der Merkmale eines LLM
MODIFY-MAP-DEFAULT	Ändern der Standardwerte für die Ausgabe von Listen
MODIFY-MODULE-ATTRIBUTES	Ändern der Attribute von Modulen
MODIFY-STD-DEFAULTS	Ändern von globalen Standardwerten
MODIFY-SYMBOL-ATTRIBUTES	Ändern der Attribute von Symbolen
MODIFY-SYMBOL-TYPE	Ändern des Typs von Symbolen
MODIFY-SYMBOL-VISIBILITY	Ändern der Maskierung von Symbolen
REMOVE-MODULES	Entfernen von Modulen
RENAME-SYMBOLS	Ändern von Symbolnamen
REPLACE-MODULES	Ersetzen von Modulen
RESOLVE-BY-AUTOLINK	Befriedigen von Externverweisen durch Autolink
SAVE-LLM	Speichern eines LLM
SET-EXTERN-RESOLUTION	Behandeln unbefriedigter Externverweise
SET-USER-SLICE-POSITION	Festlegen der Lage einer Slice
SHOW-DEFAULTS	Anzeigen von globalen Standardwerten
SHOW-LIBRARY-ELEMENTS	Anzeigen und Prüfen von Bibliothekselementen
SHOW-MAP	Ausgeben von Listen
SHOW-SYMBOL-INFORMATION	Anzeigen von Symbolinformationen
START-LLM-CREATION	Erzeugen eines LLM
START-LLM-UPDATE	Ändern eines LLM
START-STATEMENT-RECORDING	BINDER-Anweisungen protokollieren
STOP-STATEMENT-RECORDING	Protokollierung von BINDER-Anweisungen beenden

Außerdem stehen während des BINDER-Laufes auch die SDF-Standardanweisungen zur Verfügung. Diese werden (mit Ausnahme von END) nicht in diesem Handbuch beschrieben. Sie finden die Beschreibung dieser Anweisungen im Handbuch „Einführung in die Dialogschnittstelle SDF“ [13].

BEGIN-SUB-LLM-STATEMENTS

Beginn eines Sub-LLM festlegen

Diese Anweisung dient zur logischen Strukturierung eines LLM. Sie legt den Beginn eines Sub-LLM innerhalb eines LLM oder Sub-LLM fest.

Der Knoten, an dem das Sub-LLM beginnen soll, kann entweder durch den Pfadnamen festgelegt werden oder es kann der Knoten des aktuellen Sub-LLM gewählt werden.

Ein Sub-LLM wird in der Folge genauso erzeugt wie ein LLM, d.h. alle Anweisungen sind zulässig. Die Anweisung END-SUB-LLM-STATEMENTS schließt die Struktur des Sub-LLM ab.

BEGIN-SUB-LLM-STATEMENTS

```

SUB-LLM-NAME = <c-string 1..32 with-low> / <text 1..32>
, PATH-NAME = *CURRENT-SUB-LLM / <c-string 1..255 with-low> / <text 1..255>
, RESOLUTION-SCOPE = *UNCHANGED / *STD / *PARAMETERS(...)
  *PARAMETERS(...)
    |
    | HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <c-string 1..255 with-low> /
    | <text 1..255>
    |
    | , LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <c-string 1..255 with-low> /
    | <text 1..255>
    |
    | , FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE / <c-string 1..255 with-low> / <text 1..255>
    |
  , MODE = *CREATE / *UPDATE

```

SUB-LLM-NAME = <structured-name 1..32>

Legt den Namen fest, den das Sub-LLM erhält.

PATH-NAME =

Legt den Knoten in der logischen Struktur fest, an dem das Sub-LLM beginnen soll.

PATH-NAME = *CURRENT-SUB-LLM

Das Sub-LLM soll am Knoten des aktuellen Sub-LLM beginnen.

PATH-NAME = <text 1..255>

Pfadname des Knotens, an dem das Sub-LLM beginnen soll.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

RESOLUTION-SCOPE =

Legt Prioritätsklassen fest, die steuern, in welcher Reihenfolge der BINDER andere Module bei der Befriedigung von Externverweisen durchsuchen soll. Für jede Klasse müssen zwei Werte unterschieden werden:

- der dynamische Wert.
Er beeinflusst die Reihenfolge, in der Module zur Befriedigung von Externverweisen durchsucht werden. Er wird jedoch nicht im LLM abgespeichert.
- der statische Wert.
Er wird im LLM abgespeichert und bildet die Basis zur Ermittlung des dynamischen Wertes.

RESOLUTION-SCOPE = *UNCHANGED

Die statischen Werte der Prioritätsklassen sind die in den betroffenen Modulen gespeicherten Werte. Für Objektmodule (OM) erhalten sie den Wert *STD (siehe unten).

RESOLUTION-SCOPE = *STD

Die statischen Werte der Prioritätsklassen sind *STD. Das bedeutet, dass die dynamischen Werte vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen werden. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Die Voreinstellung für die statischen Werte des ROOT-Knotens eines LLMs ist *STD. Die dynamischen Werte sind in diesem Fall *NONE (siehe unten).

RESOLUTION-SCOPE = *PARAMETERS(...)

Die statischen Werte der einzelnen Prioritätsklassen werden separat vereinbart.

HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen vor allen anderen durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)).

HIGH-PRIORITY-SCOPE = *STD

Der statische Wert dieser Prioritätsklasse ist *STD. Das bedeutet, dass der dynamische Wert vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen wird. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Der statische Wert des ROOT-Knotens eines LLMs ist standardmäßig *STD. Der dynamische Wert ist in diesem Fall *NONE (siehe unten).

HIGH-PRIORITY-SCOPE = *NONE

Die Prioritätsklasse HIGH-PRIORITY-SCOPE ist nicht definiert. D.h. es gibt keine Module, die vor allen anderen zum Befriedigen von Externverweisen durchsucht werden sollen. Der Wert des übergeordneten Knotens wird nicht übernommen.

HIGH-PRIORITY-SCOPE = <c-string 1..255 with-low> / <text 1..255>

Pfadname des Sub-LLMs, das zum Befriedigen von Externverweisen zuerst durchsucht werden soll.

LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen erst dann durchsucht werden soll, wenn die Suche in allen anderen Modulen erfolglos war (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen nicht durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

Ein Beispiel zum Vererbungsmechanismus für die PRIORITY-SCOPE-Werte finden Sie bei der Beschreibung der Anweisung INCLUDE-MODULES auf [Seite 222](#).

MODE =

Legt fest, ob ein neues Sub-LLM erzeugt oder ein bereits vorhandenes geändert wird.

MODE = *CREATE

Ein neues Sub-LLM wird erzeugt.

MODE = *UPDATE

Ein bereits vorhandenes Sub-LLM wird geändert.

END

Beenden des BINDER-Laufs

Diese Anweisung beendet den BINDER-Lauf. Die vom BINDER eröffneten Eingabequellen werden geschlossen und es wird eine Endmeldung ausgegeben.

END

Die Anweisung END hat keine Operanden.

END-SUB-LLM-STATEMENTS

Ende eines Sub-LLM festlegen

Diese Anweisung dient zur logischen Strukturierung eines Sub-LLM. Sie legt das Ende eines Sub-LLM innerhalb eines LLM oder Sub-LLM fest.

Das aktuelle LLM wird auf die Stufe zurückgesetzt, die vor der zugehörigen Anweisung BEGIN-SUB-LLM-STATEMENTS das aktuelle LLM enthalten hat.

END-SUB-LLM-STATEMENTS

Die Anweisung END-SUB-LLM-STATEMENTS hat keine Operanden.

INCLUDE-MODULES

Einfügen von Modulen

Diese Anweisung liest einen oder mehrere Module aus der angegebenen Eingabequelle und fügt sie in den aktuellen LLM-Arbeitsbereich ein.

Als Module können sowohl Bindemodule (OMs) als auch LLMs eingefügt werden. Es ist aber nicht möglich, LLMs mit benutzerdefinierten Slices und LLMs ohne Relativierungsinformation einzufügen. Wenn die gesamte Strukturinformation beim Speichern des LLM übernommen wurde (Operand LOGICAL-STRUCTURE = WHOLE-LLM), dann können entweder LLMs vollständig eingefügt oder einzelne Sub-LLMs ausgewählt werden.

Die Eingabequelle kann sein:

- für Bindemodule eine Programmbibliothek (Elementtyp R), eine Bindemodulbibliothek (OML) oder die EAM-Bindemoduldatei,
- für LLMs bzw. Sub-LLMs eine Programmbibliothek (Elementtyp L).

INCLUDE-MODULES

```

MODULE-CONTAINER = *LIBRARY-ELEMENT (...) / *FILE(...) / *OMF(...)

*LIBRARY-ELEMENT(...)
    LIBRARY = *CURRENT-INPUT-LIB / <filename 1..54 without-gen-vers> / *LINK(...) /
        *BLSLIB-LINK / *OMF

        *LINK(...)
            | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
,ELEMENT = *ALL (...) / list-poss(40): <composed-name 1..64>(…) / <c-string 1..64>(…)

        *ALL(…)
            | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
            <composed-name>(…)
            | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
            ,SUB-LLM = *WHOLE-LLM / <c-string 1..255 with-low> / <text 1..255>
            <c-string>(…)
            | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
            ,SUB-LLM = *WHOLE-LLM / <c-string 1..255 with-low> / <text 1..255>

,TYPE = (*L,*R) / list-poss(2): *L / *R

```

Fortsetzung ➔

```

*FILE(...)
  | FILE-NAME = <filename 1..54 without-gen> / *LINK(...)
  | *LINK(...)
  | | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
  | ,SUB-LLM = *WHOLE-LLM / <c-string 1..255 with-low> / <text 1..255>
*OMF(...)
  | ELEMENT = *ALL / list-poss(40): <composed-name 1..64> / <c-string 1..64>
,NAME = *INTERNAL / *ELEMENT-NAME / <c-string 1..32 with-low> / <text 1..32>
,PATH-NAME = *CURRENT-SUB-LLM / <c-string 1..255 with-low> / <text 1..255>
,SLICE = *CURRENT-SLICE / *ROOT / <structured-name 1..32>
,LOGICAL-STRUCTURE = *INCLUSION-DEFAULT / *WHOLE-LLM / *OBJECT-MODULES
,TEST-SUPPORT = *INCLUSION-DEFAULT / *NO / *YES
,RUN-TIME-VISIBILITY = *UNCHANGED / *NO / *YES
,RESOLUTION-SCOPE = *UNCHANGED / *STD / *PARAMETERS(...)
  *PARAMETERS(...)
  | HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <c-string 1..255 with-low> /
  | <text 1..255>
  | ,LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <c-string 1..255 with-low> /
  | <text 1..255>
  | ,FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE / <c-string 1..255 with-low> / <text 1..255>
,NAME-COLLISION = STD / *IGNORED / *WARNING(...) / *ERROR(...)
  *WARNING(...)
  | SCOPE = *WHOLE-LLM / *SLICE
  *ERROR(...)
  | SCOPE = *WHOLE-LLM / *SLICE

```

MODULE-CONTAINER =

Legt fest, wo das LLM abgespeichert ist.

MODULE-CONTAINER = *LIBRARY-ELEMENT(...)

Das LLM ist in einer Programmbibliothek abgespeichert.

LIBRARY =

Gibt die Eingabequelle an, aus der die Module eingelesen werden.

LIBRARY = *CURRENT-INPUT-LIB

Es wird die Eingabequelle benutzt, aus der mit einer Anweisung START-LLM-UPDATE, INCLUDE-MODULES oder REPLACE-MODULES das *letzte* Modul (OM oder LLM) gelesen wurde. Der Geltungsbereich des Operanden bezieht sich auf einen *Edit-Lauf*.

LIBRARY = <filename 1..54 without-gen-vers>

Dateiname der Bibliothek, die als Eingabequelle verwendet werden soll.

LIBRARY = *LINK(...)

Bezeichnet eine Bibliothek mit dem Dateikettungsnamen.

LINKNAME = <structured-name 1..8>

Dateikettungsname der Bibliothek, die als Eingabequelle verwendet werden soll.

LIBRARY = *BLSLIB-LINK

Die Eingabequellen sind die Bibliotheken mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99). Die Bibliotheken werden in der Reihenfolge nach *aufsteigenden* Werten „nn“ des Dateikettungsnamens durchsucht.

ELEMENT =

Legt den Elementnamen und die Elementversion der Module fest, die aus der angegebenen Eingabequelle eingefügt werden.

ELEMENT = *ALL(...)

Alle Module werden aus der angegebenen Eingabequelle eingefügt.

VERSION =

Gibt die Elementversion des Moduls an. Die Elementversion ist nur für Programm-bibliotheken gültig.

VERSION = *HIGHEST-EXISTING

Der BINDER übernimmt als Elementversion den Standardwert für die höchste Version bei Programmbibliotheken (siehe Handbuch „LMS“ [3]).

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

ELEMENT = <composed-name 1..64>(...)

Explizite Angabe des Elementnamens und der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-name> und <element-version> (siehe [Seite 202](#))

VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>

Gibt die Elementversion des Moduls an. Die Elementversion ist nur für Programm-bibliotheken gültig.

Bedeutung der Operanden siehe oben.

SUB-LLM =

Legt fest, ob das vollständige LLM oder ein Sub-LLM eingefügt wird.

SUB-LLM = *WHOLE-LLM

Das vollständige LLM wird eingefügt.

SUB-LLM = <text 1..255>

Pfadname des Sub-LLM, das eingefügt wird.

Achtung: Der BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

ELEMENT = <c-string 1..64>(…)

Explizite Angabe des Elementnamens und der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-name> und <element-version> (siehe [Seite 202](#)).

Bedeutung der Operanden siehe oben.

TYPE =

Legt die Priorität der Module (Bindemodule und/oder LLMs) fest, die eingefügt werden.

TYPE = (*L,*R)

Sowohl LLMs als auch Bindemodule werden eingefügt. Ist für ein LLM der gleiche Name wie für ein Bindemodul angegeben, wird das *LLM* eingefügt.

TYPE = (*R,*L)

Sowohl LLMs als auch Bindemodule werden eingefügt. Ist für ein LLM der gleiche Name wie für ein Bindemodul angegeben, wird das *Bindemodul* eingefügt.

TYPE = *R

Nur Bindemodule werden eingefügt.

TYPE = *L

Nur LLMs werden eingefügt.

MODULE-CONTAINER = *FILE(…)**FILE-NAME =**

Das LLM ist in einer PAM-Datei abgespeichert.

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

Name der PAM-Datei, die das LLM enthält.

FILE-NAME = *LINK(…)**LINK-NAME = <structured-name 1..8>**

Dateikettungsname der PAM-Datei, die das LLM enthält.

MODULE-CONTAINER = *OMF(…)

Die Eingabequelle ist die EAM-Bindemoduldatei. Sie enthält nur Bindemodule. (Falls der Operand NAME den Wert *ELEMENT-NAME hat, ersetzt der BINDER diesen Wert in diesem Fall durch *INTERNAL.)

ELEMENT = *ALL / list-poss(40): <composed-name 1..64> / <c-string 1..64>

Bedeutung der Operanden siehe oben.

NAME =

Gibt an, welchen logischen Namen das einzufügende Modul haben soll.

NAME = *INTERNAL

Der interne Name wird als logischer Name übernommen.

NAME = *ELEMENT-NAME

Als logischer Name des Moduls wird der Name des Bibliothekselementes übernommen.

Wenn nötig, kürzt der BINDER diesen Namen auf 32 Zeichen.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

NAME = <structured-name 1..32>

Explizite Angabe des logischen Namens.

PATH-NAME =

Legt das Sub-LLM in der logischen Struktur des aktuellen LLM im Arbeitsbereich fest, in das Module eingefügt werden.

PATH-NAME = *CURRENT-SUB-LLM

Das aktuelle Sub-LLM wird angenommen

(siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

PATH-NAME = <text 1..255>

Pfadname des Sub-LLM in der logischen Struktur des aktuellen LLM.

Achtung: Der BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

SLICE =

Bestimmt die Slice in der physischen Struktur des LLM, in die die Module eingefügt werden.

Die Slice muss mit einer Anweisung SET-USER-SLICE-POSITION definiert sein

(siehe Anweisung SET-USER-SLICE-POSITION).

SLICE = *CURRENT-SLICE

Module werden in die aktuelle Slice eingefügt. Dies ist die Slice, die durch die letzte vorhergehende Anweisung SET-USER-SLICE-POSITION definiert wurde.

SLICE = *ROOT

Module werden in die Root-Slice (%ROOT) eingefügt.

SLICE = <structured-name 1..32>

Explizite Angabe der Slice, in die Module eingefügt werden.

LOGICAL-STRUCTURE =

Legt fest, ob die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird.

LOGICAL-STRUCTURE = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben *Edit-Laufs* angenommen.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird in das aktuelle LLM übernommen.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Die Strukturinformation wird nicht übernommen. Im aktuellen LLM wird nur eine Struktur aus Bindemodulen (OMs) aufgebaut.

TEST-SUPPORT =

Legt fest, ob die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.



Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben *Edit-Laufs* angenommen.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

RUN-TIME-VISIBILITY =

Legt fest, ob das Modul als Laufzeitmodul betrachtet werden soll. Alle Symbole eines Laufzeitmoduls werden beim Abspeichern maskiert und zunächst nicht mehr zur Befriedigung von Externverweisen verwendet. Bei einem späteren Lesezugriff auf das Modul (z.B. bei START-LLM-UPDATE oder INCLUDE-MODULES) wird die Maskierung wieder rückgängig gemacht.

RUN-TIME-VISIBILITY = *UNCHANGED

Der Wert wird nicht geändert. Wird ein Modul erstmals mit INCLUDE-MODULES oder REPLACE-MODULES in ein LLM eingefügt, so übernimmt der BINDER den Wert NO.

RUN-TIME-VISIBILITY = *NO

Das Modul soll nicht als Laufzeitmodul betrachtet werden.

RUN-TIME-VISIBILITY = *YES

Das Modul soll als Laufzeitmodul betrachtet werden. Alle Symbole des Moduls werden beim Abspeichern maskiert.

RESOLUTION-SCOPE =

Legt Prioritätsklassen fest, die steuern, in welcher Reihenfolge der BINDER andere Module bei der Befriedigung von Externverweisen durchsuchen soll. Für jede Klasse müssen zwei Werte unterschieden werden:

- der dynamische Wert.
Er beeinflusst die Reihenfolge, in der Module zur Befriedigung von Externverweisen durchsucht werden. Er wird jedoch nicht im LLM abgespeichert.
- der statische Wert.
Er wird im LLM abgespeichert und bildet die Basis zur Ermittlung des dynamischen Wertes.

RESOLUTION-SCOPE = *UNCHANGED

Die statischen Werte der Prioritätsklassen sind die in den betroffenen Modulen gespeicherten Werte. Für Objektmodule (OM) erhalten sie den Wert *STD (siehe unten).

RESOLUTION-SCOPE = *STD

Die statischen Werte der Prioritätsklassen sind *STD. Das bedeutet, dass die dynamischen Werte vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen werden. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Die Voreinstellung für die statischen Werte des ROOT-Knotens eines LLMs ist *STD. Die dynamischen Werte sind in diesem Fall *NONE (siehe unten).

RESOLUTION-SCOPE = *PARAMETERS(...)

Die statischen Werte der einzelnen Prioritätsklassen werden separat vereinbart.

**HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /
<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen vor allen anderen durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)).

HIGH-PRIORITY-SCOPE = *STD

Der statische Wert dieser Prioritätsklasse ist *STD. Das bedeutet, dass der dynamische Wert vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen wird. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Der statische Wert des ROOT-Knotens eines LLMs ist standardmäßig *STD. Der dynamische Wert ist in diesem Fall *NONE (siehe unten).

HIGH-PRIORITY-SCOPE = *NONE

Die Prioritätsklasse HIGH-PRIORITY-SCOPE ist nicht definiert. D.h. es gibt keine Module, die vor allen anderen zum Befriedigen von Externverweisen durchsucht werden sollen. Der Wert des übergeordneten Knotens wird nicht übernommen.

HIGH-PRIORITY-SCOPE = <c-string 1..255 with-low> / <text 1..255>

Pfadname des Sub-LLMs, das zum Befriedigen von Externverweisen zuerst durchsucht werden soll.

LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen erst dann durchsucht werden soll, wenn die Suche in allen anderen Modulen erfolglos war (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen nicht durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

Folgendes Beispiel soll den Vererbungsmechanismus für die PRIORITY-SCOPE-Werte verdeutlichen:

Eine Bibliothek enthält die beiden LLMs LLM1 und LLM2. Für diese sind statische HIGH-PRIORITY-SCOPE-Werte festgelegt, die in der folgenden Abbildung aufgelistet sind.

Modul	statischer Wert des HIGH-PRIORITY-SCOPE
LLM1	LLM1.OM12
├── OM11	*STD
└── OM12	*NONE
LLM2	*STD
├── OM21	LLM2.OM22
└── OM22	*STD

LLM2 wird in LLM1 eingebunden. BINDER ermittelt beim Binden aus den statischen HIGH-PRIORITY-SCOPE-Werten die dynamischen, die die Suchreihenfolge beim Befriedigen der Externverweise beeinflussen.

Folgende Abbildung zeigt die sich dadurch ergebende Modul-Struktur und die dynamischen Werte des HIGH-PRIORITY-SCOPE.

Modul	dynamischer Wert des HIGH-PRIORITY-SCOPE	Bemerkung
LLM1	LLM1.OM12	= statischer Wert
├── OM11	*STD	geerbt von LLM1
├── OM12	*NONE	= statischer Wert
└── LLM2	*STD	geerbt von LLM1
├── OM21	LLM2.OM22	= statischer Wert
└── OM22	*STD	geerbt von LLM1

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Die Ausführung der Anweisung wird abgebrochen, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

MERGE-MODULES

Mischen von LLMs oder Sub-LLMs

Diese Anweisung mischt alle Module eines LLMs oder Sub-LLMs, so dass das neue LLM bzw. Sub-LLM nur noch ein Großmodul mit einer einzigen CSECT enthält. Der Benutzer kann angeben, welche Attribute die neue CSECT erhält und welche Symbole im Extern-adressbuch bleiben.

MERGE-MODULES

```

NAME = <c-string 1..32 with-low> / <text 1..32>
, PATH-NAME = *CURRENT-SUB-LLM / *NONE / <c-string 1..255 with-low> / <text 1..255>
, NEW-CSECT-NAME = *NAME / *STD / <c-string 1..32 with-low> / <text 1..32>
, NEW-CSECT-ATTRIBUTES = *PARAMETERS (...)
  *PARAMETERS(...)
    | RESIDENT = STD / *YES / *NO / *UNIFORM
    | PUBLIC = *STD / *NO / *YES / *UNIFORM
    | READ-ONLY = *STD / *NO / *YES / *UNIFORM
    | ALIGNMENT = *STD / *DOUBLE-WORD / *PAGE / *BUNDLE / <integer 3..12>
    | ADDRESSING-MODE = *UNIFORM / *STD / *24 / *31 / *ANY
    | RESIDENCY-MODE = *STD / *24 / *ANY / *UNIFORM
, ENTRY-LIST = *NONE / *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
, NAME-COLLISION = *STD / *IGNORED / *WARNING(...) / *ERROR(...)
  *WARNING(...)
    | SCOPE = *WHOLE-LLM / *SLICE
  *ERROR(...)
    | SCOPE = *WHOLE-LLM / *SLICE

```

NAME = <c-string 1..32 with-low> / <text 1..32>

Name des Sub-LLM oder LLM, das gemischt werden soll.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

PATH-NAME =

Legt den in der logischen Struktur übergeordneten Knoten des Sub-LLM fest, das gemischt werden soll.

PATH-NAME = *CURRENT-SUB-LLM

Das aktuelle Sub-LLM wird angenommen
(siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

PATH-NAME = *NONE

Das gesamte LLM, das gerade bearbeitet wird, soll gemischt werden.

PATH-NAME = <text 1..255>

Pfadname des Sub-LLM in der logischen Struktur des aktuellen LLM.
Achtung: Der BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

NEW-CSECT-NAME =

Legt den Namen fest, den die gemischte CSECT erhalten soll.

NEW-CSECT-NAME = *NAME

Die neue CSECT erhält den Namen des gemischten Bindemoduls.

NEW-CSECT-NAME = *STD

Die neue CSECT erhält den Namen der ersten CSECT, die in dem zu mischenden (Sub-)LLM gefunden wird.

NEW-CSECT-NAME = <text 1..32>

Explizite Angabe des neuen CSECT-Namens.
Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

NEW-CSECT-ATTRIBUTES = *PARAMETERS(...)

Legt die Attribute fest, die die neue CSECT erhalten soll.

RESIDENT =

Legt fest, welchen Wert das Attribut RESIDENT erhalten soll.

RESIDENT = *STD

Die neue CSECT erhält das Attribut RESIDENT, wenn *mindestens eine* CSECT mit dem Attribut RESIDENT im zu mischenden Sub-LLM vorhanden ist. Ist das nicht der Fall, dann ist die CSECT seitenwechselbar (RESIDENT=NO).

RESIDENT = *YES

Die neue CSECT ist resident.

RESIDENT = *NO

Die neue CSECT ist seitenwechselbar.

RESIDENT = *UNIFORM

Bei allen CSECTs, die in den Mischvorgang einbezogen werden, muss der Wert des Attributes RESIDENT gleich sein, sonst wird die Anweisung MERGE-MODULES abgewiesen. Die neue CSECT erhält ebenfalls diesen Attributwert.

PUBLIC =

Legt fest, welchen Wert das Attribut PUBLIC erhalten soll.

PUBLIC = *STD

Die neue CSECT ist PRIVATE (PUBLIC=NO), wenn *mindestens eine* CSECT mit dem Attribut PUBLIC=NO im zu mischenden Sub-LLM vorhanden ist. Ist das nicht der Fall, dann ist die CSECT gemeinsam benutzbar (PUBLIC=YES).

PUBLIC = *NO

Die neue CSECT ist nicht gemeinsam benutzbar (PRIVATE).

PUBLIC = *YES

Die neue CSECT ist gemeinsam benutzbar.

PUBLIC = *UNIFORM

Bei allen CSECTs, die in den Mischvorgang einbezogen werden, muss der Wert des Attributes PUBLIC gleich sein, sonst wird die Anweisung MERGE-MODULES zurückgewiesen. Die neue CSECT erhält ebenfalls diesen Attributwert.

READ-ONLY =

Legt fest, welchen Wert das Attribut READ-ONLY erhalten soll.

READ-ONLY = *STD

Auf die neue CSECT ist Lese-/Schreibzugriff erlaubt, wenn *mindestens eine* CSECT mit dem Attribut READ-ONLY=NO im zu mischenden Sub-LLM vorhanden ist. Ist das nicht der Fall, dann ist die CSECT nur lesbar (READ-ONLY=YES).

READ-ONLY = *NO

Die neue CSECT ist nur lesbar.

READ-ONLY = *YES

Die neue CSECT ist lese-/schreibbar.

READ-ONLY = *UNIFORM

Bei allen CSECTs, die in den Mischvorgang einbezogen werden, muss der Wert des Attributes READ-ONLY gleich sein, sonst wird die Anweisung MERGE-MODULES zurückgewiesen. Die neue CSECT erhält ebenfalls diesen Attributwert.

ALIGNMENT =

Legt die Ausrichtung der neuen CSECT fest.

ALIGNMENT = *STD

Die Ausrichtung der neuen CSECT ist die *größte* Ausrichtung aller zu mischenden CSECTs.

ALIGNMENT = *DOUBLE-WORD

Die neue CSECT wird auf Doppelwortgrenze ausgerichtet.

ALIGNMENT = *PAGE

Die neue CSECT wird auf Seitengrenze ausgerichtet, d.h. die Adresse ist ein Vielfaches von 4096 (X'1000').

ALIGNMENT = *BUNDLE

Die neue CSECT wird auf Doppelwortgrenze ausgerichtet, d.h. die Adresse ist ein Vielfaches von 16.

ALIGNMENT = <integer 3..12>

Die neue CSECT wird auf eine Adresse ausgerichtet, die ein Vielfaches von 2^n ist. Der Exponent „n“ wird mit <integer 3..12> festgelegt. (Z.B. bedeutet 2^3 Doppelwortausrichtung und 2^{12} Ausrichtung auf Seitengrenze.)

ADDRESSING-MODE =

Legt fest, welchen Adressierungsmodus (AMODE) die neue CSECT haben soll.

ADDRESSING-MODE = *UNIFORM

Wenn das Attribut AMODE nicht bei allen zu mischenden CSECTs gleich ist, wird die Anweisung zurückgewiesen. Die neue CSECT erhält ebenfalls diesen Attributwert.

ADDRESSING-MODE = *STD

Der Adressierungsmodus der neuen CSECT ist gleich dem der ersten CSECT im zu mischenden Sub-LLM.

ADDRESSING-MODE = 24

Der neuen CSECT wird der 24-Bit-Adressierungsmodus zugeordnet.

ADDRESSING-MODE = 31

Der neuen CSECT wird der 31-Bit-Adressierungsmodus zugeordnet.

ADDRESSING-MODE = *ANY

Der neuen CSECT wird der 24- oder 31-Bit-Adressierungsmodus zugeordnet. Die Entscheidung zwischen diesen beiden Adressierungsmodi wird erst beim Laden getroffen.

RESIDENCY-MODE =

Legt fest, welchen Residenzmodus die neue CSECT erhalten soll.

RESIDENCY-MODE = *STD

Der Residenzmodus der neuen CSECT ist 24, wenn mindestens eine der zu mischenden CSECTs den Residenzmodus 24 hat; ansonsten wird der Wert ANY angenommen.

RESIDENCY-MODE = 24

Der Residenzmodus der neuen CSECT ist 24.

RESIDENCY-MODE = *ANY

Die neue CSECT darf oberhalb und unterhalb 16 Mbyte geladen.

RESIDENCY-MODE = *UNIFORM

Bei allen CSECTs, die in den Mischvorgang einbezogen werden, muss der Wert des Attributes RESIDENCY-MODE gleich sein, sonst wird die Anweisung MERGE-MODULES zurückgewiesen. Die neue CSECT erhält ebenfalls diesen Attributwert.

ENTRY-LIST =

Legt die Symbole (CSECTs oder ENTRYs) fest, die weiterhin im Externadressbuch bleiben sollen. Davon betroffene CSECTs werden in ENTRYs umgewandelt.

ENTRY-LIST = *NONE

Es sollen keine Symbole im Externadressbuch bleiben.

ENTRY-LIST = *ALL

Alle Symbole sollen im Externadressbuch bleiben.

ENTRY-LIST = <c-string 1..255>

Alle Symbole, die zum angegebenen Wildcard-Muster passen, sollen im Externadressbuch bleiben.

Achtung: BINDER prüft Sonderdatentyp <symbol-with-wild> (siehe [Seite 202](#)).

ENTRY-LIST = <text 1..32>

Explizite Angabe aller Symbole, die im Externadressbuch bleiben sollen.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten. Ein Namenskonflikt kann nur dann auftreten, wenn die neue CSECT einen neuen Namen erhält.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Legt fest, dass die Anweisung abgebrochen wird, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

MODIFY-ERROR-PROCESSING

Steuern der Fehlerbehandlung

Diese Anweisung steuert die Fehlerbehandlung und das Beenden des BINDER-Laufs. Festgelegt werden kann

- die niedrigste Fehlerklasse, ab der Meldungen ausgegeben werden,
- das Ausgabemedium, auf dem die Meldungen ausgegeben werden,
- ab welcher Fehlerklasse der BINDER-Lauf beendet werden soll und
- ob beim Auftreten bestimmter Fehler Benutzerschalter und/oder Auftragschalter gesetzt werden.

Der Geltungsbereich aller Operanden bezieht sich auf einen BINDER-Lauf. Der Wert **UNCHANGED** in den Operanden bedeutet, dass der Standardwert erhalten bleibt. In der ersten Anweisung MODIFY-ERROR-PROCESSING wird für UNCHANGED der *erste* auf den Wert UNCHANGED folgende Operandenwert angenommen.

MODIFY-ERROR-PROCESSING

```

MESSAGE-CONTROL = *UNCHANGED / *INFORMATION / *WARNING / *ERROR
,MESSAGE-DESTINATION = *UNCHANGED / *SYSOUT / *SYSLST / *BOTH
,MAX-ERROR-WEIGHT = *UNCHANGED / *FATAL / *RECOVERABLE / *SYNTAX /
                    *UNRESOLVED-EXTERNS / *WARNING
,SPECIAL-HANDLING = *UNCHANGED / *NO / *ALL(...) / *PARAMETERS(...)
  *ALL(...)
    | USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
    | ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>

```

Fortsetzung ➡

```

*PARAMETERS(...)
  WARNING = *UNCHANGED / *NO / *PARAMETERS(...)
    *PARAMETERS(...)
      USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
      ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>
    ,UNRESOLVED-EXTERNS = *UNCHANGED / *NO / *PARAMETERS(...)
      *PARAMETERS(...)
        USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
        ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>
      ,SYNTAX-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)
        *PARAMETERS(...)
          USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
          ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>
        ,RECOVERABLE-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)
          *PARAMETERS(...)
            USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
            ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>
          ,FATAL-ERROR = UNCHANGED / *NO / *PARAMETERS(...)
            *PARAMETERS(...)
              USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
              ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>
            ,INTERNAL-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)
              *PARAMETERS(...)
                USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>
                ,TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>

```

MESSAGE-CONTROL = *UNCHANGED / *INFORMATION / *WARNING / *ERROR

Legt die niedrigste Fehlerklasse fest, ab der Meldungen ausgegeben werden (siehe Seite 135).

MESSAGE-CONTROL = *INFORMATION

Die Meldungen aller Fehlerklassen werden ausgegeben.

MESSAGE-CONTROL = *WARNING

Nur Meldungen ab der Fehlerklasse WARNING werden ausgegeben. Nicht ausgegeben werden Meldungen der Fehlerklasse INFORMATION.

MESSAGE-CONTROL = *ERROR

Nur Meldungen der Fehlerklassen SYNTAX ERROR, RECOVERABLE ERROR, FATAL ERROR und INTERNAL ERROR werden ausgegeben. Nicht ausgegeben werden Meldungen der Fehlerklassen INFORMATION, WARNING und UNRESOLVED EXTERNS.

MESSAGE-DESTINATION = *UNCHANGED / *SYSOUT / *SYSLST / *BOTH

Legt das Ausgabeziel für die Meldungen fest.

MESSAGE-DESTINATION = *SYSOUT

Ausgabeziel ist die Systemdatei SYSOUT.

MESSAGE-DESTINATION = *SYSLST

Ausgabeziel ist die Systemdatei SYSLST.

MESSAGE-DESTINATION = *BOTH

Ausgabeziel sind die Systemdateien SYSOUT und SYSLST.

MAX-ERROR-WEIGHT = *UNCHANGED / *FATAL / *RECOVERABLE / *SYNTAX / *UNRESOLVED-EXTERNS / *WARNING

Legt fest, bei welcher Fehlerklasse der BINDER-Lauf beendet wird. Der BINDER-Lauf wird bei allen Fehlern beendet, deren Fehlerklasse gleich oder größer ist als die angegebene Fehlerklasse (siehe [Seite 135](#)).

SPECIAL-HANDLING = *UNCHANGED / *NO / *ALL(...) / *PARAMETERS(...)

Legt fest, ob beim Auftreten von Fehlern Benutzer- oder Auftragsschalter gesetzt werden. Wenn ja, kann unterschieden werden, ob der angegebene Benutzer- oder Auftragsschalter gesetzt wird

- beim Auftreten irgendeines Fehlers oder
- beim Auftreten eines Fehlers einer bestimmten Fehlerklasse.

SPECIAL-HANDLING = *NO

Es werden keine Schalter gesetzt.

SPECIAL-HANDLING = *ALL (...)

Alle Fehler führen zum Setzen des angegebenen Benutzer- oder Auftragsschalters. Für jede Fehlerklasse kann nur ein Schalter gesetzt werden.

USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>

Nummer des Benutzerschalters.

Bei NO wird kein Benutzerschalter gesetzt.

TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>

Nummer des Auftragsschalters.

Bei NO wird kein Auftragsschalter gesetzt.

SPECIAL-HANDLING = *PARAMETERS (...)

Beim Auftreten von Fehlern der ausgewählten Fehlerklassen werden die angegebenen Benutzer- oder Auftragsschalter gesetzt. Für jede Fehlerklasse kann nur ein Schalter gesetzt werden.

WARNING = *UNCHANGED / *NO / *PARAMETERS (...)

Legt fest, ob ein Schalter für die Fehlerklasse WARNING gesetzt wird.

WARNING = *NO

Es wird kein Schalter gesetzt.

WARNING = *PARAMETERS (...)

Legt die Nummer des Schalters fest, der gesetzt wird.

USER-SWITCH = *UNCHANGED / *NO / <integer 0..31>

Nummer des Benutzerschalters.

Bei *NO wird kein Benutzerschalter gesetzt.

TASK-SWITCH = *UNCHANGED / *NO / <integer 0..31>

Nummer des Auftragsschalters.

Bei NO wird kein Auftragsschalter gesetzt.

UNRESOLVED-EXTERNS = *UNCHANGED / *NO / *PARAMETERS(...)

Legt fest, ob ein Schalter für die Fehlerklasse UNRESOLVED EXTERNS gesetzt wird.

Operanden siehe WARNING.

SYNTAX-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)

Legt fest, ob ein Schalter für die Fehlerklasse SYNTAX ERROR gesetzt wird.

Operanden siehe WARNING.

RECOVERABLE-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)

Legt fest, ob ein Schalter für die Fehlerklasse RECOVERABLE ERROR gesetzt wird.

Operanden siehe WARNING.

FATAL-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)

Legt fest, ob ein Schalter für die Fehlerklasse FATAL ERROR gesetzt wird.

Operanden siehe WARNING.

INTERNAL-ERROR = *UNCHANGED / *NO / *PARAMETERS(...)

Legt fest, ob ein Schalter für die Fehlerklasse INTERNAL ERROR gesetzt wird. Operanden siehe WARNING.

MODIFY-LLM-ATTRIBUTES

Ändern der Merkmale eines LLM

Diese Anweisung ändert Merkmale eines LLM, die mit der Anweisung START-LLM-CREATION oder START-LLM-UPDATE festgelegt wurden.

Folgende Merkmale können geändert werden:

- der interne Name (INTERNAL-NAME),
- die interne Version (INTERNAL-VERSION),
- die physische Struktur des LLM (SLICE-DEFINITION),
- die Copyright-Information (COPYRIGHT),
- Verwendung der Strukturinformation und LSD-Information (INCLUSION-DEFAULTS).

Folgende Änderungen des Typs der physischen Struktur sind erlaubt:

1. LLM mit nach Attributen gebildeten Slices in ein LLM mit Einzelslice
2. LLM mit Einzelslice in ein LLM mit nach Attributen gebildeten Slices
3. LLM mit nach Attributen gebildeten Slices in ein LLM mit nach *anderen* Attributen gebildeten Slices
4. LLM mit benutzerdefinierten Slices in ein LLM mit benutzerdefinierten Slices und geänderten Werten für AUTOMATIC-CONTROL und EXCLUSIVE-SLICE-CALL.

Der Standardwert **UNCHANGED** in den entsprechenden Operanden bedeutet, dass die bisherige (in START-LLM-CREATION festgelegte) Vereinbarung gilt.

MODIFY-LLM-ATTRIBUTES

```

INTERNAL-NAME = *UNCHANGED / <c-string 1..32 with-low> / <text 1..32>
, INTERNAL-VERSION = *UNCHANGED / *UNDEFINED / <composed-name 1..24> / <c-string 1..24>
, SLICE-DEFINITION = *UNCHANGED / *SINGLE / *BY-ATTRIBUTES(...) / *BY-USER(...)
  *BY-ATTRIBUTES(...)
    | READ-ONLY = *UNCHANGED / *NO / *YES
    | , RESIDENT = *UNCHANGED / *NO / *YES
    | , PUBLIC = *UNCHANGED / *NO / *YES(...)
    |   *YES(...)
    |     | SUBSYSTEM-ENTRIES = *NONE / list-poss(40): <c-string 1..32 with-low> /
    |       <text 1..32>
    |   , RESIDENCY-MODE = *UNCHANGED / *NO / *YES
  *BY-USER(...)
    | AUTOMATIC-CONTROL = *UNCHANGED / *NO / *YES(...)
    |   *YES(...)
    |     | RELOAD-SLICE = *YES / *NO
    |   , EXCLUSIVE-SLICE-CALL = *UNCHANGED / *NO / *YES
, COPYRIGHT = *UNCHANGED / *PARAMETERS(...) / *NONE
  *PARAMETERS(...)
    | NAME = *UNCHANGED / *SYSTEM-DEFAULT / <c-string 1..64 with-low>
    | , YEAR = *UNCHANGED / *CURRENT / <integer 1900..2100>
    | , PATH-NAME = *UNCHANGED / *NONE / <c-string 1..255 with-low> / <text 1..255>
    | , ENTRY = *UNCHANGED / *NONE / <c-string 1..32 with-low> / <text 1..32>
, INCLUSION-DEFAULTS = *PARAMETERS (...)
  *PARAMETERS(...)
    | LOGICAL-STRUCTURE = *UNCHANGED / *WHOLE-LLM / *OBJECT-MODULES
    | , TEST-SUPPORT = *UNCHANGED / *NO / *YES

```

INTERNAL-NAME = *UNCHANGED / <structured-name 1..32>

Legt den neuen internen Namen des LLM fest.

INTERNAL-VERSION = *UNCHANGED / *UNDEFINED / <composed-name 1..24> / <c-string 1..24>

Legt die neue interne Version des LLM fest.

INTERNAL-VERSION = *UNDEFINED

Beim Speichern des LLM mit der Anweisung SAVE-LLM wird der Standardwert für die höchste Version bei Programmbibliotheken angenommen (siehe Handbuch „LMS“ [3]).

INTERNAL-VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der neuen internen Version des LLM.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

SLICE-DEFINITION = *UNCHANGED / *SINGLE / *BY-ATTRIBUTES(...) / *BY-USER(...)

Legt die neue physische Struktur des LLM fest.

SLICE-DEFINITION = *SINGLE

Das LLM besteht aus einer Einzel-Slice.

SLICE-DEFINITION = *BY-ATTRIBUTES(...)

Das LLM besteht aus Slices, die durch Kombination der Attribute von CSECTs gebildet wurden (siehe [Seite 18ff](#)). Ist der Operand BY-ATTRIBUTES angegeben und sind alle Unteroperanden auf NO gesetzt, wird SINGLE angenommen. Die Operandenwerte von READ-ONLY, RESIDENT, PUBLIC und RESIDENCY-MODE bewirken nur das Zusammenfassen zu Slices. Sie haben keine Auswirkung auf die einzelnen CSECTs. Maximal können 16 verschiedene Slices durch Kombination der Attribute gebildet werden.

READ-ONLY = *UNCHANGED / *NO / *YES

Legt fest, ob das Attribut READ-ONLY bei der Bildung der Slices berücksichtigt wird. Bei Angabe YES bildet der BINDER für CSECTs mit unterschiedlichem Attribut READ-ONLY getrennte Slices.

RESIDENT = *UNCHANGED / *NO / *YES

Legt fest, ob das Attribut RESIDENT bei der Bildung der Slices berücksichtigt wird. Bei Angabe YES bildet der BINDER für CSECTs mit unterschiedlichem Attribut RESIDENT getrennte Slices.

PUBLIC = *UNCHANGED / *NO / *YES(...)

Legt fest, ob das Attribut PUBLIC bei der Bildung der Slices berücksichtigt wird.

PUBLIC = *NO

Das Attribut PUBLIC wird bei der Bildung der Slices nicht berücksichtigt.

PUBLIC = *YES(...)

Der BINDER bildet für CSECTs mit unterschiedlichem Attribut PUBLIC getrennte Slices.

SUBSYSTEM-ENTRIES =

Legt die Symbole (CSECT oder ENTRY) der PUBLIC-Slice fest, die zur Befriedigung von Externverweisen genutzt werden können, wenn die PUBLIC-Slice als dynamisches Subsystem (siehe Handbuch „Einführung in die Systembetreuung“ [9]) geladen wurde.

SUBSYSTEM-ENTRIES = *NONE

Keine Symbole aus diesem Subsystem (der PUBLIC-Slice) werden zur Befriedigung von Externverweisen genutzt.

SUBSYSTEM-ENTRIES = <text 1..32>

Name der CSECT oder des ENTRY in der als Subsystem geladenen PUBLIC-Slice, der zur Befriedigung von Externverweisen genutzt werden kann.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

RESIDENCY-MODE = *UNCHANGED / *NO / *YES

Legt fest, ob das Attribut RMODE bei der Bildung der Slices berücksichtigt wird.

Bei Angabe YES bildet der BINDER für CSECTs und mit unterschiedlichem Attribut RMODE getrennte Slices.

SLICE-DEFINITION = *BY-USER(...)

Die physische Struktur des LLM wird vom Benutzer mit Anweisungen SET-USER-SLICE-POSITION festgelegt (User defined Slices). Dabei können Überlagerungssegmente (Overlays) festgelegt werden.

AUTOMATIC-CONTROL = *UNCHANGED / *NO / *YES(...)

Hat nur bei Überlagerungsstrukturen (Overlays) eine Bedeutung.

Gibt an, ob ein Steuermodul (Overlay Control Module, OCM) in das erzeugte LLM eingebunden wird, um das automatische Nachladen der Overlays zu steuern.

AUTOMATIC-CONTROL = *YES(...)

Ein OCM wird in das erzeugte LLM eingebunden, um das automatische Nachladen der Overlays zu steuern.

RELOAD-SLICE=

Legt fest, ob das geladene Slice nachgeladen wird, wobei das vorherige Slice im Hauptspeicher überschrieben wird.

RELOAD-SLICE= *YES

Ein bereits geladenes Slice wird nachgeladen, wobei das vorherige Slice im Hauptspeicher überschrieben wird.

RELOAD-SLICE= *NO

Ein bereits geladenes Slice wird unverändert im Hauptspeicher behalten.

EXCLUSIVE-SLICE-CALL = *UNCHANGED / *NO / *YES

Ist nur bei Überlagerungsstrukturen von Bedeutung und legt fest, ob Externverweise zwischen exklusiven Slices befriedigt werden sollen.

EXCLUSIVE-SLICE-CALL = *NO

gibt an, dass der BINDER Externverweise nur meldet und sie nicht befriedigt, falls er Verweise zwischen exklusiven Slices feststellt.

EXCLUSIVE-SLICE-CALL = *YES

veranlasst den BINDER, Externverweise zwischen exklusiven Slices zu befriedigen, d.h. der Benutzer nimmt evtl. auftretende Fehler in Kauf.

COPYRIGHT = *UNCHANGED / *PARAMETERS(...) / *NONE

Legt die neue Copyright-Information fest. Die Copyright-Information besteht aus Text und Jahreszahl.

COPYRIGHT = *PARAMETERS(...)**NAME = *UNCHANGED / *SYSTEM-DEFAULT / <c-string 1..64>**

Bezeichnet den neuen Text der Copyright-Information.

NAME = *SYSTEM-DEFAULT

Der Wert des Klasse-2-Systemparameters BLSCOPYN soll übernommen werden. Dieser Wert wird bei der Systeminstallation festgelegt (siehe Handbuch „Einführung in die Systembetreuung“ [9]).

NAME = <c-string 1..64>

Neuer Text der Copyright-Information. Besteht der Text aus Leerzeichen, wird keine Copyright-Information übernommen.

YEAR = *UNCHANGED / *CURRENT / <integer 1900..2100>

Bezeichnet die Jahreszahl.

YEAR = *CURRENT

Aktuelle Jahreszahl.

YEAR = <integer 1900..2100>

Explizite Angabe der Jahreszahl.

COPYRIGHT = *NONE

Keine neue Copyright-Information wird übernommen.

INCLUSION-DEFAULTS =

Legt die Verwendung der Strukturinformation und der LSD-Information fest. Dies ist der Standardwert, der in den Anweisungen INCLUDE-MODULES, REPLACE-MODULES und RESOLVE-BY-AUTOLINK desselben *Edit-Laufs* benutzt wird, falls in diesen Anweisungen keine spezifischen Werte angegeben sind.

Strukturinformation und LSD-Information werden beim Speichern des LLM nur übernommen, wenn dies sowohl in der Anweisung SAVE-LLM wie in vorhergehenden Anweisungen INCLUDE-MODULES, REPLACE-MODULES oder RESOLVE-BY-AUTOLINK verlangt wird.

INCLUSION-DEFAULTS = *PARAMETERS(...)**LOGICAL-STRUCTURE = *UNCHANGED / *WHOLE-LLM / *OBJECT-MODULES**

Legt fest, ob beim Einfügen oder Ersetzen von Modulen die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird in das aktuelle LLM übernommen.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Die Strukturinformation wird nicht übernommen. Im aktuellen LLM wird nur eine Struktur mit Bindemodulen (OMs) aufgebaut.

TEST-SUPPORT = *UNCHANGED / *NO / *YES

Legt fest, ob beim Einfügen oder Ersetzen von Modulen die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

MODIFY-MAP-DEFAULTS

Ändern der Standardwerte für die Ausgabe von Listen

Diese Anweisung ändert die Standardwerte für eine nachfolgende Anweisung SHOW-MAP und für die nächste folgende Anweisung SAVE-LLM mit dem Operanden MAP=YES.

Der Geltungsbereich für alle Operanden bezieht sich auf einen *BINDER-Lauf*.

Eine neue Anweisung MODIFY-MAP-DEFAULTS überschreibt die Operandenwerte einer vorhergehenden mit der gleichen MAP-NAME-Angabe.

Durch Angabe des Operanden MAP-NAME kann der Benutzer verschiedene benannte Listen mit verschiedenen Standardwerten definieren. Über den Namen, der durch den Operanden MAP-NAME festgelegt wurde, kann der Benutzer im selben BINDER-Lauf beim Ausführen der Anweisung SHOW-MAP auf die vordefinierten Listen zugreifen.

Der Wert ***UNCHANGED** in den entsprechenden Operanden bedeutet, dass der Wert gilt, der in einer vorangegangenen Anweisung MODIFY-MAP-DEFAULTS in demselben BINDER-Lauf und mit gleicher MAP-NAME-Angabe für den Operanden festgelegt wurde. In der ersten Anweisung MODIFY-MAP-DEFAULTS wird für **UNCHANGED* der *erste* auf den Wert **UNCHANGED* folgende Operandenwert angenommen.

MODIFY-MAP-DEFAULTS

```

MAP-NAME = *STD / <structured-name 1..32>
,USER-COMMENT = *UNCHANGED / *NONE / <c-string 1..255 with-low>
,HELP-INFORMATION = *UNCHANGED / *YES / *NO
,GLOBAL-INFORMATION = *UNCHANGED / *YES / *NO
,LOGICAL-STRUCTURE = *UNCHANGED / *YES(...) / *NO
  *YES(...)
    |
    | RESOLUTION-SCOPE = *UNCHANGED / *YES / *NO
    | ,HSI-CODE = *UNCHANGED / *YES / *NO / *X86
,PHYSICAL-STRUCTURE = *UNCHANGED / *YES / *NO

```

Fortsetzung ➡

```

,PROGRAM-MAP = *UNCHANGED / *PARAMETERS(...) / *NO
  *PARAMETERS(...)
    |
    | DEFINITIONS = *UNCHANGED / *ALL / *NONE / list-poss(5): *MODULE / *CSECT /
    | *ENTRY / *COMMON / *XDSECT-D
    | ,INVERTED-XREF-LIST = *UNCHANGED / *NONE / *ALL / list-poss(4): *EXTRN / *VCON /
    | *WXTRN / *XDSECT-R
    | ,REFERENCES = *UNCHANGED / *ALL / *NONE / list-poss(4): *EXTRN / *VCON / *WXTRN /
    | *XDSECT-R
,UNRESOLVED-LIST = *UNCHANGED / *SORTED(...) / *YES(...) / *NO
  *SORTED(...)
    |
    | WXTRN = *YES / *NO
    | ,NOREF = *NO / *YES
  *YES(...)
    |
    | WXTRN = *YES / *NO
    | ,NOREF = *NO / *YES
,SORTED-PROGRAM-MAP = *UNCHANGED / *NO / *YES
,PSEUDO-REGISTER = *UNCHANGED / *NO / *YES
,UNUSED-MODULE-LIST = *UNCHANGED / *NO / *YES
,DUPLICATE-LIST = *UNCHANGED / *NO / *YES(...)
  *YES(...)
    |
    | INVERTED-XREF-LIST = *YES / *NO
,MERGED-MODULES = *UNCHANGED / *YES / *NO
,INPUT-INFORMATION = *UNCHANGED / *YES / *NO
,STATEMENT-LIST = *UNCHANGED / *NO / *YES
,OUTPUT = *UNCHANGED / *SYSLST(...) / *BY-SHOW-FILE(...) / <filename 1..54 without-gen-vers>(...) /
  *LINK(...) / *EXIT-ROUTINE(...)
  *SYSLST(...)
    |
    | SYSLST-NUMBER = *STD / <integer 1..99>
    | ,LINES-PER-PAGE = 64 / <integer 10..2147483647> / *IGNORED
    | ,LINE-SIZE = 136 / <integer 132..255>

```

Fortsetzung ➡

```

*BY-SHOW-FILE(...)
  | FILE-NAME = *STD / <filename 1..54 without-gen-vers>
  | ,DELETE-FILE = *YES / *NO
  | ,LINE-SIZE = 136 / <integer 132..255>
<filename>(...)
  | LINE-SIZE = 136 / <integer 132..255>
*LINK(...)
  | LINK-NAME = BNDMAP / <structured-name 1..8> / <filename 1..8 without-gen>
  | ,LINE-SIZE = 136 / <integer 132..255>
*EXIT-ROUTINE(...)
  | ROUTINE-NAME = <c-string 1..32 with-low> / <text 1..32>
  | ,LIBRARY = *BLSLIB-LINK / <filename 1..54 without-gen-vers> / *LINK(...)
    | *LINK(...)
      | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
    | ,FILE-NAME = *STD / <filename 1..54 without-gen-vers>
    | ,LINE-SIZE = 136 / <integer 132..255>
    | ,USER-PARAMETERS = *NONE / <c-string 1..255 with-low> / <text 1..255>

```

MAP-NAME =

Gibt den Namen der Liste an, für die die neuen Standardwerte gelten sollen.

MAP-NAME = *STD

Die Werte gelten für die Standardliste mit dem Namen `BNDMAP.date.time.<tsn>`. D.h. sie gelten für die Liste, die mit `//SHOW-MAP MAP-NAME=*STD` oder mit `//SAVE-LLM ... MAP=YES` ausgegeben wird.

MAP-NAME = <structured-name 1..32>

Die Standardwerte werden für eine vom Benutzer definierte und benannte Liste festgelegt, die mit `//SHOW-MAP MAP-NAME=<structured-name 1..32>` ausgegeben werden kann.

Bedeutung der übrigen Operanden siehe Anweisung SHOW-MAP, [Seite 314ff.](#)

MODIFY-MODULE-ATTRIBUTES

Ändern der Attribute von Modulen

Diese Anweisung erlaubt die Änderung der logischen Struktur eines LLM sowie der Laufzeit-, Test- und Diagnoseinformationen eines Moduls.

MODIFY-MODULE-ATTRIBUTES

```

NAME = <c-string 1..32 with-low> / <text 1..32>
, PATH-NAME = *CURRENT-SUB-LLM / *NONE / <c-string 1..255 with-low> / <text 1..255>
, NEW-NAME = *UNCHANGED / *INTERNAL / *ELEMENT-NAME / <c-string 1..32 with-low> / <text 1..32>
, NEW-PATH-NAME = *UNCHANGED / *CURRENT-SUB-LLM / <c-string 1..255 with-low> / <text 1..255>
, TEST-SUPPORT = *UNCHANGED / *INCLUSION-DEFAULT / *NO / *YES
, RUN-TIME-VISIBILITY = *UNCHANGED / *NO / *YES
, RESOLUTION-SCOPE = *UNCHANGED / *STD / *PARAMETERS(...)
  *PARAMETERS(...)
    | HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255>
    | , LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255>
    | , FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255>
, NAME-COLLISION = *STD / *IGNORED / *WARNING(...) / *ERROR(...)
  *WARNING(...)
    | SCOPE = *WHOLE-LLM / *SLICE
  *ERROR(...)
    | SCOPE = *WHOLE-LLM / *SLICE

```

NAME = <structured-name 1..32> / <text 1..32>

Legt den Namen des Sub-LLM fest, der verändert werden soll.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

PATH-NAME =

Legt den logisch übergeordneten Knoten des zu verarbeitenden Sub-LLM fest. Standardmäßig ist das das aktuelle Sub-LLM im BINDER-Arbeitsbereich (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

PATH-NAME = ***CURRENT-SUB-LLM**

Das aktuelle Sub-LLM wird angenommen

(siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

PATH-NAME = *NONE

Das ganze LLM soll verändert werden.

PATH-NAME = <text 1..255>

Pfadname des Sub-LLM in der logischen Struktur des aktuellen LLM.

Achtung: Der BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

NEW-NAME =

Gibt an, welchen neuen Namen das (Sub-)LLM erhalten soll.

NEW-NAME = *UNCHANGED

Der Name des (Sub-)LLMs wird nicht geändert.

NEW-NAME = *INTERNAL

Der interne Name des (Sub-)LLM gilt als neuer Name.

NEW-NAME = *ELEMENT-NAME

Als neuer Name wird der Name verwendet, den das Sub-LLM als Bibliotheksname besitzt. Dieser Wert ist nur erlaubt, wenn das Modul tatsächlich als Bibliothekselement vorhanden ist.

NEW-NAME = <structured-name 1..32>

Explizite Angabe des neuen Sub-LLM-Namens.

NEW-PATH-NAME =

Gibt den neuen Pfadnamen des betrachteten Sub-LLMs an und ermöglicht damit das Ändern der logischen Struktur des LLMs.

NEW-PATH-NAME = *UNCHANGED

Der Pfadname des Sub-LLMs wird nicht geändert. Die logische Struktur des LLM wird demzufolge nicht verändert.

NEW-PATH-NAME = *CURRENT-SUB-LLM

Der neue Pfadname wird als Pfadname des aktuellen Sub-LLMs eingestellt (siehe [Seite 18](#)).

NEW-PATH-NAME = <text 1..255>

Explizite Angabe des neuen Pfadnamens.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

TEST-SUPPORT =

Legt fest, ob die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.



Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *UNCHANGED

Die LSD-Information wird nicht geändert.

TEST-SUPPORT = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben *Edit-Laufs* angenommen.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

RUN-TIME-VISIBILITY =

Legt fest, ob das Modul als Laufzeitmodul betrachtet werden soll. Alle Symbole eines Laufzeitmoduls werden beim Abspeichern maskiert und zunächst nicht mehr zur Befriedigung von Externverweisen verwendet. Bei einem späteren Lesezugriff auf das Modul (z.B. bei START-LLM-UPDATE oder INCLUDE-MODULES) wird die Maskierung wieder rückgängig gemacht.

RUN-TIME-VISIBILITY = *UNCHANGED

Der Wert wird nicht geändert.

RUN-TIME-VISIBILITY = *NO

Das Modul soll nicht als Laufzeitmodul betrachtet werden.

RUN-TIME-VISIBILITY = *YES

Das Modul soll als Laufzeitmodul betrachtet werden. Alle Symbole des Moduls werden beim Abspeichern maskiert.

RESOLUTION-SCOPE =

Legt Prioritätsklassen fest, die steuern, in welcher Reihenfolge der BINDER andere Module bei der Befriedigung von Externverweisen durchsuchen soll. Für jede Klasse müssen zwei Werte unterschieden werden:

- der dynamische Wert.
Er beeinflusst die Reihenfolge, in der Module zur Befriedigung von Externverweisen durchsucht werden. Er wird jedoch nicht im LLM abgespeichert.
- der statische Wert.
Er wird im LLM abgespeichert und bildet die Basis zur Ermittlung des dynamischen Wertes.

RESOLUTION-SCOPE = *UNCHANGED

Die statischen Werte der Prioritätsklassen sind die in den betroffenen Modulen gespeicherten Werte. Für Objektmodule (OM) erhalten sie den Wert *STD (siehe unten).

RESOLUTION-SCOPE = *STD

Die statischen Werte der Prioritätsklassen sind *STD. Das bedeutet, dass die dynamischen Werte vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen werden. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Die Voreinstellung für die statischen Werte des ROOT-Knotens eines LLMs ist *STD. Die dynamischen Werte sind in diesem Fall *NONE (siehe unten).

RESOLUTION-SCOPE = *PARAMETERS(...)

Die statischen Werte der einzelnen Prioritätsklassen werden separat vereinbart.

HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /**<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen vor allen anderen durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)).

HIGH-PRIORITY-SCOPE = *STD

Der statische Wert dieser Prioritätsklasse ist *STD. Das bedeutet, dass der dynamische Wert vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen wird. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Der statische Wert des ROOT-Knotens eines LLMs ist standardmäßig *STD. Der dynamische Wert ist in diesem Fall *NONE (siehe unten).

HIGH-PRIORITY-SCOPE = *NONE

Die Prioritätsklasse HIGH-PRIORITY-SCOPE ist nicht definiert. D.h. es gibt keine Module, die vor allen anderen zum Befriedigen von Externverweisen durchsucht werden sollen. Der Wert des übergeordneten Knotens wird nicht übernommen.

HIGH-PRIORITY-SCOPE = <c-string 1..255 with-low> / <text 1..255>

Pfadname des Sub-LLMs, das zum Befriedigen von Externverweisen zuerst durchsucht werden soll.

LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /**<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen erst dann durchsucht werden soll, wenn die Suche in allen anderen Modulen erfolglos war (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE /**<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen nicht durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

Ein Beispiel zum Vererbungsmechanismus für die PRIORITY-SCOPE-Werte finden Sie bei der Beschreibung der Anweisung INCLUDE-MODULES auf [Seite 222](#).

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten. Ein Namenskonflikt kann während dieser Anweisung nur dann auftreten, wenn für RUN-TIME-VISIBILITY=YES angegeben wurde.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert des Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Die Ausführung der Anweisung wird abgebrochen, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

MODIFY-STD-DEFAULTS

Ändern von Standardwerten

Diese Anweisung ändert globale Standardwerte für einen BINDER-Lauf.

MODIFY-STD-DEFAULTS

```

OVERWRITE = *UNCHANGED / *YES / *NO
, FOR-BS2000-VERSIONS = *UNCHANGED / *FROM-CURRENT(...) / *FROM-V10(...) /
                        *FROM-OSD-V1(...) / *FROM-OSD-V3(...) / *FROM-OSD-V4(...)
  *FROM-CURRENT(...)
    | CONNECTION-MODE = *OSD-DEFAULT / *BY-RELOCATION / *BY-RESOLUTION
  *FROM-V10(...)
    | CONNECTION-MODE = *BY-RESOLUTION / *BY-RELOCATION / *BY-RESOLUTION
  *FROM-OSD-V1(...)
    | CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION
  *FROM-OSD-V3(...)
    | CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION
  *FROM-OSD-V4(...)
    | CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION
, REQUIRED-COMPRESSION = *UNCHANGED / *NO / *YES

```

Fortsetzung ➡

```

,NAME-COLLISION = *UNCHANGED / *PARAMETERS(...)
  *PARAMETERS(...)
    INCLUSION = *UNCHANGED / *IGNORED / *WARNING(...) / *ERROR(...)
      *WARNING(...)
        | SCOPE = *WHOLE-LLM / *SLICE
      *ERROR(...)
        | SCOPE = WHOLE-LLM / *SLICE
    ,SAVE = *UNCHANGED / *IGNORED / *WARNING(...) / *ERROR(...)
      *WARNING(...)
        | SCOPE = *WHOLE-LLM / *SLICE
      *ERROR(...)
        | SCOPE = WHOLE-LLM / *SLICE
    ,SYMBOL-PROCESSING = *UNCHANGED / *IGNORED / *WARNING(...) / *ERROR(...)
      *WARNING(...)
        | SCOPE = *WHOLE-LLM / *SLICE
      *ERROR(...)
        | SCOPE = *WHOLE-LLM / *SLICE

```

OVERWRITE =

Gibt an, ob Überschreiben erlaubt ist oder nicht.

OVERWRITE = *UNCHANGED

Der BINDER übernimmt den Wert des Operanden aus der letzten Anweisung MODIFY-STD-DEFAULTS. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert *YES.

OVERWRITE = *YES

Überschreiben ist erlaubt.

OVERWRITE = *NO

Überschreiben ist nicht erlaubt.

FOR-BS2000-VERSIONS =

Gibt an, in welcher BS2000 OSD/BC-Version das erzeugte LLM durch den DBL geladen werden soll. Das LLM kann von DBL nur in der angegebenen oder einer höheren Version von BS2000 OSD/BC verarbeitet werden.

FOR-BS2000-VERSIONS = *BY-PROGRAM

BINDER legt das Format des erzeugten LLMs anhand von dessen Inhalt fest. Das LLM-Format ist immer das niedrigstmögliche, das die benötigte Funktionalität gewährleistet. Beispielsweise bekommt ein LLM, der RISC-Code enthält, das Format 3, während ein LLM, das komprimierten Text enthält, im Format 2 erzeugt wird.

FOR-BS2000-VERSIONS = *UNCHANGED

Der BINDER übernimmt den Wert des Operanden aus der letzten Anweisung MODIFY-STD-DEFAULTS. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert *FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-CURRENT(...)

Die Version von BS2000 OSD/BC, unter der BINDER gerade abläuft, wird übernommen.

CONNECTION-MODE =

Gibt die Art der Verknüpfung zwischen dem PRIVATE- und dem PUBLIC-Teil an. Dieser Operand ist nur von Bedeutung, wenn das LLM gemäß der PUBLIC-Attribute der darin enthaltenen CSECTS in Slices aufgeteilt wurde.

CONNECTION-MODE = *OSD-DEFAULT

Diese Angabe wird aus Kompatibilitätsgründen unterstützt. Sie ist gleichbedeutend mit CONNECTION-MODE = *BY-RELOCATION.

CONNECTION-MODE = *BY-RELOCATION

Die Verknüpfung zwischen PRIVATE- und PUBLIC-Teil erfolgt durch Relativieren.

CONNECTION-MODE = *BY-RESOLUTION

Die Verknüpfung zwischen PRIVATE- und PUBLIC-Teil erfolgt durch Auflösen.

FOR-BS2000-VERSIONS = *FROM-V10(...)

Das LLM kann in allen Versionen von BS2000 OSD/BC geladen werden.

CONNECTION-MODE = *BY-RESOLUTION / *BY-RELOCATION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-OSD-V1(...)

Das LLM kann in allen Versionen von BS2000 OSD/BC geladen werden.

CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-OSD-V3(...)

Das LLM kann ab BS2000/OSD-BC V3.0 geladen werden.

CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-OSD-V4(...)

Das LLM kann mit BLSSERV ab BS2000/OSD-BC V3.0 geladen werden.

CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

REQUIRED-COMPRESSION =

Gibt an, ob zur besseren Ausnutzung der Plattenkapazität eine Komprimierung der Daten durchgeführt werden soll.

REQUIRED-COMPRESSION = *UNCHANGED

Der BINDER übernimmt den Wert aus der letzten Anweisung MODIFY-STD-DEFAULTS. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert NO.

REQUIRED-COMPRESSION = *NO

Eine Komprimierung der Daten wird nicht durchgeführt.

REQUIRED-COMPRESSION = *YES

Eine Komprimierung der Daten wird durchgeführt.

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen.

NAME-COLLISION = *UNCHANGED

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so werden Namenskonflikte nicht behandelt.

NAME-COLLISION = *PARAMETERS(...)**INCLUSION =**

Gibt an, wie Namenskonflikte zu behandeln sind, die beim Einfügen von Modulen auftreten.

INCLUSION = *UNCHANGED

Der BINDER übernimmt den Wert von demselben Operanden der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in diesem Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert *IGNORED.

INCLUSION = *IGNORED

Gibt an, dass Namenskonflikte beim Einfügen von Modulen nicht behandelt werden.

INCLUSION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es beim Einfügen von Modulen zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

INCLUSION = *ERROR(...)

Legt fest, dass das Einfügen von Modulen abgebrochen wird, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE = *WHOLE-LLM / *SLICE

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten. Zur Bedeutung des Operanden siehe WARNING.

SAVE =

Gibt an, wie Namenskonflikte zu behandeln sind, die beim Abspeichern eines LLM auftreten.

SAVE = *UNCHANGED

Der BINDER übernimmt den Wert von demselben Operanden der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in diesem Edit-Lauf noch nicht angegeben, so wird das LLM nicht auf Namenskonflikte geprüft.

SAVE = *IGNORED

Gibt an, dass beim Abspeichern eines LLM nicht auf Namenskonflikte geprüft wird.

SAVE = *WARNING(...)

Der Benutzer erhält eine Warnung, falls beim Abspeichern des LLM Namenskonflikte entdeckt werden.

SCOPE = *WHOLE-LLM / *SLICE

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten. Zur Bedeutung des Operanden siehe INCLUSION=WARNING(...).

SAVE = ERROR(...)

Legt fest, dass das Abspeichern des LLM abgebrochen wird, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE = *WHOLE-LLM / *SLICE

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten. Zur Bedeutung des Operanden siehe INCLUSION=WARNING(...).

SYMBOL-PROCESSING =

Gibt an, wie Namenskonflikte zu behandeln sind, die bei der Behandlung von Symbolen auftreten.

SYMBOL-PROCESSING = *UNCHANGED

Der BINDER übernimmt den Wert von demselben Operanden der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde die Anweisung in diesem Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

SYMBOL-PROCESSING = *IGNORED

Gibt an, dass die Anweisung zur Symbolbehandlung ohne Prüfung auf Namenskonflikte abgearbeitet wird.

SYMBOL-PROCESSING = *WARNING(...)

Der Benutzer erhält eine Warnung, falls bei der Behandlung von Symbolen Namenskonflikte entdeckt werden.

SCOPE = *WHOLE-LLM / *SLICE

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten. Zur Bedeutung des Operanden siehe INCLUSION=WARNING(...).

SYMBOL-PROCESSING = *ERROR(...)

Legt fest, dass die Anweisung zur Symbolbehandlung abgebrochen wird, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE = *WHOLE-LLM / *SLICE

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten. Zur Bedeutung des Operanden siehe INCLUSION=WARNING(...).

MODIFY-SYMBOL-ATTRIBUTES

Ändern der Attribute von Symbolen

Diese Anweisung ändert die Attribute von Programmabschnitten (CSECTs) und COMMON-Bereichen im aktuellen LLM.

Folgende Attribute können geändert werden (siehe Handbuch „ASSEMBH“ [2]):

- Speicherresidenz (RESIDENT),
- gemeinsam benutzbar (PUBLIC),
- nur Lesezugriff (READ-ONLY),
- Ausrichtung (ALIGNMENT),
- Adressierungsmodus (AMODE),
- Residenzmodus (RMODE).

MODIFY-SYMBOL-ATTRIBUTES

```

SYMBOL-NAME = *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
,SYMBOL-TYPE = *DEFINITIONS / list-poss(2): *CSECT / *COMMON
,SCOPE = *CURRENT-SUB-LLM / *EXPLICIT(...) / *WHOLE-LLM
  *EXPLICIT(...)
    |
    | WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <c-string 1..255 with-low> /
    |                   <text 1..255>
    |
    | ,EXCEPT-SUB-LLM = *NONE / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
,RESIDENT = *UNCHANGED / *YES / *NO
,PUBLIC = *UNCHANGED / *YES / *NO
,READ-ONLY = UNCHANGED / *YES / *NO
,ALIGNMENT = *UNCHANGED / *DOUBLE-WORD / *PAGE / *BUNDLE / <integer 3..12>
,ADDRESSING-MODE = *UNCHANGED / *24 / *31 / *ANY
,RESIDENCY-MODE = *UNCHANGED / *24 / *ANY

```

SYMBOL-NAME =

Gibt die Namen der CSECTs und COMMON-Bereiche an, deren Attribute geändert werden.

SYMBOL-NAME = *ALL

Die Attribute von allen CSECTs und COMMON-Bereichen werden geändert.

SYMBOL-NAME = <c-string 1..255>

Mit Platzhalter (Wildcards) angegebene Namen der Symbole, deren Attribute geändert werden.

Achtung: BINDER prüft Sonderdatentyp <symbol-with-wild> (siehe [Seite 202](#)).

SYMBOL-NAME = <text 1..32>

Explizite Angabe der Namen der Symbole, deren Attribute geändert werden.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

SYMBOL-TYPE = *DEFINITIONS / list-poss(2): *CSECT / *COMMON

Wählt aus, ob Attribute nur von CSECTs, nur von COMMON-Bereichen oder von beiden geändert werden.

SCOPE =

Setzt einen oder mehrere Zeiger. Sie verweisen auf die Sub-LLMs in der logischen Struktur des LLM, in denen Attribute geändert werden.

SCOPE = *CURRENT-SUB-LLM

Der Zeiger verweist auf das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

SCOPE = *EXPLICIT(...)**WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <text 1..255>**

Die Zeiger verweisen auf die explizit angegebenen Sub-LLMs. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *WHOLE-LLM werden alle Sub-LLMs angesprochen.

EXCEPT-SUB-LLM = *NONE / list-poss(10): <text 1..255>

Erlaubt es, aus der im Operanden WITHIN-SUB-LLM angegebenen Liste einzelne Zeiger auszuschließen. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben, die nicht berücksichtigt werden sollen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *NONE werden keine Zeiger ausgeschlossen.

SCOPE = *WHOLE-LLM

Alle Sub-LLMs des aktuellen LLM werden beim Ändern von Attributen berücksichtigt.

RESIDENT =

Legt fest, ob das Attribut RESIDENT geändert wird.

RESIDENT = *UNCHANGED

Der bisherige Wert für das Attribut RESIDENT bleibt erhalten.

RESIDENT = *YES

Die angegebenen Symbole sind im Hauptspeicher resident (Klasse-3-Speicher).

RESIDENT = *NO

Die angegebenen Symbole sind seitenwechselbar.

PUBLIC =

Legt fest, ob das Attribut PUBLIC geändert wird.

PUBLIC = *UNCHANGED

Der bisherige Wert für das Attribut PUBLIC bleibt erhalten.

PUBLIC = *YES

Die angegebenen Symbole sind gemeinsam benutzbar.

PUBLIC = *NO

Die angegebenen Symbole sind nicht gemeinsam benutzbar.

READ-ONLY =

Legt fest, ob das Attribut READ-ONLY geändert wird.

READ-ONLY = *UNCHANGED

Der bisherige Wert für das Attribut READ-ONLY bleibt erhalten.

READ-ONLY = *YES

Für die angegebenen Symbole ist nur Lesezugriff erlaubt. Für ein Vektorprogramm ist die Angabe *YES nicht zulässig.

READ-ONLY = *NO

Für die angegebenen Symbole sind Lese- und Schreibzugriff erlaubt.

ALIGNMENT =

Legt fest, ob die Ausrichtung (Alignment) geändert wird oder nicht.

ALIGNMENT = *UNCHANGED

Der bisherige Wert für die Ausrichtung bleibt erhalten.

ALIGNMENT = *DOUBLE-WORD

Die angegebenen Symbole werden auf Doppelwortgrenze ausgerichtet.

ALIGNMENT = *PAGE

Die angegebenen Symbole werden auf Seitengrenze ausgerichtet, d.h. die Adresse ist ein Vielfaches von 4096 (X'1000').

ALIGNMENT = *BUNDLE

Die angegebenen Symbole werden auf Doppelwortgrenze ausgerichtet, d.h. die Adresse ist ein Vielfaches von 16.

ALIGNMENT = <integer 3..12>

Die angegebenen Symbole werden auf eine Adresse ausgerichtet, die ein Vielfaches von 2^n ist. Der Exponent „n“ wird mit <integer 3..12> festgelegt.

ADDRESSING-MODE =

Legt fest, ob der Adressierungsmodus (AMODE) geändert wird. Wenn eine Inkonsistenz zwischen den ursprünglichen und den angegebenen Werten von AMODE und RMODE auftritt, wird der Operand ignoriert.

ADDRESSING-MODE = *UNCHANGED

Der bisherige Wert für den Adressierungsmodus bleibt erhalten.

ADDRESSING-MODE = *24

Den angegebenen Symbolen wird der 24-Bit-Adressierungsmodus zugeordnet.

ADDRESSING-MODE = *31

Den angegebenen Symbolen wird der 31-Bit-Adressierungsmodus zugeordnet.

ADDRESSING-MODE = *ANY

Den angegebenen Symbolen wird der 24- oder 31-Bit-Adressierungsmodus zugeordnet. Die Entscheidung über den Adressierungsmodus wird erst beim Laden getroffen.

RESIDENCY-MODE =

Legt fest, ob der Residenzmodus (RMODE) geändert wird oder nicht. Wenn eine Inkonsistenz zwischen den ursprünglichen und den angegebenen Werten von AMODE und RMODE auftritt, wird der Operand ignoriert.

RESIDENCY-MODE = *UNCHANGED

Der bisherige Wert für den Adressierungsmodus bleibt erhalten.

RESIDENCY-MODE = *24

Die angegebenen Symbole dürfen nur unterhalb 16 Mbyte geladen werden.

RESIDENCY-MODE = *ANY

Die angegebenen Symbole dürfen unterhalb und oberhalb 16 Mbyte geladen werden.

MODIFY-SYMBOL-TYPE

Ändern des Typs von Symbolen

Diese Anweisung legt den Typ von Symbolen fest. Unter „Symbolen“ sind hier nur Externverweise (EXTRN), bedingte Externverweise (WXTRN) und V-Konstanten (VCON) zu verstehen. Folgende Umwandlungen sind möglich:

EXTRN in WXTRN oder VCON,

WXTRN in EXTRN oder VCON,

VCON in EXTRN oder WXTRN.

EXTRN und WXTRN, die nicht referenziert sind, können nicht in VCON umgewandelt werden. Der Geltungsbereich der Festlegungen kann mit dem Operanden SCOPE eingeschränkt werden.

Auf die frühere Befriedigung von Externverweisen hat die Anweisung keinen Einfluss.

MODIFY-SYMBOL-TYPE

```

SYMBOL-NAME = *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
, SYMBOL-TYPE = *REFERENCES / list-poss(3): *EXTRN / *VCON / *WXTRN
, SCOPE = *CURRENT-SUB-LLM / *EXPLICIT(...) / *WHOLE-LLM
  *EXPLICIT(...)
    |
    | WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <c-string 1..255 with-low> /
    |                   <text 1..255>
    |
    | , EXCEPT-SUB-LLM = *NONE / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
    |
NEW-SYMBOL-TYPE = *EXTRN / *VCON / *WXTRN

```

SYMBOL-NAME =

Gibt die Symbole an, deren Typ mit der Anweisung geändert werden soll.

SYMBOL-NAME = *ALL

Alle Symbole in dem mit SCOPE definierten Geltungsbereich werden bearbeitet.

SYMBOL-NAME = list-poss(40): <c-string 1..255> / <text 1..32>

Achtung: BINDER prüft Sonderdatentyp <symbol> und <symbol-with-wild> (siehe [Seite 202](#)).

SYMBOL-TYPE = *REFERENCES / list-poss(3): *EXTRN / *VCON / *WXTRN

Gibt an, welche Typen von Externverweisen verarbeitet werden sollen. Bei Angabe von REFERENCES werden alle Externverweise bearbeitet.

SCOPE =

Definiert den Geltungsbereich durch Setzen eines oder mehrerer Zeiger. Die Zeiger verweisen auf die Sub-LLMs in der logischen Struktur des LLM, in denen der Typ von Symbolen geändert werden soll.

SCOPE = *CURRENT-SUB-LLM

Der Zeiger verweist auf das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

SCOPE = *EXPLICIT(...)**WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <text 1..255>**

Die Zeiger verweisen auf die explizit angegebenen Sub-LLMs. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *WHOLE-LLM werden alle Sub-LLMs angesprochen.

EXCEPT-SUB-LLM = *NONE / list-poss(10): <text 1..255>

Erlaubt es, aus der im Operanden WITHIN-SUB-LLM angegebenen Liste einzelne Zeiger auszuschließen. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben, die nicht berücksichtigt werden sollen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *NONE werden keine Zeiger ausgeschlossen.

SCOPE = *WHOLE-LLM

Alle Sub-LLMs des aktuellen LLM werden beim Ändern des Typs von Symbolen berücksichtigt.

NEW-SYMBOL-TYPE = *EXTRN / *VCON / *WXTRN

Gibt den neuen Typ an, den die Symbole erhalten sollen.

MODIFY-SYMBOL-VISIBILITY

Ändern der Maskierung von Symbolen

Diese Anweisung legt fest, in welchem Umfang Programmabschnitte (CSECTs) und Einsprungstellen (ENTRYS) im aktuellen LLM für einen späteren BINDER- oder Bindelader-Lauf sichtbar bleiben oder maskiert werden sollen. Maskierte Symbole werden bei einer späteren Verarbeitung durch BINDER oder DBL nicht mehr zum Befriedigen von Extern-verweisen gefunden.

MODIFY-SYMBOL-VISIBILITY

```

SYMBOL-NAME = *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
, SYMBOL-TYPE = *DEFINITIONS / list-poss(3): *CSECT / *ENTRY / *COMMON
, SCOPE = *CURRENT-SUB-LLM / *EXPLICIT(...) / *WHOLE-LLM
  *EXPLICIT(...)
    | WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <c-string 1..255 with-low> /
    | <text 1..255>
    | , EXCEPT-SUB-LLM = *NONE / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
, VISIBLE = *YES / *NO(...)
  *NO(...)
    | KEEP-RESOLUTION = *YES / *NO
, NAME-COLLISION = STD / *IGNORED / *WARNING(...) / *ERROR(...)
  *WARNING(...)
    | SCOPE = *WHOLE-LLM / *SLICE
  *ERROR(...)
    | SCOPE = *WHOLE-LLM / *SLICE

```

SYMBOL-NAME =

Legt die Symbole fest, deren Maskierung geändert wird.

SYMBOL-NAME = *ALL

Die Maskierung von allen Symbolen wird geändert.

SYMBOL-NAME = list-poss(40): <c-string 1..255>/ <text 1..32>

Explizite Angabe der Symbole, deren Maskierung geändert wird. Die Angabe von Platzhaltern (Wildcards) ist zulässig.

Achtung: BINDER prüft Sonderdatentyp <symbol> und <symbol-with-wild> (siehe [Seite 202](#)).

SYMBOL-TYPE = *DEFINITIONS / list-poss(3): *CSECT / *ENTRY / *COMMON

Legt den Typ der Symbole fest, deren Maskierung geändert wird. Bei Angabe DEFINITIONS wird die Maskierung von allen angegebenen Symbolen geändert.

SCOPE =

Setzt einen oder mehrere Zeiger. Sie verweisen auf die Sub-LLMs in der logischen Struktur des LLM, in denen die Maskierung von Symbolen geändert wird.

SCOPE = *CURRENT-SUB-LLM

Der Zeiger verweist auf das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

SCOPE = *EXPLICIT(...)**WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <text 1..255>**

Die Zeiger verweisen auf die explizit angegebenen Sub-LLMs. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *WHOLE-LLM werden alle Sub-LLMs angesprochen.

EXCEPT-SUB-LLM = *NONE / list-poss(10): <text 1..255>

Erlaubt es, aus der im Operanden WITHIN-SUB-LLM angegebenen Liste einzelne Zeiger auszuschließen. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben, die nicht berücksichtigt werden sollen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *NONE werden keine Zeiger ausgeschlossen.

SCOPE = *WHOLE-LLM

Alle Sub-LLMs des aktuellen LLM werden beim Ändern der Maskierung von Symbolen berücksichtigt.

VISIBLE =

Legt fest, ob die angegebenen Symbole sichtbar bleiben oder maskiert werden sollen.

VISIBLE = *YES

Die angegebenen Symbole bleiben für spätere BINDER- oder Bindelader-Läufe sichtbar.

VISIBLE = *NO(...)

Die angegebenen Symbole werden für spätere BINDER- oder Bindelader-Läufe maskiert.

KEEP-RESOLUTION =

Legt fest, ob befriedigte Externverweise zu dem angegebenen Symbol befriedigt bleiben oder aufgelöst werden.

KEEP-RESOLUTION = *YES

Befriedigte Externverweise bleiben befriedigt.

KEEP-RESOLUTION = *NO

Befriedigte Externverweise werden aufgelöst.

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Die Ausführung der Anweisung wird abgebrochen, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

REMOVE-MODULES

Entfernen von Modulen

Diese Anweisung entfernt Module im aktuellen LLM. Entfernt werden können Bindemodule und Sub-LLMs. Nicht entfernt werden:

- das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS),
- ein Sub-LLM, dessen Beginn zwar mit der Anweisung BEGIN-SUB-LLM-STATEMENTS definiert, dessen Ende jedoch mit der Anweisung END-SUB-LLM-STATEMENTS noch nicht festgelegt wurde.

Der Benutzer kann durch den Pfadnamen festlegen, in welchem Sub-LLM des aktuellen LLM die Module entfernt werden.

REMOVE-MODULES

NAME = *ALL / list-poss(40); <c-string 1..32 with-low> / <text 1..32>

PATH-NAME = *CURRENT-SUB-LLM / <c-string 1..255 with-low> / <text 1..255>

NAME =

Legt die Module des aktuellen LLM fest, die entfernt werden.

NAME = *ALL

Alle Module des aktuellen LLM werden entfernt.

NAME = <structured-name 1..32>

Vom Benutzer vergebene Namen der Module, die entfernt werden sollen.

NAME = <text 1..32>

Logischer Name, der in den Anweisungen INCLUDE-MODULES oder REPLACE-MODULES angegeben wurde.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

PATH-NAME =

Legt das Sub-LLM in der logischen Struktur des aktuellen LLM im Arbeitsbereich fest, aus dem Module entfernt werden.

PATH-NAME = *CURRENT-SUB-LLM

Das aktuelle Sub-LLM wird angenommen (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

PATH-NAME = <text 1..255>

Pfadname des Sub-LLM in der logischen Struktur des aktuellen LLM.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

RENAME-SYMBOLS

Ändern von Symbolnamen

Diese Anweisung ersetzt den Namen von Programmdefinitionen und Referenzen im aktuellen LLM durch einen neuen Namen.

Für jeden zu ändernden Symbolnamen muss eine eigene RENAME-SYMBOLS-Anweisung gegeben werden. Der Benutzer kann festlegen, welche Typen von Symbolen dieses Namens einbezogen werden.

Folgende Programmdefinitionen können umbenannt werden:

- Programmabschnitte (CSECTs),
- Einsprungstellen (ENTRYs),
- COMMON-Bereiche.

Folgende Referenzen können umbenannt werden:

- Externverweise (EXTRNs),
- V-Konstanten,
- bedingte Externverweise (WXTRNs).

RENAME-SYMBOLS

```

SYMBOL-NAME = <c-string 1..32 with-low> / <text 1..32>
, SYMBOL-TYPE = *ALL / *DEFINITIONS / *REFERENCES / list-poss(6): *CSECT / *ENTRY / *COMMON /
*EXTRN / *VCON / *WXTRN
, SYMBOL-OCCURRENCE = *PARAMETERS (...)
  *PARAMETERS(...)
    | FIRST-OCCURRENCE = 1 / <integer 1..32767>
    | , OCCURRENCE-NUMBER = 1 / <integer 1..32767> / *ALL
    | , WARNING-MESSAGE = *EXCEED / *LOWER / *BOTH
, SCOPE = *CURRENT-SUB-LLM / *EXPLICIT(...) / *WHOLE-LLM
  *EXPLICIT(...)
    | WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <c-string 1..255 with-low> /
    | <text 1..255>
    | , EXCEPT-SUB-LLM = *NONE / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
, NEW-NAME = <c-string 1..32 with-low> / <text 1..32>
, NAME-COLLISION = *STD / *IGNORED / *WARNING(...) / *ERROR(...)
  *WARNING(...)
    | SCOPE = *WHOLE-LLM / *SLICE
  *ERROR(...)
    | SCOPE = *WHOLE-LLM / *SLICE

```

SYMBOL-NAME = <text 1..32> / <c-string 1..32>

Legt den Symbolnamen fest, der geändert werden soll.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

SYMBOL-TYPE =

Legt den Typ der Symbole fest, die umbenannt werden.

SYMBOL-TYPE = *ALL

Alle Typen von Symbolen werden umbenannt.

SYMBOL-TYPE = *DEFINITIONS

Programmdefinitionen werden umbenannt.

SYMBOL-TYPE = *REFERENCES

Referenzen werden umbenannt.

SYMBOL-TYPE = *CSECT

Programmabschnitte werden umbenannt.

SYMBOL-TYPE = *ENTRY

Einsprungstellen werden umbenannt.

SYMBOL-TYPE = *COMMON

COMMON-Bereiche werden umbenannt.

SYMBOL-TYPE = *EXTRN

Externverweise werden umbenannt.

SYMBOL-TYPE = *VCON

V-Konstanten werden umbenannt.

SYMBOL-TYPE = *WXTRN

Bedingte Externverweise werden umbenannt.

SYMBOL-OCCURRENCE = *PARAMETERS(...)

Legt die Lage und die Anzahl des Auftretens der Symbole fest. In der logischen Struktur werden die Symbole in folgender Reihenfolge durchsucht:

1. CSECTs und ENTRYs
2. COMMONs
3. EXTRN, VCON und WXTRN.

FIRST-OCCURRENCE = 1 / <integer 1..32767>

Legt die Lage des ersten Auftretens des Symbols mit dem angegebenen Namen fest. Das Umbenennen beginnt beim x-ten Auftreten (x: <integer 1..32767>) des Symbols.

OCCURRENCE-NUMBER = 1 / <integer 1..32767> / ALL

Legt die Anzahl der auftretenden Symbole fest, die umbenannt werden sollen. Bei Angabe von ALL werden alle ab der festgelegten Position auftretenden Symbole umbenannt. Ansonsten werden die ersten y (y: <integer 1..32767>) auftretenden Symbole umbenannt.

WARNING-MESSAGE =

Legt fest, wann die Meldungen gesendet werden.

WARNING-MESSAGE = *EXCEED

Die Meldung BND2103 wird gesendet, wenn die Anzahl der im Operanden OCCURRENCE-NUMBER angegebenen Occurrences die tatsächliche, im angegebenen Bereich gefundene Anzahl an Occurrences überschreitet.

WARNING-MESSAGE = *LOWER

Die Meldung BND2106 wird gesendet, wenn die Anzahl der im Operanden OCCURRENCE-NUMBER angegebenen Occurrences unter der tatsächlichen, im angegebenen Bereich gefundenen Anzahl an Occurrences liegt.

WARNING-MESSAGE = *BOTH

Die Meldung BND2103 und die Meldung BND2106 werden gesendet.

SCOPE =

Setzt einen oder mehrere Zeiger. Sie verweisen auf die Sub-LLMs in der logischen Struktur des LLM, in denen der BINDER Symbole umbenennen soll.

SCOPE = *CURRENT-SUB-LLM

Der Zeiger verweist auf das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

SCOPE = *EXPLICIT(...)**WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <text 1..255>**

Die Zeiger verweisen auf die explizit angegebenen Sub-LLMs. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben. Bei Angabe *WHOLE-LLM werden alle Sub-LLMs angesprochen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

EXCEPT-SUB-LLM = *NONE / list-poss(10): <text 1..255>

Erlaubt es, aus der im Operanden WITHIN-SUB-LLM angegebenen Liste einzelne Sub-LLMs auszuschließen. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben, die nicht berücksichtigt werden sollen. Bei Angabe *NONE werden keine Zeiger ausgeschlossen.

Achtung: BINDER prüft Sonderdatentyp <path-name>. (siehe [Seite 202](#))

SCOPE = *WHOLE-LLM

Alle Sub-LLMs des aktuellen LLM werden beim Umbenennen von Symbolen berücksichtigt.

NEW-NAME = <text 1..32> / <c-string 1..32>

Neuer Symbolname, der den unter SYMBOL-NAME angegebenen Namen ersetzt.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Die Ausführung der Anweisung wird abgebrochen, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

REPLACE-MODULES

Ersetzen von Modulen

Diese Anweisung ersetzt einen oder mehrere Module im aktuellen LLM durch neue Module. Sowohl Bindemodule als auch Sub-LLMs können Module im aktuellen LLM ersetzen. Als Module können Bindemodule, LLMs oder beide ersetzt werden. LLMs mit Slices, die vom Benutzer definiert werden, und LLMs ohne Relativierungsinformation können nicht ersetzt werden. LLMs können dabei vollständig ersetzt werden oder es können Sub-LLMs ausgewählt werden.

Die Eingabequelle kann sein:

- für Bindemodule eine Programmbibliothek (Elementtyp R), eine Bindemodulbibliothek (OML) oder die EAM-Bindemoduldatei,
- für LLMs bzw. Sub-LLMs eine Programmbibliothek (Elementtyp L).

REPLACE-MODULES

```

NAME = <c-string 1..32 with-low> / <text 1..32>
, PATH-NAME = *CURRENT-SUB-LLM / <c-string 1..255 with-low> / <text 1..255>
, MODULE-CONTAINER = *LIBRARY-ELEMENT (...) / *FILE(...) / *OMF(...)
  *LIBRARY-ELEMENT(...)
    | LIBRARY = *CURRENT-INPUT-LIB / <filename 1..54 without-gen-vers> / *LINK(...) /
    | *BLSLIB-LINK / *OMF
    | *LINK(...)
    | | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
    | | ,ELEMENT = *ALL (...) / list-poss(40): <composed-name 1..64>(…) / <c-string 1..64>(…)
    | | *ALL(…)
    | | | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
    | | <composed-name>(…)
    | | | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
    | | | ,SUB-LLM = *WHOLE-LLM / <c-string 1..255 with-low> / <text 1..255>
    | | <c-string>(…)
    | | | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
    | | | ,SUB-LLM = *WHOLE-LLM / <c-string 1..255 with-low> / <text 1..255>
    | ,TYPE = (*L,*R) / list-poss(2): *L / *R

```

Fortsetzung ➔

<pre> *FILE(...) FILE-NAME = <filename 1..54 without-gen> / *LINK(...) *LINK(...) LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen> ,SUB-LLM = *WHOLE-LLM / <c-string 1..255 with-low> / <text 1..255> *OMF(...) ELEMENT = *ALL / list-poss(40): <composed-name 1..64> / <c-string 1..64> ,NEW-NAME = *INTERNAL / *ELEMENT-NAME / <c-string 1..32 with-low> / <text 1..32> ,SLICE = *CURRENT-SLICE / *ROOT / <structured-name 1..32> ,LOGICAL-STRUCTURE = INCLUSION-DEFAULT / *WHOLE-LLM / *OBJECT-MODULES ,TEST-SUPPORT = *INCLUSION-DEFAULT / *NO / *YES ,RUN-TIME-VISIBILITY = UNCHANGED / *NO / *YES ,RESOLUTION-SCOPE = *UNCHANGED / *STD / *PARAMETERS(...) *PARAMETERS(...) HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255> ,LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255> ,FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255> ,NAME-COLLISION = *STD / *IGNORED / *WARNING(...) / *ERROR(...) *WARNING(...) SCOPE = WHOLE-LLM / *SLICE *ERROR(...) SCOPE = WHOLE-LLM / *SLICE </pre>

NAME =

Gibt den Namen des Moduls an, das im aktuellen LLM ersetzt wird.

NAME = <structured-name 1..32>

Vom Benutzer vergebener Name des zu ersetzenden Moduls.

NAME = <text 1..32>

Logischer Name, den das Modul in der Anweisung INCLUDE-MODULES oder in einer früheren REPLACE-MODULES-Anweisung (Operand NEW-NAME) erhalten hat.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

PATH-NAME =

Legt das Sub-LLM in der logischen Struktur des aktuellen LLM fest, in dem Module ersetzt werden.

PATH-NAME = *CURRENT-SUB-LLM

Das aktuelle Sub-LLM wird angenommen
(siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

PATH-NAME = <text 1..255>

Pfadname des Sub-LLM in der logischen Struktur des aktuellen LLM.
Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

MODULE-CONTAINER =

Legt fest, wo das LLM abgespeichert ist.

MODULE-CONTAINER = *LIBRARY-ELEMENT(...)

Das LLM ist in einer Programmbibliothek abgespeichert.

LIBRARY =

Gibt die Eingabequelle an, aus der die Module eingelesen werden.

LIBRARY = *CURRENT-INPUT-LIB

Es wird die Eingabequelle benutzt, aus der mit einer Anweisung START-LLM-UPDATE, INCLUDE-MODULES oder REPLACE-MODULES das *letzte* Modul (OM oder LLM) gelesen wurde. Der Geltungsbereich des Operanden bezieht sich auf einen *Edit-Lauf*.

LIBRARY = <filename 1..54 without-gen-vers>

Dateiname der Bibliothek, die als Eingabequelle verwendet werden soll.

LIBRARY = *LINK(...)

Bezeichnet eine Bibliothek mit dem Dateikettungsnamen.

LINKNAME = <structured-name 1..8>

Dateikettungsname einer Bibliothek, die als Eingabequelle verwendet werden soll.

LIBRARY = *BLSLIB-LINK

Die Eingabequellen sind Bibliotheken mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99). Die Bibliotheken werden in der Reihenfolge nach aufsteigenden Werten „nn“ des Dateikettungsnamens durchsucht.

LIBRARY = *OMF

Die Eingabequelle ist die EAM-Bindemoduldatei. Sie enthält nur Bindemodule.

ELEMENT =

Legt den Elementnamen und die Elementversion der Module fest, die aus der angegebenen Eingabequelle eingelesen werden.

ELEMENT = *ALL(...)

Alle Module werden aus der angegebenen Eingabequelle eingelesen.

VERSION =

Gibt die Elementversion des Moduls an. Die Elementversion ist nur für Programm-bibliotheken gültig.

VERSION = *HIGHEST-EXISTING

Der BINDER übernimmt als Elementversion den Standardwert für die höchste Version bei Programmbibliotheken (siehe Handbuch „LMS“ [3]).

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

ELEMENT = <composed-name 1..64>(...)

Explizite Angabe des Elementnamens und der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-name> und <element-version> (siehe [Seite 202](#)).

VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>

Gibt die Elementversion des Moduls an. Die Elementversion ist nur für Programm-bibliotheken gültig.

Bedeutung der Operanden siehe oben.

SUB-LLM =

Legt fest, ob das vollständige LLM oder ein Sub-LLM als Element übernommen wird.

SUB-LLM = *WHOLE-LLM

Das vollständige LLM wird übernommen.

SUB-LLM = <text 1..255>

Pfadname des Sub-LLM, das übernommen wird.

Achtung: Der BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

ELEMENT = <c-string 1..64>(...)

Explizite Angabe des Elementnamens und der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-name> und <element-version> (siehe [Seite 202](#)). Bedeutung der Operanden siehe oben.

TYPE =

Legt die Priorität der Module (Bindemodule und/oder LLMs) fest, die einen Modul im aktuellen LLM ersetzen können.

TYPE = (*L,*R)

Sowohl LLMs als auch Bindemodule können Module im aktuellen LLM ersetzen. Ist für ein LLM der gleiche Name wie für ein Bindemodul angegeben, wird das LLM übernommen.

TYPE = (*R,*L)

Sowohl LLMs als auch Bindemodule können Module ersetzen. Ist für ein LLM der gleiche Name wie für ein Bindemodul angegeben, wird das Bindemodul übernommen.

TYPE = *R

Nur Bindemodule können Module ersetzen.

TYPE = *L

Nur LLMs können Module ersetzen.

MODULE-CONTAINER = *FILE(...)

Das LLM ist in einer PAM-Datei abgespeichert.

FILE-NAME =

Angabe der PAM-Datei, die das LLM enthält.

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

Name der PAM-Datei, die das LLM enthält.

FILE-NAME = *LINK(...)**LINK-NAME = <structured-name 1..8>**

Dateikettungsname der PAM-Datei, die das LLM enthält.

MODULE-CONTAINER = *OMF(...)

Die Eingabequelle ist die EAM-Bindemoduldatei. Sie enthält nur Bindemodule.

ELEMENT = *ALL / list-poss(40): <composed-name 1..64> / <c-string 1..64>

Bedeutung der Operanden siehe oben.

NEW-NAME =

Gibt an, welchen neuen logischen Namen das ersetzte Modul haben soll.

NEW-NAME = *INTERNAL

Der interne Name wird als logischer Name übernommen.

NEW-NAME = *ELEMENT-NAME

Als logischer Name des Moduls wird der Name des Bibliothekselementes übernommen.

NEW-NAME = <structured-name 1..32>

Explizite Angabe des logischen Namens.

SLICE =

Bestimmt die Slice in der physischen Struktur des LLM, in der die Module ersetzt werden. Die Slice muss mit einer Anweisung SET-USER-SLICE-POSITION definiert sein.

SLICE = *CURRENT-SLICE

Module werden in der aktuellen Slice ersetzt. Dies ist die Slice, die durch die letzte vorhergehende Anweisung SET-USER-SLICE-POSITION definiert wurde.

SLICE = *ROOT

Module werden in der Root-Slice (%ROOT) ersetzt.

SLICE = <structured-name 1..32>

Explizite Angabe der Slice, in der Module ersetzt werden.

LOGICAL-STRUCTURE =

Legt fest, ob die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird.

LOGICAL-STRUCTURE = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben *Edit-Laufs* angenommen.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird in das aktuelle LLM übernommen.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Die Strukturinformation wird nicht übernommen. Im aktuellen LLM wird nur eine Struktur mit Bindemodulen (OMs) aufgebaut.

TEST-SUPPORT =

Legt fest, ob die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.



Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben *Edit-Laufs* angenommen.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

RUN-TIME-VISIBILITY =

Legt fest, ob das Modul als Laufzeitmodul betrachtet werden soll. Alle Symbole eines Laufzeitmoduls werden beim Abspeichern maskiert und zunächst nicht mehr zur Befriedigung von Externverweisen verwendet. Bei einem späteren Lesezugriff auf das Modul (z.B. bei START-LLM-UPDATE oder INCLUDE-MODULES) wird die Maskierung wieder rückgängig gemacht.

RUN-TIME-VISIBILITY = *UNCHANGED

Der Wert wird nicht geändert. Wird ein Modul erstmals mit INCLUDE-MODULES oder REPLACE-MODULES in ein LLM eingefügt, so übernimmt der BINDER den Wert NO.

RUN-TIME-VISIBILITY = *NO

Das Modul soll nicht als Laufzeitmodul betrachtet werden.

RUN-TIME-VISIBILITY = *YES

Das Modul soll als Laufzeitmodul betrachtet werden. Alle Symbole des Moduls werden beim Abspeichern maskiert.

RESOLUTION-SCOPE =

Legt Prioritätsklassen fest, die steuern, in welcher Reihenfolge der BINDER andere Module bei der Befriedigung von Externverweisen durchsuchen soll. Für jede Klasse müssen zwei Werte unterschieden werden:

- der dynamische Wert.
Er beeinflusst die Reihenfolge, in der Module zur Befriedigung von Externverweisen durchsucht werden. Er wird jedoch nicht im LLM abgespeichert.
- der statische Wert.
Er wird im LLM abgespeichert und bildet die Basis zur Ermittlung des dynamischen Wertes.

RESOLUTION-SCOPE = *UNCHANGED

Die statischen Werte der Prioritätsklassen sind die in den betroffenen Modulen gespeicherten Werte. Für Objektmodule (OM) erhalten sie den Wert *STD (siehe unten).

RESOLUTION-SCOPE = *STD

Die statischen Werte der Prioritätsklassen sind *STD. Das bedeutet, dass die dynamischen Werte vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen werden. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Die Voreinstellung für die statischen Werte des ROOT-Knotens eines LLMs ist *STD. Die dynamischen Werte sind in diesem Fall *NONE (siehe unten).

RESOLUTION-SCOPE = *PARAMETERS(...)

Die statischen Werte der einzelnen Prioritätsklassen werden separat vereinbart.

**HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /
<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen vor allen anderen durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)).

HIGH-PRIORITY-SCOPE = *STD

Der statische Wert dieser Prioritätsklasse ist *STD. Das bedeutet, dass der dynamische Wert vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen wird. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Der statische Wert des ROOT-Knotens eines LLMs ist standardmäßig *STD. Der dynamische Wert ist in diesem Fall *NONE (siehe unten).

HIGH-PRIORITY-SCOPE = *NONE

Die Prioritätsklasse HIGH-PRIORITY-SCOPE ist nicht definiert. D.h. es gibt keine Module, die vor allen anderen zum Befriedigen von Externverweisen durchsucht werden sollen. Der Wert des übergeordneten Knotens wird nicht übernommen.

HIGH-PRIORITY-SCOPE = <c-string 1..255 with-low> / <text 1..255>

Pfadname des Sub-LLMs, das zum Befriedigen von Externverweisen zuerst durchsucht werden soll.

**LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /
<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen erst dann durchsucht werden soll, wenn die Suche in allen anderen Modulen erfolglos war (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

**FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE /
<c-string 1..255 with-low> / <text 1..255>**

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen nicht durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

Ein Beispiel zum Vererbungsmechanismus für die PRIORITY-SCOPE-Werte finden Sie bei der Beschreibung der Anweisung RESOLVE-BY-AUTOLINK auf [Seite 285](#).

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Die Ausführung der Anweisung wird abgebrochen, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

RESOLVE-BY-AUTOLINK

Befriedigen von Externverweisen durch Autolink

Diese Anweisung legt fest, wie der BINDER unbefriedigte Externverweise befriedigen soll (Autolink-Funktion).

Der BINDER durchsucht die angegebenen Bibliotheken nach Modulen mit passenden CSECTs und ENTRYs und fügt die gefundenen Module in das aktuelle LLM ein.

LLMs mit Slices, die vom Benutzer definiert werden, und LLMs ohne Relativierungsinformationen werden nicht eingefügt. Findet der BINDER solche LLMs, bricht er die Autolink-Funktion ab.

RESOLVE-BY-AUTOLINK

```

LIBRARY = *CURRENT-INPUT-LIB / *BLSLIB-LINK /
           list-poss(40): <filename 1..54 without-gen-vers> / *LINK(...)

*LINK(...)
  | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
, TYPE = (*L, *R) / list-poss(2): *L / *R
, SYMBOL-NAME = *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
, SCOPE = *CURRENT-SUB-LLM / *EXPLICIT(...) / *WHOLE-LLM
*EXPLICIT(...)
  | WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
  | EXCEPT-SUB-LLM = *NONE / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
, NAME = *INTERNAL / *ELEMENT-NAME
, PATH-NAME = *CURRENT-SUB-LLM / <c-string 1..255 with-low> / <text 1..255>
, LOGICAL-STRUCTURE = *INCLUSION-DEFAULT / *WHOLE-LLM / *OBJECT-MODULES
, TEST-SUPPORT = *INCLUSION-DEFAULT / *NO / *YES
, RUN-TIME-VISIBILITY = *UNCHANGED / *NO / *YES
, RESOLUTION-SCOPE = *UNCHANGED / *STD / *PARAMETERS(...)
*PARAMETERS(...)
  | HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255>
  | LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255>
  | FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE / <text 1..255>

```

Fortsetzung ➡

```
,NAME-COLLISION = STD / *IGNORED / *WARNING(...) / *ERROR(...)
  *WARNING(...)
    | SCOPE = *WHOLE-LLM / *SLICE
  *ERROR(...)
    | SCOPE = *WHOLE-LLM / *SLICE
```

LIBRARY =

Gibt eine oder mehrere Eingabebibliotheken an, die der BINDER zur Befriedigung von Externverweisen durchsuchen soll.

LIBRARY = *CURRENT-INPUT-LIB

Es wird die Eingabebibliothek durchsucht, aus der mit einer Anweisung START-LLM-UPDATE, INCLUDE-MODULES oder REPLACE-MODULES das *letzte* Modul (OM oder LLM) gelesen wurde.

Der Geltungsbereich des Operanden bezieht sich auf einen Edit-Lauf.

LIBRARY = *BLSLIB-LINK

Die Eingabebibliotheken mit den Dateikettungsnamen BLSLIBnn (00≤nn≤99) werden durchsucht.

LIBRARY = <filename 1..54 without-gen-vers>

Dateinamen der Eingabebibliotheken, die durchsucht werden.

LIBRARY = *LINK(...)**LINK-NAME = <structured-name 1..8>**

Dateikettungsname der Eingabebibliothek, die durchsucht wird.

TYPE =

Legt die Priorität der Module (Bindemodule und/oder LLMs) fest, die nach passenden CSECTs und Einsprungstellen durchsucht werden.

TYPE = (*L,*R)

Sowohl LLMs als auch Bindemodule werden durchsucht. Ist für ein LLM der gleiche Name wie für ein Bindemodul vorhanden, wird das LLM durchsucht.

TYPE = (*R,*L)

Sowohl LLMs als auch Bindemodule werden durchsucht. Ist für ein LLM der gleiche Name wie für ein Bindemodul vorhanden, wird das Bindemodul durchsucht.

TYPE = *R

Nur Bindemodule werden durchsucht.

TYPE = *L

Nur LLMs werden durchsucht.

SYMBOL-NAME =

Legt die Externverweise fest, die der BINDER befriedigen soll.

SYMBOL-NAME = *ALL

Alle Externverweise sollen befriedigt werden.

SYMBOL-NAME = <c-string 1..255> / <text 1..32>

Namen der Externverweise, die befriedigt werden sollen. Die Angabe von Platzhaltern (Wildcards) ist zulässig.

Achtung: BINDER prüft Sonderdatentyp <symbol> und <symbol-with-wild> (siehe [Seite 202](#)).

SCOPE =

Setzt einen oder mehrere Zeiger. Sie verweisen auf die Sub-LLMs in der logischen Struktur des LLM, in denen der BINDER unbefriedigte Externverweise befriedigen soll.

SCOPE = *CURRENT-SUB-LLM

Der Zeiger verweist auf das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

SCOPE = *EXPLICIT(...)**WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <text 1..255>**

Die Zeiger verweisen auf die explizit angegebenen Sub-LLMs. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe von *WHOLE-LLM werden alle Sub-LLMs angesprochen.

EXCEPT-SUB-LLM = *NONE / list-poss(10): <text 1..255>

Erlaubt es, aus der im Operanden WITHIN-SUB-LLM angegebenen Liste einzelne Zeiger auszuschließen. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben, die nicht berücksichtigt werden sollen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe von *NONE werden keine Zeiger ausgeschlossen.

SCOPE = *WHOLE-LLM

Alle Sub-LLMs des aktuellen LLM werden beim Befriedigen von Externverweisen berücksichtigt.

NAME =

Gibt an, welcher Name als logischer Name übernommen wird, wenn das Modul durch die Autolink-Funktion eingefügt wird.

NAME = *INTERNAL

Der interne Name wird als logischer Name übernommen.

NAME = *ELEMENT-NAME

Der Name des Moduls als Bibliothekselement wird als logischer Name übernommen. Wenn nötig, kürzt der BINDER diesen Namen auf 32 Zeichen.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

PATH-NAME =

Legt das Sub-LLM in der logischen Struktur des aktuellen LLM im Arbeitsbereich fest, in das Module eingefügt werden.

PATH-NAME = *CURRENT-SUB-LLM

Der *aktuelle Sub-LLM* wird angenommen

(siehe Anweisung BEGIN-SUB-LLM-STATEMENTS auf [Seite 211](#)).

PATH-NAME = <text 1..255>

Pfadname des Sub-LLM in der logischen Struktur des aktuellen LLM.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

LOGICAL-STRUCTURE =

Legt fest, ob die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird.

LOGICAL-STRUCTURE = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben Edit-Laufs angenommen.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird in das aktuelle LLM übernommen.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Die Strukturinformation wird nicht übernommen. Im aktuellen LLM wird nur eine Struktur mit Bindemodulen (OMs) aufgebaut.

TEST-SUPPORT =

Legt fest, ob die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.



Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *INCLUSION-DEFAULT

Es werden die Werte des Operanden INCLUSION-DEFAULTS aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE oder MODIFY-LLM-ATTRIBUTES desselben Edit-Laufs angenommen.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

RUN-TIME-VISIBILITY =

Legt fest, ob Symbole beim Abspeichern maskiert und zunächst nicht mehr zur Befriedigung von Externverweisen verwendet werden. Bei einem späteren Lesezugriff auf das Modul (z.B. bei START-LLM-UPDATE oder INCLUDE-MODULES) wird die Maskierung wieder rückgängig gemacht.

RUN-TIME-VISIBILITY = *UNCHANGED

Der Wert wird nicht geändert. Wird ein Modul erstmals mit INCLUDE-MODULES oder REPLACE-MODULES in ein LLM eingefügt, so übernimmt der BINDER den Wert NO.

RUN-TIME-VISIBILITY = *NO

Symbole die per Autolink eingebunden wurden, bleiben sichtbar.

RUN-TIME-VISIBILITY = *YES

Symbole die per Autolink eingebunden wurden, werden beim Abspeichern maskiert.

RESOLUTION-SCOPE =

Legt Prioritätsklassen fest, die steuern, in welcher Reihenfolge der BINDER andere Module bei der Befriedigung von Externverweisen durchsuchen soll. Für jede Klasse müssen zwei Werte unterschieden werden:

- der dynamische Wert.
Er beeinflusst die Reihenfolge, in der Module zur Befriedigung von Externverweisen durchsucht werden. Er wird jedoch nicht im LLM abgespeichert.
- der statische Wert.
Er wird im LLM abgespeichert und bildet die Basis zur Ermittlung des dynamischen Wertes.

RESOLUTION-SCOPE = *UNCHANGED

Die statischen Werte der Prioritätsklassen sind die in den betroffenen Modulen gespeicherten Werte. Für Objektmodule (OM) erhalten sie den Wert *STD (siehe unten).

RESOLUTION-SCOPE = *STD

Die statischen Werte der Prioritätsklassen sind *STD. Das bedeutet, dass die dynamischen Werte vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen werden. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Die Voreinstellung für die statischen Werte des ROOT-Knotens eines LLMs ist *STD. Die dynamischen Werte sind in diesem Fall *NONE (siehe unten).

RESOLUTION-SCOPE = *PARAMETERS(...)

Die statischen Werte der einzelnen Prioritätsklassen werden separat vereinbart.

HIGH-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen vor allen anderen durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)).

HIGH-PRIORITY-SCOPE = *STD

Der statische Wert dieser Prioritätsklasse ist *STD. Das bedeutet, dass der dynamische Wert vom übergeordneten Knoten in der logischen Struktur des LLMs übernommen wird. Dieser Vererbungsmechanismus wird jedes Mal angewandt, wenn eine neue Suche zur Auflösung von Externverweisen gestartet wird. Der statische Wert des ROOT-Knotens eines LLMs ist standardmäßig *STD. Der dynamische Wert ist in diesem Fall *NONE (siehe unten).

HIGH-PRIORITY-SCOPE = *NONE

Die Prioritätsklasse HIGH-PRIORITY-SCOPE ist nicht definiert. D.h. es gibt keine Module, die vor allen anderen zum Befriedigen von Externverweisen durchsucht werden sollen. Der Wert des übergeordneten Knotens wird nicht übernommen.

HIGH-PRIORITY-SCOPE = <c-string 1..255 with-low> / <text 1..255>

Pfadname des Sub-LLMs, das zum Befriedigen von Externverweisen zuerst durchsucht werden soll.

LOW-PRIORITY-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen erst dann durchsucht werden soll, wenn die Suche in allen anderen Modulen erfolglos war (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

FORBIDDEN-SCOPE = *UNCHANGED / *STD / *NONE /

<c-string 1..255 with-low> / <text 1..255>

Gibt an, welches Sub-LLM zum Befriedigen von Externverweisen nicht durchsucht werden soll (siehe [Abschnitt „Regeln zur Befriedigung von Externverweisen“ auf Seite 81](#)). Die Bedeutung der einzelnen Operandenwerte ist analog zur Prioritätsklasse HIGH-PRIORITY-SCOPE.

Folgendes Beispiel soll den Vererbungsmechanismus für die PRIORITY-SCOPE-Werte verdeutlichen:

Eine Bibliothek enthält die beiden LLMs LLM1 und LLM2. Für diese sind statische HIGH-PRIORITY-SCOPE-Werte festgelegt, die in der folgenden Abbildung aufgelistet sind.

Modul	statischer Wert des HIGH-PRIORITY-SCOPE
LLM1	LLM1.OM12
└─ OM11	*STD
└─ OM12	*NONE
LLM2	*STD
└─ OM21	LLM2.OM22
└─ OM22	*STD

LLM2 wird in LLM1 eingebunden. BINDER ermittelt beim Binden aus den statischen HIGH-PRIORITY-SCOPE-Werten die dynamischen, die die Suchreihenfolge beim Befriedigen der Externverweise beeinflussen.

Folgende Abbildung zeigt die sich dadurch ergebende Modul-Struktur und die dynamischen Werte des HIGH-PRIORITY-SCOPE.

Modul	dynamischer Wert des HIGH-PRIORITY-SCOPE	Bemerkung
LLM1	LLM1.OM12	= statischer Wert
└─ OM11	*STD	geerbt von LLM1
└─ OM12	*NONE	= statischer Wert
└─ LLM2	*STD	geerbt von LLM1
└─ OM21	LLM2.OM22	= statischer Wert
└─ OM22	*STD	geerbt von LLM1

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der Anweisung auftreten.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Legt fest, dass die Anweisung abgebrochen wird, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht. Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

SAVE-LLM

Speichern eines LLM

Diese Anweisung speichert das aktuelle LLM, das mit einer Anweisung START-LLM-CREATION erzeugt oder START-LLM-UPDATE geändert wurde, als Element vom Typ L in einer Programmbibliothek.

Ein mit der Anweisung START-LLM-UPDATE geändertes LLM wird dabei in das ursprüngliche Element geschrieben, falls das neue Element den gleichen Elementnamen und die gleiche Elementversion in der Programmbibliothek behält und OVERWRITE=*YES angegeben wurde.

Die Anweisung SAVE-LLM beendet den BINDER-Lauf nicht. Deshalb kann der Benutzer im selben BINDER-Lauf das aktuelle LLM weiterverarbeiten, mit einer weiteren Anweisung START-LLM-CREATION ein neues LLM erzeugen oder mit einer weiteren Anweisung START-LLM-UPDATE ein anderes LLM ändern.

SAVE-LLM

MODULE-CONTAINER = *CURRENT / *LIBRARY-ELEMENT(...) / *FILE(...)

*LIBRARY-ELEMENT(...)

LIBRARY = *CURRENT / <filename 1..54 without-gen-vers> / *LINK(...)

*LINK(...)

| **LINK-NAME** = <structured-name 1..8> / <filename 1..8 without-gen>

,**ELEMENT** = *CURRENT-NAME (...) / *INTERNAL-NAME(...) / <composed-name 1..64>(...) /
<c-string 1..64>(...)

*CURRENT-NAME(...)

| **VERSION** = *CURRENT-VERSION / *INTERNAL-VERSION / *UPPER-LIMIT /
*INCREMENT / <composed-name 1..24> / <c-string 1..24>

*INTERNAL-NAME(...)

| **VERSION** = *INTERNAL-VERSION / *UPPER-LIMIT / *INCREMENT /
<composed-name 1..24> / <c-string 1..24>

<composed-name 1..64>(...)

| **VERSION** = *INTERNAL-VERSION / *UPPER-LIMIT / *INCREMENT /
<composed-name 1..24> / <c-string 1..24>

<c-string 1..64>(...)

| **VERSION** = *INTERNAL-VERSION / *UPPER-LIMIT / *INCREMENT /
<composed-name 1..24> / <c-string 1..24>

Fortsetzung ➡

```

*FILE(...)
  |
  | FILE-NAME = *CURRENT / <filename 1..54 without-gen> / *LINK(...)
  |
  | *LINK(...)
  | |
  | | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
  |
, OVERWRITE = *LAST-SAVE / *STD / *YES / *NO
, FOR-BS2000-VERSIONS = *LAST-SAVE / *STD / *FROM-CURRENT(...) / *FROM-V10(...) /
  *FROM-OSD-V1(...) / *FROM-OSD-V3(...) / *FROM-OSD-V4(...)

*FROM-CURRENT(...)
  |
  | CONNECTION-MODE = *OSD-DEFAULT / *BY-RELOCATION / *BY-RESOLUTION

*FROM-V10(...)
  |
  | CONNECTION-MODE = *BY-RESOLUTION / *BY-RELOCATION

*FROM-OSD-V1(...)
  |
  | CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

*FROM-OSD-V3(...)
  |
  | CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

*FROM-OSD-V4(...)
  |
  | CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

, REQUIRED-COMPRESSION = *LAST-SAVE / *STD / *NO / *YES
, NAME-COLLISION = *LAST-SAVE / *STD / *IGNORED / *WARNING(...) / *ERROR(...)

*WARNING(...)
  |
  | SCOPE = *WHOLE-LLM / *SLICE

*ERROR(...)
  |
  | SCOPE = WHOLE-LLM / *SLICE

, SYMBOL-DICTIONARY = *LAST-SAVE / *YES / *NO
, RELOCATION-DATA = *LAST-SAVE / *YES / *NO / *UNRESOLVED-ONLY
, LOGICAL-STRUCTURE = *LAST-SAVE / *WHOLE-LLM / *OBJECT-MODULES / *NONE
, TEST-SUPPORT = *LAST-SAVE / *YES / *NO

```

Fortsetzung ➔


```

,LOAD-ADDRESS = *LAST-SAVE / *UNDEFINED / *NULL / *BY-SLICES(...) / <x-string 1..8>
  *BY-SLICES(...)
    ADDRESSES = list-poss(40): *REGION(...) / *SLICE(...)
      *REGION(...)
        REGION-NAME = <structured-name 1..32>
        ,REGION-ADDRESS = <x-string 1..8>
      *SLICE(...)
        SLICE-NAME = *ROOT / <structured-name 1..32>
        ,SLICE-ADDRESS = <x-string 1..8> / *BEHIND-SLICE(...)
          *BEHIND-SLICE(...)
            SLICE = <structured-name 1..32>
,ENTRY-POINT = *LAST-SAVE / *STD / *BY-MODULE(...) / <c-string 1..32 with-low> / <text 1..32>
  *BY-MODULE(...)
    PATH-NAME = <text 1..255>
,MAP = *LAST-SAVE / *YES / *NO

```

MODULE-CONTAINER =

Legt fest, wo das LLM abgespeichert wird.

MODULE-CONTAINER = *CURRENT

Das LLM wird in der Bibliothek oder PAM-Datei abgespeichert, die in der letzten vorhergehenden Anweisung SAVE-LLM oder START-LLM-UPDATE angegeben wurde.

MODULE-CONTAINER = *LIBRARY-ELEMENT(...)

Das LLM wird in einer Programmbibliothek abgespeichert.

LIBRARY =

Legt die Programmbibliothek fest, in der das LLM abgespeichert wird.

LIBRARY = *CURRENT

Der BINDER übernimmt die Programmbibliothek aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, wählt der BINDER die Programmbibliothek wie folgt aus:

- er übernimmt die Bibliothek aus der zugehörigen Anweisung START-LLM-UPDATE, wenn das LLM mit einer Anweisung START-LLM-UPDATE geändert wurde,
- er gibt eine Fehlermeldung aus und fordert die explizite Angabe der Bibliothek, wenn das LLM mit einer Anweisung START-LLM-CREATION erzeugt wurde.

LIBRARY = <filename 1..54 without-gen-vers>

Dateiname der Programmbibliothek, in der das LLM abgespeichert wird.

LIBRARY = *LINK(...)**LINK-NAME = <structured-name 1..8>**

Dateikettungsname der Programmbibliothek, in der das LLM abgespeichert werden soll.

ELEMENT =

Elementname und Elementversion, die das LLM beim Speichern in der Programmbibliothek erhalten soll.

Hinweis

Das Abspeichern eines LLM unter demselben Elementnamen und derselben Elementversion ist nicht möglich, wenn andere Tasks das LLM gleichzeitig bearbeiten oder einfügen. Das Bibliothekselement ist dann für Schreibzugriffe gesperrt. Zur Auflösung dieses Konfliktes müssen bis auf eine alle konkurrierenden Tasks das LLM unter einem anderen Namen bzw. mit einer anderen Version zwischenspeichern.

ELEMENT = *CURRENT-NAME(...)

Der BINDER übernimmt den Elementnamen aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, übernimmt der BINDER als Elementname des LLM:

- den Elementnamen aus der zugehörigen Anweisung START-LLM-UPDATE, wenn das LLM mit einer Anweisung START-LLM-UPDATE geändert wurde,
- den internen Namen aus der zugehörigen Anweisung START-LLM-CREATION, wenn das LLM mit einer Anweisung START-LLM-CREATION erzeugt wurde.

VERSION =

Legt die Elementversion fest.

VERSION = *CURRENT-VERSION

Der BINDER übernimmt die Elementversion aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, übernimmt der BINDER als Version des LLM:

- die angegebene Elementversion aus der zugehörigen Anweisung START-LLM-UPDATE, wenn das LLM mit einer Anweisung START-LLM-UPDATE geändert wurde,
- die interne Version aus der zugehörigen Anweisung START-LLM-CREATION, wenn das LLM mit einer Anweisung START-LLM-CREATION erzeugt wurde.

VERSION = *INTERNAL-VERSION

Der BINDER übernimmt als Elementversion die interne Version des LLM.

VERSION = *UPPER-LIMIT

Der BINDER übernimmt als Elementversion den Standardwert für die höchste Version bei Programmbibliotheken (siehe Handbuch „LMS“ [3]).

VERSION = *INCREMENT

Die aktuelle Elementversion wird um 1 erhöht.

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

ELEMENT = *INTERNAL-NAME(...)

Der BINDER übernimmt als Elementnamen den internen Namen des LLM.

VERSION =

Legt die Elementversion fest.

VERSION = *INTERNAL-VERSION

Der BINDER übernimmt als Elementversion die interne Version des LLM.

VERSION = *UPPER-LIMIT

Der BINDER übernimmt als Elementversion den Standardwert für die Version bei Programmbibliotheken (siehe Handbuch „LMS“ [3]).

VERSION = *INCREMENT

Die aktuelle Elementversion wird um 1 erhöht.

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

ELEMENT = <composed-name 1..64>(...)

Explizite Angabe des Elementnamens.

Achtung: BINDER prüft Sonderdatentyp <element-name> (siehe [Seite 202](#)).

In den DBL-Kommandos LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM, siehe Handbuch „BLSSERV Bindelader-Starter“ [1]) ist bei Einsatz von BLSSERV bis V2.4 die Länge der Elementnamen auf 32 Zeichen begrenzt.

BINDER gibt deshalb folgende Warnmeldung aus, wenn der Elementname länger als 32 Zeichen ist:

```
BND2110 WARNUNG: ELEMENT-NAME LAENGER ALS 32 ZEICHEN NICHT MIT 'DBL'
          VERARBEITBAR
```

VERSION = *INTERNAL-VERSION / *UPPER-LIMIT / *INCREMENT / <composed-name 1..24> / <c-string 1..24>

Legt die Elementversion fest.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

Bedeutung der Operanden siehe oben.

ELEMENT = <c-string 1..64>(…)

Explizite Angabe des Elementnamens

Achtung: BINDER prüft Sonderdatentyp <element-name> (siehe [Seite 202](#)).

In den DBL-Kommandos LOAD-/START-EXECUTABLE-PROGRAM (bzw. LOAD-/START-PROGRAM, siehe Handbuch „BLSSERV Bindelader-Starter“ [1]) ist bei Einsatz von BLSSERV bis V2.4 die Länge der Elementnamen auf 32 Zeichen begrenzt.

BINDER gibt deshalb folgende Warnmeldung aus, wenn der Elementname länger als 32 Zeichen ist:

```
BND2110 WARNUNG: ELEMENT-NAME LAENGER ALS 32 ZEICHEN NICHT MIT 'DBL'
          VERARBEITBAR
```

VERSION = *INTERNAL-VERSION / *UPPER-LIMIT / *INCREMENT / <composed-name 1..24> / <c-string 1..24>

Legt die Elementversion fest.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

Bedeutung der Operanden siehe oben.

MODULE-CONTAINER = *FILE(…)

Das LLM wird in einer PAM-Datei abgespeichert. Ein solches LLM wird als PAM-LLM bezeichnet.

Hinweise

- PAM-LLMs können ab BLSSERV V2.5 geladen werden.
- Wenn ein LLM, das in einer PAM-Datei abgespeichert ist, mit START-LLM-UPDATE geändert und mit SAVE-LLM in eine PLAM-Bibliothek abgespeichert wird, dann ist der Wert für *CURRENT-NAME der interne Name des LLMs.
- Bei der Ausgabe eines LLMs in eine PAM-Datei legt BINDER zunächst eine temporäre PAM-Datei an, in der das LLM zwischengespeichert wird. Erst nach erfolgreicher Generierung des LLMs wird es in die mit SAVE-LLM festgelegte PAM-Datei geschrieben, und die temporäre Datei wird gelöscht. Zur Erzeugung der temporären Datei wird die BS2000-Funktion „Temporäre Dateien“ genutzt, sofern diese von der Systembetreuung mit dem Klasse2-Systemparameter TEMPFILE aktiviert ist. Andernfalls erzeugt BINDER eine Datei, deren Name sich aus dem Namen der Eingabedatei sowie dem Datum und der Uhrzeit zusammensetzt.

FILE-NAME =

Angabe der Datei, in die das LLM gespeichert werden soll.

FILE-NAME = *CURRENT

Das LLM wird in der PAM-Datei abgespeichert, die in der letzten vorhergehenden Anweisung SAVE-LLM oder START-LLM-UPDATE angegeben wurde.

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

Name der PAM-Datei, in der das LLM abgespeichert wird.

FILE-NAME = *LINK(...)**LINK-NAME = <structured-name 1..8>**

Dateikettungsname der PAM-Datei, in der das LLM abgespeichert wird.

OVERWRITE =

Gibt an, ob Überschreiben erlaubt ist oder nicht.

OVERWRITE = *LAST-SAVE

Der BINDER übernimmt den Wert aus der letzten SAVE-LLM-Anweisung desselben Edit-Laufs. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt BINDER den Wert *STD.

OVERWRITE = *STD

Der BINDER übernimmt den Wert des Operanden aus der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert *YES.

OVERWRITE = *YES

Überschreiben ist erlaubt.

OVERWRITE = *NO

Überschreiben ist nicht erlaubt.

FOR-BS2000-VERSIONS =

Gibt an, in welcher BS2000 OSD/BC-Version das erzeugte LLM durch den DBL geladen werden soll. Das LLM kann von DBL nur in der angegebenen oder einer höheren Version von BS2000 OSD/BC verarbeitet werden.

FOR-BS2000-VERSIONS = *BY-PROGRAM

BINDER legt das Format des erzeugten LLMs anhand von dessen Inhalt fest. Das LLM-Format ist immer das niedrigstmögliche, das die benötigte Funktionalität gewährleistet. Beispielsweise bekommt ein LLM, der RISC-Code enthält, das Format 3, während ein LLM, das komprimierten Text enthält, im Format 2 erzeugt wird.

FOR-BS2000-VERSIONS = *LAST-SAVE

(nur in der Anweisung SAVE-LLM, wie bisher)

FOR-BS2000-VERSIONS = *STD

Der BINDER übernimmt den Wert des Operanden aus der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert *FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-CURRENT(...)

Die Version von BS2000 OSD/BC, unter der BINDER gerade abläuft, wird übernommen.

CONNECTION-MODE =

Gibt die Art der Verknüpfung zwischen dem PRIVATE- und dem PUBLIC-Teil an. Dieser Operand ist nur von Bedeutung, wenn das LLM gemäß der PUBLIC-Attribute der darin enthaltenen CSECTS in Slices aufgeteilt wurde.

CONNECTION-MODE = *OSD-DEFAULT

Diese Angabe wird aus Kompatibilitätsgründen unterstützt. Sie ist gleichbedeutend mit CONNECTION-MODE = *BY-RELOCATION.

CONNECTION-MODE = *BY-RELOCATION

Die Verknüpfung zwischen PRIVATE- und PUBLIC-Teil erfolgt durch Relativieren.

CONNECTION-MODE = *BY-RESOLUTION

Die Verknüpfung zwischen PRIVATE- und PUBLIC-Teil erfolgt durch Auflösen.

FOR-BS2000-VERSIONS = *FROM-V10(...)

Das LLM kann in allen Versionen von BS2000 OSD/BC geladen werden.

CONNECTION-MODE = *BY-RESOLUTION / *BY-RELOCATION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-OSD-V1(...)

Das LLM kann in allen Versionen von BS2000 OSD/BC geladen werden.

CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-OSD-V3(...)

Das LLM kann ab BS2000/OSD-BC V3.0 geladen werden.

CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

FOR-BS2000-VERSIONS = *FROM-OSD-V4(...)

Das LLM kann ab BS2000/OSD-BC V4.0 geladen werden.

CONNECTION-MODE = *BY-RELOCATION / *BY-RESOLUTION

Bedeutung des Operanden und der Werte wie bei FOR-BS2000-VERSION=*FROM-CURRENT.

REQUIRED-COMPRESSION =

Gibt an, ob zur besseren Ausnutzung der Plattenkapazität eine Komprimierung der Textinformation (TXT) durchgeführt werden soll.

REQUIRED-COMPRESSION = *LAST-SAVE

Der BINDER übernimmt den Wert aus der letzten SAVE-LLM-Anweisung desselben Edit-Laufs. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt BINDER den Wert STD.

REQUIRED-COMPRESSION = *STD

Der BINDER übernimmt den Wert vom Operanden REQUIRED-COMPRESSION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt der BINDER den Wert NO.

REQUIRED-COMPRESSION = *NO

Eine Komprimierung der Textinformation wird nicht durchgeführt.

REQUIRED-COMPRESSION = *YES

Eine Komprimierung der Textinformation wird in jedem Fall durchgeführt.

NAME-COLLISION =

Gibt an, wie Namenskonflikte behandelt werden sollen, die während der Bearbeitung der SAVE-LLM-Anweisung auftreten.

NAME-COLLISION = *LAST-SAVE

Der BINDER übernimmt den Wert aus der letzten SAVE-LLM-Anweisung desselben Edit-Laufs. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt BINDER den Wert STD.

NAME-COLLISION = *STD

Der BINDER übernimmt den Wert vom Operanden NAME-COLLISION der letzten MODIFY-STD-DEFAULTS-Anweisung. Wurde diese Anweisung in demselben Edit-Lauf noch nicht angegeben, so übernimmt BINDER den Wert IGNORED.

NAME-COLLISION = *IGNORED

Namenskonflikte werden nicht behandelt.

NAME-COLLISION = *WARNING(...)

Der Benutzer erhält eine Warnung, falls es bei der Bearbeitung der SAVE-LLM-Anweisung zu Namenskonflikten kommt.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

NAME-COLLISION = *ERROR(...)

Legt fest, dass die Anweisung SAVE-LLM abgebrochen wird, wenn Namenskonflikte (behebbarer Fehler) auftreten.

SCOPE =

Definiert den Geltungsbereich der Festlegungen für die Behandlung von Namenskonflikten.

SCOPE = *WHOLE-LLM

Die Festlegungen für Namenskonflikte sind für das gesamte LLM gültig.

SCOPE = *SLICE

Die Festlegungen für Namenskonflikte sind nur auf Slice-Ebene gültig, d.h. Namenskonflikte zwischen verschiedenen Slices beachtet der BINDER nicht.

Dieser Wert kann nur für benutzerdefinierte Slices angegeben werden.

SYMBOL-DICTIONARY =

Legt fest, ob das LLM mit oder ohne Externadressbuch gespeichert wird.

Hinweis

Falls explizit oder implizit (über *LAST-SAVE) LOGICAL-STRUCTURE=*NONE gesetzt ist, muss SYMBOL-DICTIONARY=*NO gesetzt sein.

SYMBOL-DICTIONARY = *LAST-SAVE

Der BINDER übernimmt die Werte aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, übernimmt BINDER den Wert YES.

SYMBOL-DICTIONARY = *YES

Das Externadressbuch wird beim Speichern des LLM übernommen.

SYMBOL-DICTIONARY = *NO

Kein Externadressbuch wird beim Speichern des LLM übernommen.

RELOCATION-DATA =

Legt fest, ob das LLM mit oder ohne Relativierungsinformation gespeichert wird.

RELOCATION-DATA = *LAST-SAVE

Der BINDER übernimmt die Werte aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, übernimmt der BINDER den Wert YES.

RELOCATION-DATA = *YES

Die Relativierungsinformation wird beim Speichern des LLM übernommen.

RELOCATION-DATA = *NO

Keine Relativierungsinformation wird beim Speichern des LLM übernommen. Der Wert NO wird übergangen, wenn Slices aus Attributen gebildet werden.

RELOCATION-DATA = *UNRESOLVED-ONLY

Die Relativierungsinformation wird nur für die immer noch unbefriedigten Externverweise abgespeichert.

LOGICAL-STRUCTURE =

Legt fest, ob das LLM mit oder ohne Strukturinformation gespeichert wird.

LOGICAL-STRUCTURE = *LAST-SAVE

Der BINDER übernimmt die Werte aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, übernimmt der BINDER den Wert WHOLE-LLM.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird beim Speichern des LLM übernommen. Die logische Struktur des LLM kann dann nachträglich geändert werden.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Beim Speichern des LLM wird nur eine Struktur mit internem Namen (Root) und Bindemodulen (OMs) abgelegt.

LOGICAL-STRUCTURE = *NONE

Es wird nur der interne Name (Root) der Struktur abgelegt.

TEST-SUPPORT =

Legt fest, ob das LLM mit oder ohne LSD-Information gespeichert wird.



Falls explizit oder implizit (über *LAST-SAVE) LOGICAL-STRUCTURE=*NONE oder SYMBOL-DICTIONARY=*NO gesetzt ist, muss TEST-SUPPORT =*NO gesetzt sein.

Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *LAST-SAVE

Der BINDER übernimmt die Werte aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde noch keine Anweisung SAVE-LLM im Edit-Lauf angegeben, übernimmt der BINDER den Wert *YES.

TEST-SUPPORT = *YES

Die LSD-Information wird beim Speichern des LLM übernommen.

TEST-SUPPORT = *NO

Keine LSD-Information wird beim Speichern des LLM übernommen.

LOAD-ADDRESS =

Legt eine virtuelle Adresse fest, an die das LLM geladen werden soll. Dieser Operand wird übergangen, wenn Slices nach Attributen gebildet werden.

LOAD-ADDRESS = *LAST-SAVE

Der BINDER übernimmt die Adresse aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs.

Wurde noch keine entsprechende Anweisung SAVE-LLM im Edit-Lauf angegeben, wird folgender Wert für die Ladeadresse festgelegt:

- UNDEFINED, wenn das LLM mit einer Anweisung START-LLM-CREATION erzeugt wurde,
- die ursprüngliche Ladeadresse, wenn das LLM mit einer Anweisung START-LLM-UPDATE geändert wurde.

LOAD-ADDRESS = *UNDEFINED

Die virtuelle Adresse wird wie folgt festgelegt:

- auf den Wert 0, wenn der Operand RELOCATION-DATA den Wert *NO oder *UNRESOLVED-ONLY hat,
- oberhalb 16 MByte auf den Wert des Klasse-2-Systemparameters BLSLDPXS, wenn dieser bei der Systeminstallation festgelegt wurde,
- auf den Wert X'FFFFFFFF' in allen anderen Fällen.

LOAD-ADDRESS = NULL

Als Ladeadresse wird die virtuelle Adresse 0 festgelegt.

LOAD-ADDRESS = <x-string 1..8>

Explizite Angabe der Ladeadresse.

Die Adresse muss auf Seitengrenze liegen, d.h. ein Vielfaches von 4096 (X'1000') sein. Davon abweichende Adressen richtet der BINDER automatisch auf Seitengrenze aus.

LOAD-ADDRESS = *BY-SLICES(...)

Angabe der Ladeadressen für einzelne Slices. Ladeadressen von Slices, die nicht explizit angegeben werden, werden von BINDER berechnet. Für LLMs, die nicht in PAM-Dateien ausgegeben werden, wird diese Angabe wie *UNDEFINED behandelt.

ADDRESSES = list-poss(40): *REGION(...) / *SLICE(...)

Angabe der Ladeadressen für Slices.

***REGION(...)**

Die Ladeadresse für eine Region wird festgelegt.

REGION-NAME = <structured-name 1..32>

Name der Region, für die eine Ladeadresse festgelegt wird.

REGION-ADDRESS = <x-string 1..8>

Ladeadresse für die Region.

***SLICE(...)**

Die Ladeadresse für eine Slice wird festgelegt.

SLICE-NAME = *ROOT / <structured-name 1..32>

Name der Slice, für die eine Ladeadresse festgelegt wird.

SLICE-ADDRESS = <x-string 1..8> / *BEHIND-SLICE(...)

Ladeadresse für die Region.

BEHIND-SLICE(...)*SLICE = <structured-name 1..32>**

Name einer Slice, die verwendet wird um die Ladeadresse der Slice zu berechnen, die in SLICE-NAME angegeben wurde.

Hinweis

Wenn der Anwender eine Ladeadresse für eine Slice festgelegt hat, berücksichtigt BINDER diese nur, wenn sie mindestens so groß ist, wie die kleinstmögliche Ladeadresse, die durch die physische Struktur des LLMs vorgegeben ist.

Beispiel

Slice X hat die Ladeadresse X'1000' und die Länge X'4000'. Slice Y schließt sich an Slice X an. Die kleinstmögliche Ladeadresse für Slice Y ist demnach X'4000' + X'1000' = x'5000'. Eine benutzerdefinierte Sliceadresse wird in diesem Fall von BINDER nur dann akzeptiert, wenn sie x'5000' ist oder größer. Wird eine kleinere Ladeadresse angegeben, ersetzt BINDER sie durch X'5000'.

ENTRY-POINT =

Gibt den Namen eines Symbols an, d.h. den Namen einer Adresse, auf die nach dem Laden des LLM verzweigt werden soll.

ENTRY-POINT = *LAST-SAVE

Der BINDER übernimmt den Namen des Symbols aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs.

Wurde noch keine entsprechende Anweisung SAVE-LLM im Edit-Lauf angegeben, wird folgender Name für das Symbol festgelegt:

- der Name aus dem END-Satz des *ersten* Bindemoduls, wenn das LLM mit einer Anweisung START-LLM-CREATION erzeugt wurde,
- der ursprüngliche Name, wenn das LLM mit einer Anweisung START-LLM-UPDATE geändert wurde.

ENTRY-POINT = *STD

Der Name des Symbols wird aus dem END-Satz des ersten Bindemoduls (OM) des LLM übernommen. Fehlt im END-Satz eine derartige Angabe, wird beim Laden zu der Adresse verzweigt, die durch das *erste* Byte des ersten Bindemoduls festgelegt ist.

ENTRY-POINT = *BY-MODULE(...)

Legt das Bindemodul fest, aus dessen END-Satz der Name des Symbols übernommen werden soll. Fehlt im END-Satz eine derartige Angabe, wird beim Laden zu der Adresse verzweigt, die durch das *erste* Byte des Bindemoduls festgelegt ist.

PATH-NAME = <text 1..255>

Pfadname des Bindemoduls (siehe [Seite 25ff](#)).

Wird der Pfadname eines Sub-LLM angegeben, wird das *erste* Bindemodul des Sub-LLM angenommen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

ENTRY-POINT = <text 1..32>

Explizite Angabe des Namens einer CSECT oder eines ENTRY für den Namen des Symbols.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

MAP = *LAST-SAVE / *YES / *NO

Legt fest, ob Listen mit Informationen über das gespeicherte LLM ausgegeben werden (siehe [Seite 141ff](#)).

Das Ausgabeziel und die Art der Listen sind bestimmt durch die Werte in der letzten vorhergehenden Anweisung MODIFY-MAP-DEFAULTS mit der Angabe MAP-NAME=*STD, bzw. ohne MAP-NAME-Angabe. Wurde keine Anweisung MODIFY-MAP-DEFAULTS mit MAP-NAME=*STD oder ohne den Operanden MAP-NAME angegeben, gelten die Standardwerte.

MAP = *LAST-SAVE

Der BINDER übernimmt den Wert aus der letzten Anweisung SAVE-LLM desselben Edit-Laufs. Wurde die Anweisung in diesem Edit-Lauf noch nicht gegeben, so übernimmt der BINDER den Wert *YES.

SET-EXTERN-RESOLUTION

Behandeln unbefriedigter Externverweise

Diese Anweisung legt fest, wie der BINDER verbleibende Externverweise behandeln soll, die nicht befriedigt werden können. Festgelegt werden kann, dass nicht befriedigte Externverweise zulässig oder nicht zulässig sind.

Sind nicht befriedigte Externverweise zulässig, werden sie beim Speichern des LLM übernommen. Dabei können die unbefriedigten Externverweise mit einer angegebenen Adresse besetzt werden. Falls nicht befriedigte Externverweise unzulässig sind, wird das LLM beim Speichern mit der Anweisung SAVE-LLM abgewiesen.

SET-EXTERN-RESOLUTION

```

SYMBOL-NAME = *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
SYMBOL-TYPE = *REFERENCES / list-poss(3): *EXTRN / *VCON / *WXTRN
SCOPE = *CURRENT-SUB-LLM / *EXPLICIT(...) / *WHOLE-LLM
  *EXPLICIT(...)
    |
    | WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
    |
    | EXCEPT-SUB-LLM = *NONE / list-poss(10): <c-string 1..255 with-low> / <text 1..255>
RESOLUTION = *STD / *BY-SYMBOL(...) / *MANDATORY
  *BY-SYMBOL(...)
    |
    | SYMBOL = <c-string 1..32 with-low> / <text 1..32>

```

SYMBOL-NAME =

Legt die Externverweise fest, die der BINDER behandeln soll, wenn sie nicht befriedigt werden können.

SYMBOL-NAME = *ALL

Alle Externverweise sollen behandelt werden.

SYMBOL-NAME = list-poss(40): <c-string 1..255> / <text 1..32>

Namen der Externverweise, die behandelt werden sollen. Die Angabe von Platzhaltern (Wildcards) ist zulässig. Achtung: BINDER prüft Sonderdatentyp <symbol> und <symbol-with-wild> (siehe [Seite 202](#)).

SYMBOL-TYPE = *REFERENCES / list-poss(3): *EXTRN / *VCON / *WXTRN

Legt den Typ der Externverweise fest, die behandelt werden sollen. Es können Externverweise (EXTRNs), V-Konstante (VCON) und bedingte Externverweise (WXTRNs) ausgewählt werden. Die Angabe *REFERENCES behandelt alle Typen von Externverweisen.

SCOPE =

Setzt einen oder mehrere Zeiger. Sie verweisen auf die Sub-LLMs in der logischen Struktur des LLM, in denen der BINDER unbefriedigte Externverweise behandeln soll.

SCOPE = *CURRENT-SUB-LLM

Der Zeiger verweist auf das aktuelle Sub-LLM (siehe Anweisung BEGIN-SUB-LLM-STATEMENTS).

SCOPE = *EXPLICIT(...)**WITHIN-SUB-LLM = *WHOLE-LLM / list-poss(10): <text 1..255>**

Die Zeiger verweisen auf die explizit angegebenen Sub-LLMs. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *WHOLE-LLM werden alle Sub-LLMs angesprochen.

EXCEPT-SUB-LLM = *NONE / list-poss(10): <text 1..255>

Erlaubt es, aus der im Operanden WITHIN-SUB-LLM angegebenen Liste einzelne Zeiger auszuschließen. Für <text 1..255> sind die Pfadnamen der Sub-LLMs anzugeben, die nicht berücksichtigt werden sollen.

Achtung: BINDER prüft Sonderdatentyp <path-name> (siehe [Seite 202](#)).

Bei Angabe *NONE werden keine Zeiger ausgeschlossen.

SCOPE = *WHOLE-LLM

Alle Sub-LLMs des aktuellen LLM werden berücksichtigt.

RESOLUTION =

Legt die Art fest, wie der BINDER die nicht befriedigten Externverweise behandeln soll.

RESOLUTION = *STD

Unbefriedigte Externverweise sind zulässig. Beim Speichern des LLM mit der Anweisung SAVE-LLM werden die unbefriedigten Externverweise übernommen.

RESOLUTION = *BY-SYMBOL(...)

Legt fest, dass unbefriedigte Externverweise mit einer Adresse besetzt werden.

SYMBOL = <c-string 1..32 with-low> / <text 1..32>

Explizite Angabe der Adresse.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

RESOLUTION = *MANDATORY

Unbefriedigte Externverweise sind nicht zulässig. Das LLM wird beim Speichern mit der Anweisung SAVE-LLM abgewiesen, wenn nicht befriedigte Externverweise vorhanden sind.

SET-USER-SLICE-POSITION

Festlegen der Lage einer Slice

Diese Anweisung vereinbart die physische Struktur für das LLM. Sie ist nur für LLMs mit benutzerdefinierten Slices zulässig.

Jede Anweisung bezeichnet Name und Lage einer Slice, in die der BINDER die Module bindet. Eine Slice kann auch an einer Region beginnen.

SET-USER-SLICE-POSITION

```

SLICE-NAME = *ROOT / <structured-name 1..32>
, MODE = *CREATE (...) / *UPDATE
  *CREATE(...)
    | POSITION = *BEHIND-SLICE (...) / *BEGIN-REGION(...)
    | *BEHIND-SLICE(...)
    | | SLICE = *CURRENT-SLICE / *ROOT / <structured-name 1..32>
    | *BEGIN-REGION(...)
    | | REGION = *CURRENT-REGION / <structured-name 1..32>
    | , NEW-REGION = *NO / *YES

```

SLICE-NAME =

Legt den Namen der Slice in der physischen Struktur fest.

SLICE-NAME = *ROOT

Die Slice ist die Root-Slice (%ROOT).

SLICE-NAME = <structured-name 1..32>

Explizite Angabe des Namens.

Der Name muss eindeutig innerhalb der physischen Struktur sein.

MODE =

Gibt an, ob die angegebene Slice neu ist oder in der physische Struktur schon vorhanden ist.

MODE = *CREATE(...)

Es wird eine neue Slice erzeugt.

POSITION =

Legt den Adresspegel der Slice in der physischen Struktur fest.

POSITION = *BEHIND-SLICE(...)**SLICE =**

Der BINDER fügt die betreffende Slice unmittelbar an die angegebene Slice an.

SLICE = *CURRENT-SLICE

Die Slice wird unmittelbar an die aktuelle Slice angefügt.

SLICE = *ROOT

Die Slice wird unmittelbar an die Root-Slice (%ROOT) angefügt.

SLICE = <structured-name 1..32>

Explizite Angabe der Slice, an die unmittelbar angefügt wird.

POSITION = *BEGIN-REGION(...)

Legt den Anfang einer Region fest, d.h. stellt sicher, dass die dort beginnende Slice vorangehende Slices nicht überlagert.

REGION =

Name der Region, an der die Slice beginnen soll.

REGION = *CURRENT-REGION

Aktuelle Region

Die aktuelle Region enthält die aktuelle Slice. Wenn die aktuelle Slice in der Region liegt, die die Root-Slice (%ROOT) enthält, muss der Name der Region explizit angegeben werden.

REGION = <structured-name 1..32>

Explizite Angabe der Region.

Der Name der Region muss eindeutig innerhalb der physischen Struktur sein.

NEW-REGION = *NO / *YES

Gibt an, ob die angegebene Region neu ist.

MODE = *UPDATE

Die Slice ist in der physischen Struktur schon vorhanden.

SHOW-DEFAULTS

Anzeigen von Standardwerten

Diese Anweisung ermöglicht es dem Benutzer, sich Standardwerte anzeigen zu lassen. Die Werte werden auf der Standardausgabe SYSOUT ausgegeben.

SHOW-DEFAULTS

```

STD-DEFAULTS = *YES / *NO
, CURRENT-DEFAULTS = *YES / *NO
, INCLUSION-DEFAULTS = *YES / *NO
, LAST-SAVE = *YES / *NO
, MAP-DEFAULTS = *YES (...) / *NO
  *YES(...)
    | MAP-NAME = *STD / *ALL / <structured-name 1..32>

```

STD-DEFAULTS = *YES / *NO

Anzeigen aller Standardwerte, die mit dem Operandenwert STD angegeben werden.

CURRENT-DEFAULTS = *YES / *NO

Anzeigen aller Standardwerte, die für CURRENT-Operanden gelten.

INCLUSION-DEFAULTS = *YES / *NO

Anzeigen aller Standardwerte, die beim Einfügen von Modulen gelten.

LAST-SAVE = *YES / *NO

Anzeigen aller Standardwerte, die beim letzten Abspeichern eines LLM festgelegt wurden.

MAP-DEFAULTS = *YES(...) / *NO

Gibt an, ob die Standardwerte für die Listenausgabe angezeigt werden.

MAP-DEFAULTS = *YES(...)

Die Standardwerte für die Listenausgabe werden angezeigt.

MAP-NAME =

Gibt den Namen der Liste an, deren Standardwerte angezeigt werden sollen.

MAP-NAME = *STD

Die Standardwerte für die Liste mit dem Standardnamen `BNDMAP.date.time.<tsn>` werden ausgegeben.

MAP-NAME = *ALL

Die Standardwerte für die Liste mit dem Standardnamen und die Standardwerte für alle selbst definierten Listen werden ausgegeben.

MAP-NAME = <structured-name 1..32>

Die Standardwerte für die selbst definierte Liste mit dem angegebenen Namen werden ausgegeben.

SHOW-LIBRARY-ELEMENTS

Anzeigen und Prüfen von Bibliothekselementen

Diese Anweisung ermöglicht es dem Benutzer, sich während des BINDER-Laufs über Bibliothekselemente (Bindemodule und LLMs) zu informieren. Er kann dabei prüfen, ob in den Bindemodulen bzw. LLMs enthaltene Symbole Namenskonflikte hervorrufen könnten. Die Informationen werden standardgemäß auf SYSLST ausgegeben, sie können aber durch den Benutzer auf andere Ausgabeziele umgeleitet werden. Mit Ausnahme der auf SYSLST ausgegebenen Liste sind die Listen ISAM-Dateien mit ISAM-Schlüsseln der Länge 8 (siehe auch [Seite 141](#)).

SHOW-LIBRARY-ELEMENTS

```

LIBRARY = *CURRENT-INPUT-LIB / *BLSLIB-LINK /
           list-poss(40): <filename 1..54 without-gen-vers> / *LINK(...)
           *LINK(...)
           |   LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
ELEMENT = *ALL (...) / list-poss(40): <composed-name 1..64>(…) / <c-string 1..64>(…)
           *ALL(…)
           |   VERSION = *ALL / *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
           <composed-name>(…)
           |   VERSION = *ALL / *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
           <c-string>(…)
           |   VERSION = *ALL / *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
,TYPE = (*L*R) / list-poss(2): *L / *R
,SYMBOL-NAME = *ALL / *NONE / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
,SYMBOL-TYPE = (*CSECT*ENTRY) / list-poss(2): *CSECT / *ENTRY
,SELECT = *ALL / *NAME-COLLISION

```

Fortsetzung ➔

```

,OUTPUT = *SYSLST (...) / *BY-SHOW-FILE(...) / <filename 1..54 without-gen-vers>(…) / *LINK(…) /
          *EXIT-ROUTINE(…)

*SYSLST(…)
  |
  | SYSLST-NUMBER = STD / <integer 1..99>
  |
  | ,LINES-PER-PAGE = 64 / <integer 10..2147483647> / *IGNORED
  |
  | ,LINE-SIZE = 72 / <integer 72..255>
  |
*BY-SHOW-FILE(…)
  |
  | FILE-NAME = *STD / <filename 1..54 without-gen-vers>
  |
  | ,DELETE-FILE = *YES / *NO
  |
  | ,LINE-SIZE = 72 / <integer 72..255>
  |
<filename>(…)
  |
  | LINE-SIZE = 72 / <integer 72..255>
  |
*LINK(…)
  |
  | LINK-NAME = BNDMAP / <structured-name 1..8> / <filename 1..8 without-gen>
  |
  | ,LINE-SIZE = 72 / <integer 72..255>
  |
*EXIT-ROUTINE(…)
  |
  | ROUTINE-NAME =
  |
  |   <c-string 1..32 with-low> / <text 1..32>
  |
  | ,LIBRARY = *BLSLIB-LINK / <filename 1..54 without-gen-vers> / *LINK(…)
  |
  |   *LINK(…)
  |     |
  |     | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
  |     |
  |     | ,FILE-NAME = *STD / <filename 1..54 without-gen-vers>
  |     |
  |     | ,LINE-SIZE = 72 / <integer 72..255>
  |     |
  |     | ,USER-PARAMETERS = *NONE / <c-string 1..255 with-low> / <text 1..255>
  |

```

LIBRARY =

Gibt die zu durchsuchende(n) Bibliothek(en) an. Dieser Operand ist obligatorisch.

LIBRARY = *CURRENT-INPUT-LIB

Die zu durchsuchende Bibliothek ist diejenige, aus der das letzte Element genommen wurde (mit den Anweisungen **START-LLM-UPDATE**, **INCLUDE-MODULES** oder **REPLACE-MODULES**). Der Geltungsbereich des Operanden bezieht sich auf einen Edit-Lauf.

LIBRARY = *BLSLIB-LINK

Genutzt werden die Bibliotheken mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99). Die Bibliotheken werden in der Reihenfolge nach aufsteigenden Werten „nn“ des Dateikettungsnamens durchsucht.

LIBRARY = list-poss(40): <filename 1..54 without-gen-vers>

Dateiname der Bibliothek, die durchsucht werden soll.

LIBRARY = *LINK(...)

Bezeichnet eine Bibliothek mit dem Dateikettungsnamen.

LINKNAME = <structured-name 1..8> / <filename 1..8 without-gen>

Dateikettungsname der Bibliothek, die durchsucht werden soll.

ELEMENT =

Legt den Elementnamen und die Elementversion der Module fest, die überprüft werden sollen.

ELEMENT = *ALL(...)

Alle Elemente der angegebenen Bibliothek werden überprüft. Die Elemente werden in der Reihenfolge bearbeitet, wie sie in der Bibliothek angeordnet sind.

VERSION =

Gibt die Elementversion des Moduls an. Die Elementversion ist nur für Programmbibliotheken gültig.

VERSION = *ALL

Alle Versionen von Bibliothekselementen gleichen Namens werden geprüft.

VERSION = *HIGHEST-EXISTING

Der BINDER übernimmt als Elementversion den Standardwert für die höchste Version bei Programmbibliotheken (siehe Handbuch „LMS“ [3]).

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

ELEMENT = <composed-name 1..64>(...)

Explizite Angabe des Elementnamens und der Elementversion.

Achtung: BINDER prüft Sonderdatentypen <element-name> und <element-version> (siehe [Seite 202](#))

VERSION = *ALL / *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>

Gibt die Elementversion des Moduls an. Die Elementversion ist nur für Programmbibliotheken gültig.

Bedeutung der Operanden siehe oben.

ELEMENT = <c-string 1..64>(…)

Explizite Angabe des Elementnamens und der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-name> und <element-version> (siehe [Seite 202](#)).

Bedeutung der Operanden siehe oben.

TYPE =

Legt fest, welcher Typ von Elementen beim Prüfen berücksichtigt werden soll.

TYPE = (*L,*R)

Sowohl LLMs als auch Bindemodule werden geprüft. Ist für ein LLM der gleiche Name wie für ein Bindemodul angegeben, wird das *LLM* geprüft.

TYPE = (*R,*L)

Sowohl LLMs als auch Bindemodule werden geprüft. Ist für ein LLM der gleiche Name wie für ein Bindemodul angegeben, wird das *Bindemodul* geprüft.

TYPE = *R

Nur Bindemodule werden geprüft.

TYPE = *L

Nur LLMs werden geprüft.

SYMBOL-NAME =

Gibt die Symbole des Bibliothekselementes an, die berücksichtigt werden sollen.

SYMBOL-NAME = *ALL

Alle Symbole werden berücksichtigt.

SYMBOL-NAME = *NONE

Keine Symbole werden berücksichtigt. Bei der Ausgabe des Bibliotheksinhaltes erscheinen dann nur LLMs (ohne zugehörige Symbole).

SYMBOL-NAME = list-poss(40): <c-string 1..255> / <text 1..32>

Gibt die Namen der Symbole an, die berücksichtigt werden sollen. Die Angabe von Platzhaltern (Wildcards) ist zulässig.

Achtung: BINDER prüft Sonderdatentypen <symbol> und <symbol-with-wild> (siehe [Seite 202](#)).

SYMBOL-TYPE = (*CSECT,*ENTRY) / list-poss(2): *CSECT / *ENTRY

Legt fest, welchen Typen von Symbolen berücksichtigt werden sollen. Es kann eine Liste von CSECTs und ENTRYs oder nur CSECTs bzw. nur ENTRYs angefordert werden.

SELECT =

Gibt an, was mit den bei SYMBOL-NAME angegebenen Symbolen geschehen soll.

SELECT = *ALL

Alle mit bei SYMBOL-NAME angegebenen Symbole werden aufgelistet.

SELECT = *NAME-COLLISION

Aus den bei SYMBOL-NAME festgelegten Symbolen werden diejenigen herausgesucht, die einen Namenskonflikt verursachen könnten, wenn das betrachtete Element in das gerade bearbeitete LLM eingebaut wird.

BINDER sucht auch in dem gerade bearbeiteten LLM nach Duplikaten.

Wenn ein Duplikat in dem LLM entdeckt wird, werden zusätzlich die folgenden drei Zeilen ausgegeben:

- "IN THE LLM <LLM name> CURRENTLY IN PROCESS".
- Typ und Name des Objekts, das das Symbol beinhaltet.
- Typ des Symbols im Objekt.

OUTPUT =

Legt das Ausgabeziel für die Listen fest.

OUTPUT = *SYSLST(...)

Das Ausgabeziel ist eine Systemdatei SYSLST.

SYSLST-NUMBER =**SYSLST-NUMBER = *STD**

Es gilt die Systemdatei SYSLST.

SYSLST-NUMBER = <integer 1..99>

Es gilt eine Systemdatei aus der Menge SYSLST01 bis SYSLST99, deren Nummer hier anzugeben ist.

LINES-PER-PAGE = 64 / <integer 10..2147483647> / IGNORED

Legt die Anzahl der Zeilen pro Seite fest. Die Anzahl der Zeilen pro Seite ist für die Generierung von Seitenvorschüben (form-feed) notwendig.

Bei LINES-PER-PAGE=*IGNORED wird *kein* Seitenvorschub gemacht.

LINE-SIZE = 72 / <integer 72..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = *BY-SHOW-FILE(...)

Das Ausgabeziel ist eine durch ihren Dateinamen festgelegte ISAM-Datei. Die Datei wird anschließend automatisch mit dem Kommando SHOW-FILE eröffnet (siehe Handbuch „Kommandos“ [5]). Die in der Datei enthaltenen ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

FILE-NAME =

Legt den Dateinamen der ISAM-Datei fest.

FILE-NAME = *STD

Protokolliert wird in die Datei mit dem Standarddateinamen
 BNDMAP.date.time.<tsn>

Das Datum „date“ hat das Format yyyy-mm-ddjjj

yyyy Jahr
 mm Monat
 dd Tag
 jjj Tag des Jahres

Die Tageszeit „time“ hat das Format hhmmss

hh Stunden
 mm Minuten
 ss Sekunden

Die Angabe „<tsn>“ bedeutet die TSN (Task Sequence Number) der aktuellen Task.

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

Explizite Angabe des Dateinamens.

DELETE-FILE = *YES / *NO

Legt fest, ob die Datei nach Ausführung des Kommandos SHOW-FILE gelöscht wird.

LINE-SIZE = 72 / <integer 72..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = <filename 1..54 without-gen-vers>

Das Ausgabeziel ist eine durch den angegebenen Dateinamen festgelegte ISAM-Datei. Die darin enthaltenen ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

LINE-SIZE = 72 / <integer 72..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = *LINK(...)

Das Ausgabeziel ist eine durch den Dateikettungsname festgelegte ISAM-Datei. Die darin enthaltenen ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

LINK-NAME = BNDMAP / <structured-name 1..8> / <filename 1..8 without-gen>

Legt den Dateikettungsname fest. Der Standard-Dateikettungsname ist BNDMAP.

LINE-SIZE = 72 / <integer 72..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = *EXIT-ROUTINE(...)

Die Liste wird in die bei FILE-NAME angegebene ISAM-Datei ausgegeben. Das Unterprogramm wird mit dem Makroaufruf BIND geladen. Zu Registerkonventionen und den Parametern für den BIND-Makroaufruf siehe SHOW-MAP.

ROUTINE-NAME = <text 1..32>

Name des Unterprogramms, das aufgerufen werden soll.

LIBRARY =

Erlaubt die Angabe der Bibliothek, die das Unterprogramm enthält.

LIBRARY = *BLSLIB-LINK

Bibliotheken mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99) werden genutzt. Die Bibliotheken werden in der Reihenfolge nach aufsteigenden Werten „nn“ des Dateikettungsnamens durchsucht.

LIBRARY = <filename 1..54 without-gen-vers>

Explizite Angabe des Bibliotheksnamens.

LIBRARY = *LINK(...)

Die Bibliothek wird über den Dateikettungsnamen angegeben.

LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>

Dateikettungsname der Bibliothek.

FILE-NAME = *STD / <filename 1..54 without-gen-vers>

Legt den Namen (auch Dateikettungsnamen) der ISAM-Datei fest, in die die Liste ausgegeben werden soll. Standardmäßig wird der Dateiname BNDMAP.date.time.<tsn> verwendet (Format siehe Operand OUTPUT=BY-SHOW-FILE(...)). Die in der Datei enthaltenen ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

LINE-SIZE = 72 / <integer 72..255>

Legt die Anzahl der Zeichen pro Zeile fest.

USER-PARAMETERS = *NONE / <c-string 1..255> / <text 1..255>

Gibt die Parameter an, die dem Makroaufruf BINDER übergeben werden sollen.

SHOW-MAP

Ausgeben von Listen

Diese Anweisung gibt Listen mit Informationen über das aktuelle LLM aus (siehe Seite 141ff).

Der Wert ***MAP-DEFAULT** entspricht dem Wert, der mit einer vorhergehenden Anweisung MODIFY-MAP-DEFAULTS mit gleicher MAP-NAME-Angabe für denselben Operanden festgelegt wurde. Wurde bisher noch keine Anweisung MODIFY-MAP-DEFAULTS mit gleicher MAP-NAME-Angabe angegeben, wird der erste auf den Wert MAP-DEFAULT folgende Operandenwert angenommen.

Der Benutzer kann sich neben der Standardliste ***STD** auch andere (benannte) Listen ausgeben lassen, deren Standardwerte er mit der Anweisung MODIFY-MAP-DEFAULTS MAP-NAME= <structured-name 1..32> festgelegt hat.

SHOW-MAP

```

MAP-NAME = *STD / <structured-name 1..32>
, USER-COMMENT = *MAP-DEFAULT / *NONE / <c-string 1..255 with-low>
, HELP-INFORMATION = *MAP-DEFAULT / *YES / *NO
, GLOBAL-INFORMATION = *MAP-DEFAULT / *YES / *NO
, LOGICAL-STRUCTURE = *MAP-DEFAULT / *YES(...) / *NO
  *YES(...)
    |
    | RESOLUTION-SCOPE = *MAP-DEFAULT / *YES / *NO
    | , HSI-CODE = *MAP-DEFAULT / *YES / *NO / *X86
, PHYSICAL-STRUCTURE = *MAP-DEFAULT / *YES / *NO
, PROGRAM-MAP = *MAP-DEFAULT / *PARAMETERS(...) / *NO
  *PARAMETERS(...)
    |
    | DEFINITIONS = *MAP-DEFAULT / *ALL / *NONE / list-poss(5): *MODULE / *CSECT /
    | *ENTRY / *COMMON / *XDSECT-D
    | , INVERTED-XREF-LIST = *MAP-DEFAULT / *NONE / *ALL / list-poss(4): *EXTRN / *VCON /
    | *WXTRN / *XDSECT-R
    | , REFERENCES = *MAP-DEFAULT / *ALL / *NONE / list-poss(4): *EXTRN / *VCON /
    | *WXTRN / *XDSECT-R

```

Fortsetzung ➔

```

,UNRESOLVED-LIST = *MAP-DEFAULT / *SORTED(...) / *YES(...) / *NO
  *SORTED(...)
    | WXTRN = *YES / *NO
    | ,NOREF = *NO / *YES
  *YES(...)
    | WXTRN = *YES / *NO
    | ,NOREF = *NO / *YES
,SORTED-PROGRAM-MAP = *MAP-DEFAULT / *NO / *YES
,PSEUDO-REGISTER = *MAP-DEFAULT / *NO / *YES
,UNUSED-MODULE-LIST = *MAP-DEFAULT / *NO / *YES
,DUPLICATE-LIST = *MAP-DEFAULT / *NO / *YES(...)
  *YES(...)
    | INVERTED-XREF-LIST = *YES / *NO
,MERGED-MODULES = *MAP-DEFAULT / *YES / *NO
,INPUT-INFORMATION = *MAP-DEFAULT / *YES / *NO
,STATEMENT-LIST = *MAP-DEFAULT / *NO / *YES
,OUTPUT = *MAP-DEFAULT / *SYSLST(...) / *BY-SHOW-FILE(...) / <filename 1..54 without-gen-vers>(...) /
  *LINK(...) / *EXIT-ROUTINE(...)
  *SYSLST(...)
    | SYSLST-NUMBER = *STD / <integer 1..99>
    | ,LINES-PER-PAGE = 64 / <integer 10..2147483647> / *IGNORED
    | ,LINE-SIZE = 136 / <integer 132..255>
  *BY-SHOW-FILE(...)
    | FILE-NAME = *STD / <filename 1..54 without-gen-vers>
    | ,DELETE-FILE = *YES / *NO
    | ,LINE-SIZE = 136 / <integer 132..255>
  <filename 1..54 without-gen-vers>(...)
    | LINE-SIZE = 136 / <integer 132..255>
  *LINK(...)
    | LINK-NAME = BNDMAP / <structured-name 1..8> / <filename 1..8 without-gen>
    | ,LINE-SIZE = 136 / <integer 132..255>

```

Fortsetzung ➡

```

*EXIT-ROUTINE(...)
  |
  | ROUTINE-NAME = <c-string 1..32 with-low> / <text 1..32>
  |
  | ,LIBRARY = *BLSLIB-LINK / <filename 1..54 without-gen-vers> / *LINK(...)
  |
  |   *LINK(...)
  |   |
  |   | LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>
  |   |
  |   | ,FILE-NAME = *STD / <filename 1..54 without-gen-vers>
  |   |
  |   | ,LINE-SIZE = 136 / <integer 132..255>
  |   |
  |   | ,USER-PARAMETERS = *NONE / <c-string 1..255 with-low> / <text 1..255>
  |
  |

```

MAP-NAME =

Gibt den Namen der auszugebenden Liste an.

MAP-NAME = *STD

Die Standardliste mit dem Namen `BNDMAP.date.time.<tsn>` wird ausgegeben.

MAP-NAME = <structured-name 1..32>

Eine vom Benutzer selbst mit `MODIFY-MAP-DEFAULTS` definierte Liste wird ausgegeben.

USER-COMMENT = *MAP-DEFAULT / *NONE / <c-string 1..255 with-low>

Legt fest, ob in den Listen eine Kopfzeile mit Benutzerkommentar ausgegeben wird. Die Kopfzeile wird zu Beginn einer jeden Liste wiederholt.

HELP-INFORMATION = *MAP-DEFAULT / *YES / *NO

Legt fest, ob eine Liste der Abkürzungen ausgegeben wird.

In dieser Liste sind die Abkürzungen erklärt, die in den nachfolgenden Listen verwendet werden (z.B. SD für CSECT).

GLOBAL-INFORMATION = *MAP-DEFAULT / YES / NO

Legt fest, ob die Header-Informationen mit auszugeben sind.

LOGICAL-STRUCTURE = *MAP-DEFAULT / *YES(...) / *NO

Legt fest, ob eine Übersicht über die logische Struktur des LLM protokolliert wird.

LOGICAL-STRUCTURE = *YES(...)

Die logische Struktur des LLM wird protokolliert.

RESOLUTION-SCOPE = *MAP-DEFAULT / *YES / *NO

Gibt an, ob die Prioritätsklassen, die dem logischen Knoten des LLMs zugeordnet sind, in der Ausgabe enthalten sind.

HSI-CODE = *MAP-DEFAULT / *YES / *NO / *X86

Gibt an, ob die HSI-Code-Angabe in der Ausgabe enthalten ist.

PHYSICAL-STRUCTURE = *MAP-DEFAULT / *YES / *NO

Legt fest, ob eine Übersicht über die physische Struktur des LLM protokolliert wird.

PROGRAM-MAP = *MAP-DEFAULT / *PARAMETERS(...) / *NO

Legt fest, ob eine Programmübersicht protokolliert wird.

Hinweis

Unabhängig vom Wert des Operanden PROGRAM-MAP wird die Programmübersicht mit Modulinformationen immer ausgegeben, wenn UNUSED-MODULE-LIST=*YES gesetzt ist.

PROGRAM-MAP = *PARAMETERS(...)

Legt den Inhalt der Programmübersicht fest.

DEFINITIONS = *MAP-DEFAULT / *ALL / *NONE / list-poss(5): *MODULE / *CSECT / *ENTRY / *COMMON / *XDSECT-D

Legt fest, welche Programmdefinitionen die Programmübersicht enthält.

DEFINITIONS = *ALL

Die Programmübersicht enthält alle unten aufgeführten Programmdefinitionen.

DEFINITIONS = *NONE

Die Programmübersicht enthält keine Programmdefinitionen.

DEFINITIONS = *MODULE

Die Programmübersicht enthält Modulinformationen.

DEFINITIONS = *CSECT

Die Programmübersicht enthält Definitionen von Programmabschnitten.

DEFINITIONS = *ENTRY

Die Programmübersicht enthält Definitionen von Einsprungstellen.

DEFINITIONS = *COMMON

Die Programmübersicht enthält Definitionen von COMMON-Bereichen.

DEFINITIONS = *XDSECT-D

Die Programmübersicht enthält Definitionen von externen Pseudoabschnitten.

INVERTED-XREF-LIST = *MAP-DEFAULT / *NONE / *ALL / list-poss(4): *EXTRN / *VCON / *WXTRN / *XDSECT-R

Legt den Inhalt einer Querverweisliste fest, die für jedes Modul die befriedigten Referenzen mit Verweisen auf die zugehörigen Programmdefinitionen enthält.

INVERTED-XREF-LIST = *NONE

Es wird keine Querverweisliste ausgegeben.

INVERTED-XREF-LIST = *ALL

Die Querverweisliste enthält für jedes Modul alle unten aufgeführten befriedigten Referenzen.

INVERTED-XREF-LIST = *EXTRN

Die Querverweisliste enthält für jedes Modul die befriedigten EXTRNs.

INVERTED-XREF-LIST = *VCON

Die Querverweisliste enthält für jedes Modul die befriedigten V-Konstanten.

INVERTED-XREF-LIST = *WXTRN

Die Querverweisliste enthält für jedes Modul die befriedigten bedingten Externverweise.

INVERTED-XREF-LIST = *XDSECT-R

Die Querverweisliste enthält für jedes Modul die befriedigten Pseudoabschnitt-Referenzen.

**REFERENCES = *MAP-DEFAULT / *ALL / *NONE /
list-poss(4): *EXTRN / *VCON / *WXTRN / *XDSECT-R**

Legt fest, welche Referenzliste die Programmübersicht enthält. Bedeutung der Operandenwerte siehe Operand INVERTED-XREF-LIST.

UNRESOLVED-LIST = *MAP-DEFAULT / *SORTED(...) / *YES(...) / *NO

Legt fest, ob eine Liste der unbefriedigten Externverweise protokolliert wird oder nicht und legt ihren Inhalt fest.

UNRESOLVED-LIST = *SORTED(...)

Unbefriedigte Externverweise werden sortiert ausgegeben.

WXTRN = *YES / *NO

Gibt an, ob bedingte Externverweise mit ausgegeben werden oder nicht.

NOREF = *NO / *YES

Gibt an, ob nicht referenzierte Externverweise mit ausgegeben werden oder nicht.

UNRESOLVED-LIST = *YES(...)

Die unbefriedigten Externverweise werden in der Reihenfolge ausgegeben, in der sie gefunden werden.

WXTRN = *YES / *NO

Gibt an, ob bedingte Externverweise mit ausgegeben werden oder nicht.

NOREF = *NO / *YES

Gibt an, ob nicht referenzierte Externverweise mit ausgegeben werden oder nicht.

UNRESOLVED-LIST = *NO

Unbefriedigte Externverweise werden nicht protokolliert.

SORTED-PROGRAM-MAP = *MAP-DEFAULT / *NO / *YES

Legt fest, ob eine sortierte Liste der Programmdefinitionen ausgegeben wird.

PSEUDO-REGISTER = *MAP-DEFAULT / *NO / *YES

Legt fest, ob eine sortierte Liste der Pseudo-Register ausgegeben wird.

UNUSED-MODULE-LIST = *MAP-DEFAULT / *NO / *YES

Legt fest, ob eine Liste der nicht benutzten Module ausgegeben wird.

DUPLICATE-LIST = *MAP-DEFAULT / *NO / *YES(...)

Legt fest, ob eine sortierte Liste der mehrfachen Programmdefinitionen ausgegeben wird.

DUPLICATE-LIST = *YES(...)

Eine Liste der mehrfachen Programmdefinitionen wird ausgegeben.

INVERTED-XREF-LIST = *YES / *NO

Legt fest, ob in der Liste der mehrfachen Programmdefinitionen Querverweise ausgegeben werden.

MERGED-MODULES = *MAP-DEFAULT / *YES / *NO

Gibt an, ob die gemischten Module in den Listen erscheinen sollen.

INPUT-INFORMATION = *MAP-DEFAULT / *YES / *NO

Legt fest, ob eine Liste mit Eingabeinformationen über das LLM ausgegeben wird.

STATEMENT-LIST = *MAP-DEFAULT / *NO / *YES

Legt fest, ob eine Liste der protokollierten BINDER-Anweisungen ausgegeben wird (siehe //START-STATEMENT-RECORDING und //STOP-STATEMENT-RECORDING). Falls eine protokollierte Anweisung nicht ordnungsgemäß beendet wurde, kann es sein, dass die Protokollierung dieser Anweisung unvollständig ist.

OUTPUT = *MAP-DEFAULT / *SYSLST(...) / *BY-SHOW-FILE(...) / <filename 1..54 without-gen-vers> / *LINK(...) /EXIT-ROUTINE(...)

Legt das Ausgabeziel für die Listen fest.

OUTPUT = *SYSLST(...)

Das Ausgabeziel ist eine Systemdatei SYSLST.

SYSLST-NUMBER =

SYSLST-NUMBER = *STD

Es gilt die Systemdatei SYSLST.

SYSLST-NUMBER = <integer 1..99>

Es gilt eine Systemdatei aus der Menge SYSLST01 bis SYSLST99, deren Nummer hier anzugeben ist.

LINES-PER-PAGE = 64 / <integer 10..2147483647>

Legt die Anzahl der Zeilen pro Seite fest.

LINE-SIZE = 136 / <integer 132..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = *BY-SHOW-FILE(...)

Das Ausgabeziel ist eine durch ihren Dateinamen festgelegte ISAM-Datei. Die Datei wird anschließend automatisch mit dem Kommando SHOW-FILE eröffnet (siehe Handbuch „Kommandos“ [5]). Die Datei enthält ISAM-Schlüssel die im Anhang beschrieben sind (siehe [Seite 369f](#)).

FILE-NAME =

Legt den Dateinamen der Datei fest.

FILE-NAME = *STD

Protokolliert wird in die Datei mit dem Standarddateinamen

`BNDMAP.date.time.<tsn>`

Das Datum „date“ hat das Format `yyyy-mm-ddjjj`

yyyy Jahr

mm Monat

dd Tag

jjj Tag des Jahres

Die Tageszeit „time“ hat das Format `hhmms`

hh Stunden

mm Minuten

ss Sekunden

Die Angabe „<tsn>“ bedeutet die TSN (Task Sequence Number) der aktuellen Task.

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

Explizite Angabe des Dateinamens.

DELETE-FILE = *YES / *NO

Legt fest, ob die Datei nach Ausführung des Kommandos SHOW-FILE gelöscht wird.

LINE-SIZE = 136 / <integer 132..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = <filename 1..54 without-gen-vers>

Das Ausgabeziel ist eine durch den angegebenen Dateinamen festgelegte ISAM-Datei. Die darin enthalten ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

LINE-SIZE = 136 / <integer 132...255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = *LINK(...)

Das Ausgabeziel ist eine durch den Dateikettungsnamen festgelegte ISAM-Datei. Die darin enthaltenen ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

LINK-NAME = BNDMAP / <structured-name 1..8> / <filename 1..8 without-gen>

Legt den Dateikettungsnamen fest. Der Standard-Dateikettungsname ist BNDMAP.

LINE-SIZE = 136 / <integer 132..255>

Legt die Anzahl der Zeichen pro Zeile fest.

OUTPUT = *EXIT-ROUTINE(...)

Das Ausgabeziel ist ein benutzereigenes Unterprogramm, das von BINDER mit dem DBL-Makroaufruf BIND nachgeladen wird (BIND-Makro: siehe Handbuch „BLSSERV Binde-lader-Starter“ [1]). Folgende Registerkonventionen sind zu beachten:

Register 1: enthält die Adresse eines Feldes mit 2 Elementen
(Eingabe)

Element 1:	Adresse des max. 54 Zeichen langen Dateinamens
Element 2:	Adresse der max. 255 Zeichen langen Parameterliste

Register 1: enthält den Standard-Returncode des BIND-Makros
(Ausgabe)

Register 14: Rücksprungadresse

ROUTINE-NAME = <text 1..32>

Gibt den Namen des benutzereigenen Unterprogrammes an.

LIBRARY =

Angabe der Bibliothek, die das benutzereigene Unterprogramm enthält.

LIBRARY = *BLSLIB-LINK

In den Bibliotheken mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99) wird nach dem Unterprogramm gesucht. Die Bibliotheken werden in der Reihenfolge nach aufsteigenden Werten „nn“ des Dateikettungsnamens durchsucht.

LIBRARY = <filename 1..54 without-gen-vers>

Explizite Angabe des Bibliotheksnamens.

LIBRARY = *LINK(...)

Die Bibliothek wird über den Dateikettungsnamen angegeben.

LINK-NAME = <structured-name 1..8>

Dateikettungsname der Bibliothek.

FILE-NAME = *STD / <filename 1..54 without-gen-vers>

Legt den Namen (auch Dateikettungsnamen) der ISAM-Datei fest, in die die Liste ausgegeben werden soll. Standardmäßig wird der Dateiname

BNDMAP.date.time.<tsn> verwendet (Format siehe Operand OUTPUT=BY-SHOW-FILE(...)). Die in der Datei enthaltenen ISAM-Schlüssel sind im Anhang beschrieben ([Seite 369f](#)).

LINE-SIZE = 136 / <integer 132..255>

Legt die Anzahl der Zeichen pro Zeile fest.

USER-PARAMETERS = *NONE / <c-string 1..255 with-low> / <text 1..255>

Gibt die Parameter an, die dem DBL-Makroaufruf BIND übergeben werden sollen.

SHOW-SYMBOL-INFORMATION

Ausgeben von Symbolinformationen

Diese Anweisung gibt Informationen über Symbole auf SYSOUT aus.

Folgende Informationen können ausgegeben werden:

- die logische Position der Symbole im LLM (d.h. der Name des Moduls, in dem sie sich befinden),
- die Attribute der Symbole,
- die Adresse der Slice, in der sich die Symbole befinden.

Diese Informationen können angefordert werden für:

- alle sichtbaren Programmdefinitionen,
- die initialisierten COMMON-Bereiche,
- die befriedigten Externverweise
- alle sichtbaren Programmdefinitionen, die Duplikate haben,
- die unbefriedigten Externverweise.

SHOW-SYMBOL-INFORMATION

```

SYMBOL-NAME = *ALL / list-poss(40): <c-string 1..255 with-low> / <text 1..32>
,INFORMATION = *LOGICAL-POSITION / *ALL /
                list-poss(3): *LOGICAL-POSITION / *ATTRIBUTES / *ADDRESS
,SELECT = *ALL / *COMMON-PROMOTION / *EXTERN-RESOLUTION / *DUPLICATE-LIST /
          *UNRESOLVED-LIST(...)
          *UNRESOLVED-LIST(...)
          | REFERENCE-TYPE = *ALL / list-poss(4): *EXTRN / *VCON / *WXTRN / *XDSECT-R

```

SYMBOL-NAME =

Gibt die Namen der Symbole an, über die Informationen angezeigt werden.

SYMBOL-NAME = *ALL

Informationen werden über alle Symbole angezeigt.

SYMBOL-NAME = list-poss(40): <c-string 1..255 with-low> / <text 1..32>

Informationen werden nur über Symbole mit dem angegebenen Namen angezeigt.

Die Angabe von Platzhaltern (Wildcards) ist zulässig.

Achtung: BINDER prüft Sonderdatentyp <symbol> und <symbol-with-wild> (siehe [Seite 202](#)).

INFORMATION =

Gibt an, welche Informationen über Symbole angezeigt werden.

INFORMATION = *LOGICAL-POSITION

Zeigt den Modulnamen an, in dem sich die Symbole befinden.

INFORMATION = *ALL

Alle Informationen werden ausgegeben.

INFORMATION = list-poss(3): *LOGICAL-POSITION / *ATTRIBUTES / *ADDRESS

Gibt die Liste der angeforderten Informationen an.

INFORMATION = *ATTRIBUTES

Die Adresse, die Länge und die Attribute von Programmdefinitionen werden ausgegeben.

INFORMATION = *ADDRESS

Der Name der Slice, in der die Symbole enthalten sind, wird ausgegeben.

SELECT =

Auswahl von Informationen über die Behandlung der Symbole.

SELECT = *ALL

Anzeigen von Informationen über alle sichtbaren Programmdefinitionen.

SELECT = *COMMON-PROMOTION

Angezeigt werden die initialisierten COMMON-Bereiche mit der CSECT, durch die sie initialisiert wurden.

SELECT = *EXTERN-RESOLUTION

Angezeigt werden die befriedigten Externverweise mit den Symbolen, durch die sie befriedigt wurden.

SELECT = *DUPLICATE-LIST

Anzeigen von Informationen über mehrfach verwendete Programmdefinitionen.

SELECT = *UNRESOLVED-LIST(...)

Anzeigen von unbefriedigten Externverweisen und ihrer logischen Position (d.h. der Modulnamen, in denen sich die unbefriedigten Externverweise befinden).

REFERENCE-TYPE = *ALL / list-poss(3): *EXTRN / *VCON / *WXTRN

Legt fest, welche Typen von Referenzen beim Anzeigen der Informationen einbezogen werden.

START-LLM-CREATION

Erzeugen eines LLM

Diese Anweisung erzeugt ein neues LLM im Arbeitsbereich. Der Arbeitsbereich wird dabei gelöscht. Für das LLM können folgende Merkmale vereinbart werden:

- interner Name (INTERNAL-NAME),
- interne Version (INTERNAL-VERSION),
- physische Struktur (SLICE-DEFINITION),
- Copyright-Information (COPYRIGHT),
- Verwendung der Strukturinformation und LSD-Information (INCLUSION-DEFAULTS).

In dieses aktuelle LLM fügt dann der BINDER die Module ein, die mit Anweisungen INCLUDE-MODULES festgelegt werden.

Das erzeugte LLM wird mit der Anweisung SAVE-LLM als Element vom Typ L in einer Programmbibliothek gespeichert.

START-LLM-CREATION

```

INTERNAL-NAME = <c-string 1..32 with-low> / <text 1..32>
,INTERNAL-VERSION = *UNDEFINED / <composed-name 1..24> / <c-string 1..24>
,SLICE-DEFINITION = SINGLE / *BY-ATTRIBUTES(...) / *BY-USER(...)
  *BY-ATTRIBUTES(...)
    | READ-ONLY = *NO / *YES
    | RESIDENT = *NO / *YES
    | PUBLIC = NO / *YES(...)
    | *YES(...)
    | | SUBSYSTEM-ENTRIES = *NONE / list-poss(40): <c-string 1..32 with-low> / <text 1..32>
    | RESIDENCY-MODE = *NO / *YES
  *BY-USER(...)
    | AUTOMATIC-CONTROL = *YES (...) / *NO
    | *YES(...)
    | | RELOAD-SLICE = *YES / *NO
    | EXCLUSIVE-SLICE-CALL = *NO / *YES
,COPYRIGHT = *PARAMETERS (...) / *NONE
  *PARAMETERS(...)
    | NAME = *SYSTEM-DEFAULT / <c-string 1..64 with-low>
    | YEAR = *CURRENT / <integer 1900..2100>
    | PATH-NAME = *NONE / <c-string 1..255 with-low> / <text 1..255>
    | ENTRY = *NONE / <c-string 1..32 with-low> / <text 1..32>
,INCLUSION-DEFAULTS = *PARAMETERS (...)
  *PARAMETERS(...)
    | LOGICAL-STRUCTURE = *WHOLE-LLM / *OBJECT-MODULES
    | TEST-SUPPORT = *NO / *YES

```

INTERNAL-NAME = <structured-name 1..32>

Legt den internen Namen des erzeugten LLM fest. Der interne Name bildet die Wurzel (root) in der logischen Struktur des LLM (siehe [Seite 17ff](#)). Der interne Name wird beim Speichern des LLM als Elementname in die Programmbibliothek eingetragen, wenn in der Anweisung SAVE-LLM für den Operanden ELEMENT entsprechende Werte gesetzt sind (siehe Anweisung SAVE-LLM).

INTERNAL-VERSION =

Legt die interne Version des erzeugten LLM fest. Die interne Version wird beim Speichern des LLM in die Programmbibliothek als Elementversion verwendet, wenn in der Anweisung SAVE-LLM für den Operanden VERSION entsprechende Werte gesetzt sind (siehe Anweisung SAVE-LLM auf [Seite 287](#)).

INTERNAL-VERSION = *UNDEFINED

Beim Speichern des LLM mit der Anweisung SAVE-LLM wird der Standardwert für die höchste Version bei Programmbibliotheken angenommen (siehe Handbuch „LMS“ [3]).

INTERNAL-VERSION = <composed-name 1..24> / <c-string 1..24>

Interne Version des LLM.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

SLICE-DEFINITION =

Legt die physische Struktur des LLM fest.

SLICE-DEFINITION = *SINGLE

Das LLM besteht aus einer Einzel-Slice.

SLICE-DEFINITION = *BY-ATTRIBUTES(...)

Das LLM besteht aus Slices, die durch Kombination der Attribute von CSECTs gebildet wurden (siehe [Seite 18ff](#)). Ist der Operand BY-ATTRIBUTES angegeben und sind alle Unteroperanden auf NO gesetzt, wird SINGLE angenommen. Die Operandenwerte von READ-ONLY, RESIDENT, PUBLIC und RESIDENCY-MODE bewirken nur das Zusammenfassen zu Slices. Sie haben keine Auswirkung auf die einzelnen CSECTs. Maximal können 16 verschiedene Slices durch Kombination der Attribute gebildet werden.

READ-ONLY = *NO / *YES

Legt fest, ob das Attribut READ-ONLY bei der Bildung der Slices berücksichtigt wird. Bei Angabe YES bildet der BINDER für CSECTs mit unterschiedlichem Attribut READ-ONLY getrennte Slices.

RESIDENT = *NO / *YES

Legt fest, ob das Attribut RESIDENT bei der Bildung der Slices berücksichtigt wird. Bei Angabe YES bildet der BINDER für CSECTs mit unterschiedlichem Attribut RESIDENT getrennte Slices.

PUBLIC =

Legt fest, ob das Attribut PUBLIC bei der Bildung der Slices berücksichtigt wird.

PUBLIC = *NO

Das Attribut PUBLIC wird bei der Bildung der Slices nicht berücksichtigt.

PUBLIC = *YES(...)

Der BINDER bildet für CSECTs mit unterschiedlichem Attribut PUBLIC getrennte Slices.

SUBSYSTEM-ENTRIES = *NONE / list-poss(40): <text 1..32>

Legt die Symbole (CSECT oder ENTRY) der PUBLIC-Slice fest, die zur Befriedigung von Externverweisen genutzt werden können, wenn die PUBLIC-Slice als dynamisches Subsystem (siehe Handbuch „Einführung in die Systembetreuung“ [9]) geladen wurde.

SUBSYSTEM-ENTRIES = *NONE

Keine Symbole aus diesem Subsystem (der PUBLIC-Slice) werden zur Befriedigung von Externverweisen genutzt.

SUBSYSTEM-ENTRIES = <text 1..32>

Name der CSECT oder des ENTRY in der als Subsystem geladenen PUBLIC-Slice, der zur Befriedigung von Externverweisen genutzt werden kann.

Achtung: BINDER prüft Sonderdatentyp <symbol> (siehe [Seite 202](#)).

RESIDENCY-MODE = *NO / *YES

Legt fest, ob das Attribut RMODE bei der Bildung der Slices berücksichtigt wird.

Bei Angabe *YES bildet der BINDER für CSECTs und mit unterschiedlichem Attribut RMODE getrennte Slices.

SLICE-DEFINITION = *BY-USER(...)

Die physische Struktur des LLM wird vom Benutzer mit Anweisungen SET-USER-SLICE-POSITION festgelegt (User defined Slices). Dabei können Überlagerungssegmente (Overlays) festgelegt werden.

AUTOMATIC-CONTROL = *YES(...) / *NO

Hat nur bei Überlagerungsstrukturen (Overlays) eine Bedeutung.

Gibt an, ob ein Steuermodul (Overlay Control Module, OCM) in das erzeugte LLM eingebunden wird, um das automatische Nachladen der Overlays zu steuern.

AUTOMATIC-CONTROL = *YES(...)

Ein OCM wird in das erzeugte LLM eingebunden, um das automatische Nachladen der Overlays zu steuern.

RELOAD-SLICE=

Legt fest, ob das geladene Slice nachgeladen wird, wobei das vorherige Slice im Hauptspeicher überschrieben wird.

RELOAD-SLICE= *YES

Ein bereits geladenes Slice wird nachgeladen, wobei das vorherige Slice im Hauptspeicher überschrieben wird.

RELOAD-SLICE= *NO

Ein bereits geladenes Slice wird unverändert im Hauptspeicher behalten.

EXCLUSIVE-SLICE-CALL =

Ist nur bei Überlagerungsstrukturen von Bedeutung und legt fest, ob Externverweise zwischen exklusiven Slices befriedigt werden sollen.

EXCLUSIVE-SLICE-CALL = *NO

gibt an, dass der BINDER Externverweise nur meldet und sie nicht befriedigt, falls er Verweise zwischen exklusiven Slices feststellt.

EXCLUSIVE-SLICE-CALL = *YES

veranlasst den BINDER, Externverweise zwischen exklusiven Slices zu befriedigen, d.h. der Benutzer nimmt evtl. auftretende Fehler in Kauf.

COPYRIGHT =

Legt die Copyright-Information fest, die in das erzeugte LLM eingetragen wird. Die Copyright-Information besteht aus Text und Jahreszahl.

COPYRIGHT = *PARAMETERS(...)**NAME =**

Text der Copyright-Information.

NAME = *SYSTEM-DEFAULT

Der Wert des Klasse-2-Systemparameters BLSCOPYN soll übernommen werden. Dieser Wert wird bei der Systeminstallation festgelegt (siehe Handbuch „Einführung in die Systembetreuung“ [9]).

NAME = <c-string 1..64>

Neuer Text der Copyright-Information. Besteht der Text aus Leerzeichen, wird keine Copyright-Information eingetragen.

YEAR =

Jahreszahl der Copyright-Information.

YEAR = *CURRENT

Aktuelle Jahreszahl.

YEAR = <integer 1900..2100>

Explizite Angabe der Jahreszahl.

COPYRIGHT = *NONE

Keine Copyright-Information wird eingetragen.

INCLUSION-DEFAULTS =

Legt die Verwendung der Strukturinformation und der LSD-Information fest. Dies ist der Standardwert, der in den Anweisungen INCLUDE-MODULES, REPLACE-MODULES und RESOLVE-BY-AUTOLINK desselben *Edit-Laufs* benutzt wird, falls in diesen Anweisungen keine spezifischen Werte angegeben sind. Strukturinformation und LSD-Information werden beim Speichern des LLM nur übernommen, wenn dies sowohl in der Anweisung SAVE-LLM wie in vorhergehenden Anweisungen INCLUDE-MODULES, REPLACE-MODULES oder RESOLVE-BY-AUTOLINK verlangt wird.

INCLUSION-DEFAULTS = *PARAMETERS(...)**LOGICAL-STRUCTURE =**

Legt fest, ob beim Einfügen oder Ersetzen von Modulen die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird in das aktuelle LLM übernommen.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Die Strukturinformation wird nicht übernommen. Im aktuellen LLM wird nur eine Struktur mit Bindemodulen (OMs) aufgebaut.

TEST-SUPPORT =

Legt fest, ob beim Einfügen oder Ersetzen von Modulen die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.



Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

START-LLM-UPDATE Ändern eines LLM

Diese Anweisung ändert ein LLM, das in einer Programmbibliothek als Element vom Typ L gespeichert ist. Die Anweisung START-LLM-UPDATE liest das LLM aus der Programmbibliothek in den BINDER-Arbeitsbereich. Nach dem Einlesen wird das LLM zum *aktuellen LLM*, d.h. es hat den gleichen Zustand wie vor dem Speichern in der Programmbibliothek. Das aktuelle LLM kann anschließend im Arbeitsbereich verarbeitet werden.

Die Verarbeitung des aktuellen LLM wird beendet, ohne dass es implizit gespeichert wird. Er wird mit der Anweisung SAVE-LLM wieder in einer Programmbibliothek als Element vom Typ L gespeichert. Falls das neue Element den gleichen Elementnamen und die gleiche Elementversion behält und `OVERWRITE=YES` angegeben wurde, wird das bisherige Element in der Programmbibliothek überschrieben. Der Benutzer sollte daran denken, dass bei START-LLM-UPDATE (bzw. auch bei INCLUDE-/REPLACE-MODULES) die Eingabequelle als nur lesbar eröffnet wird und auch andere Tasks lesend darauf zugreifen können. Das LLM kann in diesem Fall nicht mit demselben Elementnamen und derselben Elementversion abgespeichert werden.

START-LLM-UPDATE

```

MODULE-CONTAINER = *LIBRARY-ELEMENT (...) / *FILE(...)

*LIBRARY-ELEMENT(...)
    |
    | LIBRARY = *CURRENT / <filename 1..54 without-gen-vers> / *LINK(...)
    |
    | *LINK(...)
    | |
    | | LINK-NAME =
    | | <structured-name 1..8> / <filename 1..8 without-gen>
    |
    | ,ELEMENT = <composed-name 1..64>(…) / <c-string 1..64>(…)
    |
    | <composed-name>(…)
    | |
    | | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
    | | <c-string>(…)
    | |
    | | VERSION = *HIGHEST-EXISTING / <composed-name 1..24> / <c-string 1..24>
    |
*FILE(…)
    |
    | FILE-NAME =
    | <filename 1..54 without-gen-vers> / *LINK(…)
    |
    | *LINK(…)
    | |
    | | LINK-NAME =
    | | <structured-name 1..8> / <filename 1..8 without-gen>
    |
,INCLUSION-DEFAULTS = *PARAMETERS (...)

*PARAMETERS(…)
    |
    | LOGICAL-STRUCTURE = *UNCHANGED / *WHOLE-LLM / *OBJECT-MODULES
    |
    | ,TEST-SUPPORT = *UNCHANGED / *NO / *YES

```

MODULE-CONTAINER =

Legt fest, wo das LLM abgespeichert ist.

MODULE-CONTAINER = *LIBRARY-ELEMENT(...)

Das LLM ist in einer Programmbibliothek.

LIBRARY =

Gibt die Programmbibliothek an, die das LLM, das geändert werden soll, als Element enthält.

LIBRARY = *CURRENT

Es soll die Programmbibliothek verwendet werden, die in der letzten vorhergehenden Anweisung START-LLM-UPDATE oder SAVE-LLM angegeben wurde. Der Geltungsbereich des Operanden bezieht sich auf einen *Edit-Lauf*.

LIBRARY = <filename 1..54 without-gen-vers>

Dateiname der Programmbibliothek, die das LLM als Element enthält.

LIBRARY = *LINK(...)

Bezeichnet eine Bibliothek mit dem Dateikettungsnamen

LINK-NAME = <structured-name 1..8> / <filename 1..8 without-gen>

Dateikettungsname der Programmbibliothek.

ELEMENT =

Elementname und Elementversion des LLM in der Programmbibliothek.

ELEMENT = <composed-name 1..64>(…)

Elementname des LLM.

Achtung: BINDER prüft Sonderdatentyp <element-name> (siehe [Seite 202](#)).

VERSION =

Elementversion des LLM.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

VERSION = *HIGHEST-EXISTING

Es wird der Standardwert für die höchste Version in der Programmbibliothek angenommen (siehe Handbuch „LMS“ [3]).

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

ELEMENT = <c-string 1..64>(…)

Elementname des LLM.

Achtung: BINDER prüft Sonderdatentyp <element-name> (siehe [Seite 202](#)).

VERSION =

Version des LLM.

VERSION = *HIGHEST-EXISTING

Es wird der Standardwert für die Versionsbezeichnung in der Programmbibliothek angenommen (siehe Handbuch „LMS“ [3]).

VERSION = <composed-name 1..24> / <c-string 1..24>

Explizite Angabe der Elementversion.

Achtung: BINDER prüft Sonderdatentyp <element-version> (siehe [Seite 202](#)).

MODULE-CONTAINER = *FILE(...)

Das LLM ist in einer PAM-Datei abgespeichert.

FILE-NAME =

Angabe der Datei, die das PAM-LLM enthält.

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

Name der PAM-Datei, die das LLM enthält.

FILE-NAME = *LINK(...)**LINK-NAME = <structured-name 1..8>**

Dateikettungsname der PAM-Datei, in der das LLM abgespeichert ist.

INCLUSION-DEFAULTS =

Legt die Verwendung der Strukturinformation und der LSD-Information fest. Dies ist der Standardwert, der in den Anweisungen INCLUDE-MODULES, REPLACE-MODULES und RESOLVE-BY-AUTOLINK desselben *Edit-Laufs* benutzt wird, falls in diesen Anweisungen keine spezifischen Werte angegeben sind. Strukturinformation und LSD-Information werden beim Speichern des LLM nur übernommen, wenn dies sowohl in den Anweisungen SAVE-LLM wie in vorhergehenden Anweisungen INCLUDE-MODULES, REPLACE-MODULES oder RESOLVE-BY-AUTOLINK verlangt wird.

INCLUSION-DEFAULTS = *PARAMETERS(...)**LOGICAL-STRUCTURE =**

Legt fest, ob beim Einfügen oder Ersetzen von Modulen die Strukturinformation aus den Modulen in das aktuelle LLM übernommen wird.

LOGICAL-STRUCTURE = *UNCHANGED

Es gilt der Wert, der beim Erzeugen des LLM in der Anweisung START-LLM-CREATION festgelegt wurde.

LOGICAL-STRUCTURE = *WHOLE-LLM

Die gesamte Strukturinformation wird in das aktuelle LLM übernommen.

LOGICAL-STRUCTURE = *OBJECT-MODULES

Die Strukturinformation wird nicht übernommen. Im aktuellen LLM wird nur eine Struktur mit Bindemodulen (OMs) aufgebaut.

TEST-SUPPORT =

Legt fest, ob beim Einfügen oder Ersetzen von Modulen die LSD-Information aus den Modulen in das aktuelle LLM übernommen wird.



Auskunft über das Vorhandensein von Test- und Diagnoseinformation gibt die Spalte „T&D“ in BINDER-Listen. Das Feld TEST-SUPPORT in BINDER-Listen zeigt nur die Einstellung dieses Operanden TEST-SUPPORT an.

TEST-SUPPORT = *UNCHANGED

Es gilt der Wert, der beim Erzeugen des LLM in der Anweisung START-LLM-CREATION festgelegt wurde.

TEST-SUPPORT = *NO

Die LSD-Information wird nicht übernommen.

TEST-SUPPORT = *YES

Die LSD-Information wird übernommen.

START-STATEMENT-RECORDING BINDER-Anweisungen protokollieren

Diese Anweisung startet die Anweisungsprotokollierung. Alle nach dieser Anweisung eingegebenen BINDER-Anweisungen werden im Speicher protokolliert. Mit der Anweisung //STOP-STATEMENT-RECORDING wird die Protokollierung wieder beendet. Die Ausgabe der protokollierten Anweisungen in BINDER-Listen steuert der Operand STATEMENT-LIST der Anweisung //SHOW-MAP.

START-STATEMENT-RECORDING
STATEMENT-FORM= *SHORT / *LONG ,RECORDING-MODE= *NEW / *EXTEND

STATEMENT-FORM =

Steuert die Protokollierung von Standardwerten.

STATEMENT-FORM = *SHORT

Nur die Operanden werden protokolliert, deren Wert vom Standardwert abweicht.

STATEMENT-FORM = *LONG

Alle Operanden werden protokolliert, auch wenn ihr Wert dem Standardwert entspricht.

RECORDING-MODE= *NEW / *EXTEND

Steuert die Behandlung von Protokolleinträgen für Anweisungen, die mit einer vorangegangenen //START-STATEMENT-RECORDING-Anweisung im selben BINDER-Lauf erzeugt wurden.

RECORDING-MODE= *NEW

Ältere Protokolleinträge werden aus dem Speicher gelöscht.

RECORDING-MODE= *EXTEND

Ältere Protokolleinträge bleiben erhalten.

Hinweise

Bei Speichersättigung wird die Protokollierung mit der Meldung BND2102 beendet.

Falls eine BINDER-Anweisung, die protokolliert werden soll, nicht ordnungsgemäß beendet wird, ist der entsprechende Protokolleintrag möglicherweise unvollständig.

STOP-STATEMENT-RECORDING

Protokollierung von BINDER-Anweisungen beenden

Diese Anweisung beendet die mit //START-STATEMENT-RECORDING eingeleitete Anweisungsprotokollierung. Nach dieser Anweisung eingegebene BINDER-Anweisungen werden nicht (mehr) im Speicher protokolliert.

Außerdem können mit dieser Anweisung die Einträge für bereits protokollierte Anweisungen aus dem Speicher gelöscht werden.

STOP-STATEMENT-RECORDING
DELETE-RECORDED-STMT= *NO / *YES

DELETE-RECORDED-STMT =

Steuert die Behandlung vorhandener Protokolleinträge.

DELETE-RECORDED-STMT= *NO

Die Protokolleinträge bleiben im Speicher erhalten.

DELETE-RECORDED-STMT= *YES

Die Protokolleinträge werden aus dem Speicher gelöscht.

8 Nutzungsmodelle für Bindelademodule (LLMs)

Dieses Kapitel beschreibt verschiedene Nutzungsmodelle für ein Bindelademodul oder einen Satz von Bindelademodulen.

Die beschriebenen Nutzungsmodelle orientieren sich an den Eigenschaften, die das erzeugte LLM haben muss; so unterscheidet sich beispielsweise das Modell für die Generierung eines selbstständigen Programms von dem Modell für die Generierung eines Moduls, das dynamisch gebunden und geladen werden muss.

Ohne den Einsatz von LLMs müssten verschiedene Programmformate mit unterschiedlichen BLS-Eigenschaften generiert werden:

- eine Phase (Programmdatei oder Bibliothekselement vom Typ C in einer Programmbibliothek) für ein selbstständiges Programm,
- ein Bindemodul (Bibliothekselement vom Typ R).

Bei Verwendung von LLMs entfällt diese Unterscheidung. Daher ist es erforderlich, verschiedene Nutzungsmodelle zu definieren.

Anhand der verschiedenen Nutzungsmöglichkeiten und der wichtigsten Betriebseigenschaften (dynamisches Binden/Laden, Performance, partielles Ersetzen, Rekonfigurierung) können folgende Nutzungsmodelle von LLMs unterschieden werden:

1. Programm
2. Modul
3. Programmbibliothek
4. Modulbibliothek

8.1 Programm

Zweck dieses Modells ist die Generierung von selbstständig ablauffähigen Programmen.

Haupteigenschaften

Dieses Modell hat folgende Haupteigenschaften:

- Es bietet die optimale Performance beim Laden.
- Im Programm enthaltene BLS-Metadaten müssen auf ein Minimum reduziert werden. Daher ist keine Modifizierung durch den BINDER (z.B. Anweisung REPLACE-MODULES) möglich.

Darüberhinaus bietet das LLM-Format die folgenden Merkmale:

- Die Länge der Symbolnamen ist nicht auf 8 Zeichen beschränkt.
- Ein LLM kann (ganz oder teilweise) als Shared Code geladen werden (z.B. als DSSM-Subsystem).

Die Reduzierung der BLS-Metadaten in einem Programm ist eine wesentliche Voraussetzung dafür, dass es bei der Generierung (oder beim Laden) eines Programms nicht zu einer unerwarteten Befriedigung von Externverweisen durch ein bereits vorhandenes LLM kommt.

Mögliche Rekonfigurationen

Für dieses Modell sind folgende Rekonfigurationen möglich:

- Installation einer kompletten neuen Programmversion (wie bei einer Phase). Der LLM-Container ist während der Ausführung nicht gesperrt
- dynamisches Entladen des als DSSM-Subsystem geladenen Programms (bzw. des PUBLIC-Teils des Programms) mit der Anweisung STOP-SUBSYSTEM und Laden einer neuen Version mit der Anweisung START-SUBSYSTEM
- dynamisches Entladen des als Shared Code des Benutzers geladenen Programms (bzw. des PUBLIC-Teils des Programms) mit dem DSHARE-Makro und Laden einer neuen Version mit dem ASHARE-Makro.

In den beiden letzten Fällen gilt folgendes: Wird der neue PRIVATE-Teil des Programms geladen, ehe der PUBLIC-Teil verfügbar ist, so wird der PUBLIC-Teil in den tasklokalen Speicher geladen. (Zur Vermeidung von Inkonsistenzen zwischen dem PRIVATE-Teil und dem PUBLIC-Teil führt BLS eine Validierung anhand eines Zeitstempels durch.)

Korrekturmöglichkeiten

Für dieses Modell bestehen folgende Korrekturmöglichkeiten:

- Verwendung einer REP-Datei zum Online-Laden von Korrekturen, sofern das Extern-adressbuch mindestens einen Programmabschnitt (CSECT) enthält.
- Verwendung der statischen Objektkorrektur über LMS (LMS-Anweisung MODIFY-ELEMENT).

Einsatzmöglichkeiten

Diese Art von LLM kann folgendermaßen eingesetzt werden:

- Es kann mit dem Kommando START-EXECUTABLE-PROGRAM (bzw. START-<programm>) aufgerufen werden.
- Es kann mit dem Makro BIND aufgerufen werden, jedoch nur über die Haupt-Einsprungstelle des Programms.

Einschränkungen

Ein solches LLM unterliegt folgenden Einschränkungen:

- Es kann nicht statisch in ein anderes Programm eingebunden werden.
- Es kann nicht von externen Programmen über spezifische Einsprungstellen (ENTRYs) referenziert werden; ein solches LLM kann nur als Ganzes aufgerufen werden.

8.2 Modul

Zweck dieses Modells ist die Generierung von Modulen, die in ein Programm eingefügt oder dynamisch geladen werden können (durch die dynamische Befriedigung von Externverweisen oder durch Aufruf des BIND-Makros).

Bei der Migration wird dieses Modell insbesondere dazu verwendet, die von den Compilern erzeugten Bindemodule (Objektmodule, OMs) zu ersetzen.

Haupteigenschaften

Dieses Modell hat folgende Haupteigenschaften:

- Die Performance beim Laden ist im Vergleich zu Programmen auf Grund des dynamischen Bindens nicht optimal.
- Im Vergleich zum alten OM-Format ist jedoch eine drastische Verbesserung der Lade-Performance zu verzeichnen (nachweislich Faktor 3 bis 10).
- Im Programm enthaltene BLS-Metadaten müssen auf ein Minimum reduziert werden. Es dürfen nur die Namen im Modul enthalten bzw. sichtbar sein, die von externen Modulen oder Programmen referenziert werden.

Diese Eigenschaften sind mit denen eines Bindemoduls vergleichbar. Darüberhinaus bietet das LLM-Format die folgenden Merkmale:

- Die Länge der Symbolnamen ist nicht auf 8 Zeichen beschränkt.
- Ein LLM kann (ganz oder teilweise) als Shared Code geladen werden (z.B. als DSSM-Subsystem).
- Ein LLM kann aktualisiert werden (z.B. mit der Anweisung REPLACE-MODULES), sofern die benötigten Informationen gespeichert sind.

Mögliche Rekonfigurierungen

Für dieses Modell sind folgende Rekonfigurierungen möglich:

- dynamisches Entladen eines in den tasklokalen Speicher geladenen Moduls mit dem UNBIND-Makro und Laden eines neuen Moduls mit dem BIND-Makro; diese Möglichkeit besteht auch für das OM-Format
- dynamisches Entladen des als DSSM-Subsystem geladenen Moduls mit der Anweisung STOP-SUBSYSTEM und Laden einer neuen Version mit der Anweisung START-SUBSYSTEM; diese Möglichkeit besteht auch für das OM-Format
- dynamisches Entladen des als Shared Code des Benutzers geladenen Moduls mit DSHARE und Laden einer neuen Version mit ASHARE; diese Möglichkeit besteht auch für das OM-Format.

In den beiden letzten Fällen gilt folgendes: Wird der neue PRIVATE-Teil geladen, ehe der PUBLIC-Teil verfügbar ist, so wird der PUBLIC-Teil in den tasklokalen Speicher geladen. (Zur Vermeidung von Inkonsistenzen zwischen dem PRIVATE-Teil und dem PUBLIC-Teil führt BLS eine Validierung anhand eines Zeitstempels durch.)

Korrekturmöglichkeiten

Für dieses Modell bestehen folgende Korrekturmöglichkeiten:

- statische Modifizierung des LLM über den BINDER (z.B. Anweisung REPLACE-MODULES)
- Verwendung einer REP-Datei zum Online-Laden von Korrekturen
- Verwendung der statischen Objektkorrektur über LMS (LMS-Anweisung MODIFY-ELEMENT).

Einsatzmöglichkeiten

Diese Art von LLM kann folgendermaßen eingesetzt werden:

- Diese Art von LLM kann statisch in ein anderes Programm eingebunden werden.
- Es kann von externen Programmen über spezifische Einsprungstellen, Aufruf des BIND-Makros oder die Autolink-Funktion von BLS referenziert werden.

Einschränkung

Wenn diese Art von LLM mit dem Kommando START-EXECUTABLE-PROGRAM (bzw. START-<programm>) aufgerufen werden soll, ist es in der Regel notwendig, alternative Bibliotheken (Dateikettungsname BLSLIBnn) für die Autolink-Funktion des BLS zuzuweisen.

8.3 Programmbibliothek

Zweck dieses Modells ist die Zusammenfassung eines Satzes von Programmen, die zu derselben Anwendung gehören, in einer Bibliothek. Dabei gelten für jedes dieser Programme dieselben Eigenschaften wie für ein einzelnes Programm.

Folgende Aspekte sind hier zu berücksichtigen:

- In den meisten Fällen ist eine dynamische Befriedigung von Externverweisen zwischen verschiedenen Programmen unbedingt zu vermeiden. Darum muss das Externadressbuch auf das unbedingt erforderliche Minimum reduziert werden.
- Wenn ein Teil der Programme Shared Code enthält, können diese als ein anwendungsspezifisches Subsystem geladen werden.

Mögliche Rekonfigurierungen

Für dieses Modell sind folgende Rekonfigurierungen möglich:

- Austausch der gesamten Anwendung:
 - Installation einer kompletten neuen Version der Programmbibliothek (wie bei einer Phasenbibliothek)
 - dynamisches Entladen der als DSSM-Subsystem geladenen Programme (bzw. ihrer PUBLIC-Teile) mit der Anweisung STOP-SUBSYSTEM und Laden einer neuen Version mit der Anweisung START-SUBSYSTEM
 - dynamisches Entladen der als Shared Code des Benutzers geladenen Programme (bzw. der PUBLIC-Teile der Programme) mit dem Makro DSHARE und Laden einer neuen Version mit dem Makro ASHARE.

Wird die Bibliothek ausgetauscht, ehe der gemeinsam benutzbare Teil entladen/geladen wird, so führt die Ausführung eines Programms dazu, dass sowohl der PRIVATE-Teil als auch der PUBLIC-Teil in den tasklokalen Speicher geladen wird.

- Austausch einzelner Programme der Bibliothek:

Wenn die PUBLIC-Teile der Programme als DSSM-Subsystem geladen wurden, muss das gesamte Subsystem entladen und neu geladen werden. Bei Verwendung von Shared Code des Benutzers besteht die Möglichkeit, einzelne Programme zu entladen/neu zu laden. (Allerdings findet in diesem Fall keine Validierung hinsichtlich der Tasks statt, die die gemeinsam benutzbaren Teile ausführen).

Werden die PUBLIC-Teile nicht neu geladen, führt die Ausführung eines der neuen Programme dazu, dass sowohl der PRIVATE-Teil als auch der PUBLIC-Teil in den tasklokalen Speicher geladen wird.

Korrekturmöglichkeiten

Für dieses Modell bestehen folgende Korrekturmöglichkeiten:

- Verwendung einer REP-Datei zum Online-Laden von Korrekturen, sofern das Externadressbuch mindestens einen Programmabschnitt (CSECT) enthält. Es kann eine einzige REP-Datei für die gesamte Bibliothek verwendet werden. In diesem Fall muss eine NOREF-Datei verwendet werden, um zu vermeiden, dass die Meldung 'Rep name error' ausgegeben wird.
- Verwendung der statischen Objektkorrektur über LMS (LMS-Anweisung MODIFY-ELEMENT).

Einsatzmöglichkeiten

Diese Art von LLM kann folgendermaßen eingesetzt werden:

- Es kann mit dem Kommando START-EXECUTABLE-PROGRAM (bzw. START-<programm>) aufgerufen werden.
- Es kann mit dem Makro BIND aufgerufen werden, jedoch nur über die Haupt-Einsprungstelle des Programms.

Einschränkungen

Ein solches LLM unterliegt folgenden Einschränkungen:

- Es kann nicht statisch in ein anderes Programm eingebunden werden.
- Es kann nicht von externen Programmen über spezifische Einsprungstellen referenziert werden; ein solches LLM kann nur als Ganzes aufgerufen werden.

8.4 Modulbibliothek

Zweck dieses Modells ist die Zusammenfassung eines Satzes von Modulen, die zu derselben Anwendung gehören, in einer Bibliothek. Ein typisches Beispiel sind Laufzeitbibliotheken mit Sprachmodulen. Dabei gelten für jedes dieser Module dieselben Eigenschaften wie für ein einzelnes Modul.

Wenn ein Teil der Module Shared Code enthält, können die Module als ein anwendungsspezifisches Subsystem geladen werden.

Mögliche Rekonfigurierungen

Für dieses Modell sind folgende Rekonfigurierungen möglich:

- Austausch der gesamten Anwendung:
 - Installation einer kompletten neuen Version der Modulbibliothek (wie bei einer Phasenbibliothek)
 - dynamisches Entladen der als DSSM-Subsystem geladenen Module (bzw. der PUBLIC-Teile der Module) mit der Anweisung STOP-SUBSYSTEM und Laden einer neuen Version mit der Anweisung START-SUBSYSTEM
 - dynamisches Entladen der als Shared Code des Benutzers geladenen Module (bzw. der PUBLIC-Teile der Module) mit dem Makro DSHARE und Laden einer neuen Version mit dem Makro ASHARE.

Wird die Bibliothek ausgetauscht, ehe der gemeinsam benutzbare Teil entladen/geladen wird, so führt das Laden des PRIVATE-Teils eines Moduls dazu, dass sowohl der PRIVATE-Teil als auch der PUBLIC-Teil in den tasklokalen Speicher geladen wird.

- Austausch einzelner Module der Bibliothek:

Wenn die PUBLIC-Teile der Module als DSSM-Subsystem geladen wurden, muss das gesamte Subsystem entladen und neu geladen werden. Bei Verwendung von Shared Code des Benutzers besteht die Möglichkeit, einzelne Module zu entladen/neu zu laden. (Allerdings findet in diesem Fall keine Validierung hinsichtlich der Tasks statt, die die gemeinsam benutzbaren Teile ausführen.) Werden die PUBLIC-Teile nicht neu geladen, führt das Laden eines der neuen Module dazu, dass sowohl der PRIVATE-Teil als auch der PUBLIC-Teil in den tasklokalen Speicher geladen wird.

Korrekturmöglichkeiten

Für dieses Modell bestehen folgende Korrekturmöglichkeiten:

- statische Modifizierung des LLM über den BINDER (z.B. Anweisung REPLACE-MODULES)
- Verwendung einer REP-Datei zum Online-Laden von Korrekturen
- Verwendung der statischen Objektkorrektur über LMS (LMS-Anweisung MODIFY-ELEMENT).

Einsatzmöglichkeiten

Diese Art von LLM kann folgendermaßen eingesetzt werden:

- Diese Art von LLM kann statisch in ein anderes Programm eingebunden werden.
- Sie kann von externen Programmen über spezifische Einsprungstellen, BIND-Aufruf oder die Autolink-Funktion von BLS referenziert werden.

Einschränkung

Wenn diese Art von LLM mit dem Kommando START-EXECUTABLE-PROGRAM (bzw. START-<programm>) aufgerufen werden soll, ist es in der Regel notwendig, alternative Bibliotheken (Dateikettungsname BLSLIBnn) für die Autolink-Funktion des BLS zuzuweisen.

8.5 Generierung der unterschiedlichen Programmtypen

In diesem Abschnitt ist beschrieben, auf welche Weise LLMs erzeugt werden müssen, um den einzelnen Nutzungsmodellen zu entsprechen.

8.5.1 Generierung als Programm

Ein Programm ist ein LLM, das selbstständig ablauffähig ist. Es kann andere Objekte referenzieren (über die dynamische Befriedigung von Externverweisen oder Aufrufe des BIND-Makros). Es kann jedoch weder von externen Objekten referenziert werden, noch kann es in andere LLMs statisch eingebunden werden. Es kann auch als Shared Code geladen werden.

8.5.1.1 Nicht gemeinsam benutzbares Programm

Das LLM kann als Einzel-Slice strukturiert sein oder in mehrere Slices aufgeteilt werden; dabei werden die Slices nach den Attributen READ-ONLY bzw. READ-WRITE und/oder RMODE=ANY bzw. RMODE=24 gebildet. Beim Speichern des LLM muss weder das Externadressbuch noch die Logische Strukturinformation mitgespeichert werden. Die Relativierungsinformation kann bei Bedarf mitgespeichert werden.

Bindeprozedur

```
//START-LLM-CREATION INTERNAL-NAME=<interner-name>, -  
//      INTERNAL-VERSION=<version>  
//INCLUDE-MODULES ... _____ (1)  
//INCLUDE-MODULES ...  
//RESOLVE-BY-AUTOLINK ... _____ (2)  
//SAVE-LLM LIBRARY=<bibliothek>,ELEMENT=<element>, -  
//      SYMBOL-DICTIONARY=*NO, LOGICAL-STRUCTURE=*NONE,  
//      RELOCATION-DATA=*YES/*NO _____ (3)
```

- (1) Alle Module des Programms (OMs oder LLMs) werden explizit eingefügt.
- (2) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (3) Das LLM wird ohne Externadressbuch und ohne Logische Strukturinformation gespeichert. Die Relativierungsinformation kann bei Bedarf mitgespeichert werden.

Programmausführung

Das Programm kann mit dem folgenden Kommando ausgeführt werden:

```
/START-EXECUTABLE-PROGRAM -
/      FROM-FILE=*LIBRARY-ELEMENT( -
/      LIBRARY=<bibliothek>,ELEMENT-OR-SYMBOL=<element>), -
/      DBL-PARAMETERS=(LOADING=(PROGRAM-MODE=*ANY,REP-FILE=<rep-dateiname>))
```

Es ist auch möglich (und empfehlenswert), ein spezifisches SDF-Kommando START-<programm> zu erstellen und z.B. über eine BS2000- Prozedur zu implementieren. Auf diese Weise können die Parameter, die zum Starten der Programmausführung benötigt werden (z.B. der Name der REP-Datei) 'verborgen' werden. Außerdem erreicht man damit Unabhängigkeit von den Dateinamen, die zum Laden des Programms benötigt werden.

8.5.1.2 Teilweise gemeinsam benutzbares Programm

Ein teilweise gemeinsam benutzbares Programm enthält sowohl gemeinsam benutzbaren als auch nicht gemeinsam benutzbaren Code. Ein teilweise gemeinsam benutzbares LLM muss in mehrere, nach den Attributen PUBLIC/PRIVATE gebildete Slices aufgeteilt sein. Das CSECT-Attribut generiert der Compiler automatisch (s.u. Übersetzung). Der BINDER fasst dann automatisch alle CSECTs, die das gleiche Attribut haben, in einer Slice zusammen (s.u. Bindeprozedur). Um die PUBLIC-Slice als DSSM-Subsystem zu laden, muss ein DSSM-Katalog generiert werden (s.u. DSSM-Deklaration). Das Modell basiert auf LLM-Format 2.

Übersetzung

Um CSECTs mit den Attributen PUBLIC bzw. PRIVATE zu erzeugen, müssen die Quellprogramme mit der folgenden Option übersetzt werden (Beispiel eines C Compilers):

```
COMPILER-ACTION=*MODULE-GENERATION(SHAREABLE-CODE=*YES,
MODULE-FORMAT=*OM/*LLM)
```

Wenn Symbolnamen mit mehr als 8 Zeichen verwendet werden, ist MODULE-FORMAT=*LLM anzugeben.

Bindeprozedur

```

//START-LLM-CREATION INTERNAL-NAME=<name>, -
//  INTERNAL-VERSION=<version>, -
//  SLICE-DEFINITION=*BY-ATTRIBUTES(PUBLIC=*YES -
//                                (SUB-ENTRIES=<symbolname>)) _____ (1)
//INCLUDE-MODULES ... _____ (2)
//INCLUDE-MODULES ...
//RESOLVE-BY-AUTOLINK ... _____ (3)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=*ALL,VISIBLE=*NO _____ (4)
[/RENAME-SYMBOL SYMBOL-NAME=<public-definition>, -
//  NEW-NAME=<symbolname>,SYMBOL-OCCURENCE=*PARAMETERS -
//  (FIRST-OCCURENCE=1,OCCURENCE-NUMBER=*ALL)] _____ (5)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=<symbolname>, -
//  VISIBLE=*YES _____ (6)
//SAVE-LLM LIBRARY=<bibliothek>,ELEMENT=<element>

```

- (1) Die Slice-Definition kann mit der Anweisung START-LLM-CREATION (oder mit MODIFY-LLM-ATTRIBUTES) erfolgen. Für SUB-ENTRIES muss der Benutzer eine Programmdefinition (CSECT oder ENTRY) in der PUBLIC-Slice angeben, die in der PRIVATE-Slice verwendet wird. Der hier angegebene Name muss das Subsystem eindeutig identifizieren. Ist dies nicht möglich, muss der Symbolname geändert werden (s.u. 5).
- (2) Alle Module des Programms (OMs oder LLMs) werden explizit eingefügt.
- (3) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (4) Alle Symbole werden maskiert.
- (5) Optional: Kann keine der Programmdefinitionen in dem LLM als eindeutige Angabe für SUB-ENTRIES verwendet werden, so muss der Benutzer eine PUBLIC-Definition, die in der PRIVATE-Slice verwendet wird, umbenennen, um einen eindeutigen Namen zu erzeugen.
- (6) Das benötigte Symbol wird sichtbar gemacht.

DSSM-Deklaration

Um die PUBLIC-Slice des LLM zu laden, müssen die folgenden Definitionen im Subsystem-Katalog vorgenommen werden (mit dem Produkt SSCM):

```
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=<subsystem-name>,-
// LIBRARY=<bibliothekensname>,LINK-ENTRY=<symbolname>,-
// SUBSYSTEM-ENTRIES=(<symbolname>(CONNECTION-SCOPE=*PROGRAM)), -
// MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*LOW/*HIGH)
```

Der Subsystem-Zugriff muss in Abhängigkeit vom Adressierungsmodus der Programme definiert werden (Empfehlung: HIGH).

Vorladen der PUBLIC-Slice

Das Subsystem kann mit dem Kommando START-SUBSYSTEM geladen werden. Durch Angabe des Parameters CREATION-TIME=*AFTER-SYSTEM-READY in der SSCM-Anweisung SET-SUBSYSTEM-ATTRIBUTES kann angegeben werden, dass das Subsystem automatisch geladen werden soll. Wird das Subsystem nicht geladen, so wird die PUBLIC-Slice bei Programmausführung in den Benutzeradressraum geladen.

Programmausführung

Das Programm kann mit dem folgenden Kommando ausgeführt werden:

```
/START-EXECUTABLE-PROGRAM -
/ FROM-FILE=*LIBRARY-ELEMENT( -
/ LIBRARY=<bibliothek>,ELEMENT-OR-SYMBOL=<element>), -
/ DBL-PARAMETERS=(LOADING=(PROGRAM-MODE=*ANY,REP-FILE=<rep-dateiname>))
```

Es ist auch möglich und empfehlenswert, ein spezifisches SDF-Kommando START-<programm> zu erstellen und z.B. über eine Prozedur zu implementieren. Auf diese Weise können die Parameter, die zum Starten der Programmausführung benötigt werden (z.B. der Name der REP-Datei) 'verborgen' werden. Außerdem wird damit Unabhängigkeit von den Dateinamen erreicht, die zum Laden des Programms benötigt werden. Für ein teilweise gemeinsam benutzbares Programm lädt BLS dann die PRIVATE-Slice in den Benutzeradressraum und verknüpft sie mit der dazugehörigen PUBLIC-Slice (über den Symbolnamen). Dabei überprüft BLS zur Vermeidung von Inkonsistenzen anhand eines Zeitstempels, ob die beiden Slices tatsächlich zum selben LLM gehören. Ist das nicht der Fall, so lädt BLS die PUBLIC-Slice in den Benutzeradressraum.

8.5.1.3 Vollständig gemeinsam benutzbares Programm

Ein vollständig gemeinsam benutzbares Programm enthält ausschließlich gemeinsam benutzbaren Code. Ein vollständig gemeinsam benutzbares LLM kann der Benutzer als Einzel-Slice-LLM erzeugen. Das Externadressbuch darf in diesem Fall nur ein Symbol enthalten, das als „Connectable Entry“ sowie als Einsprungstelle für das Laden des Programms verwendet werden muss. Um das LLM als DSSM-Subsystem zu laden, muss ein DSSM-Katalog generiert werden (s.u. DSSM-Deklaration).

Bindeprozedur

```
//START-LLM-CREATION INTERNAL-NAME=<interner-name>, -
//                                INTERNAL-VERSION=<version>
//INCLUDE-MODULES ... _____ (1)
//INCLUDE-MODULES ...
//RESOLVE-BY-AUTOLINK ... _____ (2)
//MERGE-MODULES NAME=<interner-name>,ENTRY-LIST=<symbol> _____ (3)
//SAVE-LLM LIBRARY=<bibliothek>,ELEMENT=<element>, -
// LOGICAL-STRUCTURE=*WHOLE-LLM, SYMBOL-DICTIONARY=*YES, -
// RELOCATION-DATA=*YES
```

- (1) Alle Module des Programms (OMs oder LLMs) werden explizit eingefügt.
- (2) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (3) Alle Symbole außer <symbol> werden aus dem Externadressbuch entfernt. Nach der Anweisung MERGE-MODULES enthält das LLM nur noch eine einzige CSECT. Das bedeutet, dass die Eigenschaften dieser CSECT für das ganze Programm gelten (z.B. READ-ONLY, RMODE, AMODE,...). Soll dies vermieden werden, besteht die Möglichkeit, verschiedene MERGE-MODULES-Anweisungen abzusetzen (diese Anweisung kann auf Sub-LLM- oder OM-Ebene verwendet werden).

DSSM-Deklaration

Zum Laden des LLM müssen die folgenden Definitionen im Subsystem-Katalog vorgenommen werden (mit dem Produkt SSCM):

```
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=<subsystem-name>, -
// LIBRARY= <bibliothekensname>, LINK-ENTRY=<symbol>, -
// SUBSYSTEM-ENTRIES=(<symbol>(CONNECTION-SCOPE=*PROGRAM)), -
// MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*LOW/*HIGH)
```

Der Subsystem-Zugriff muss in Abhängigkeit vom Adressierungsmodus der Programme definiert werden (Empfehlung: HIGH).

Vorladen des Programms

Das Subsystem kann mit dem Kommando START-SUBSYSTEM geladen werden. Durch Angabe des Parameters CREATION-TIME=*AFTER-SYSTEM-READY in der SSCM-Anweisung SET-SUBSYSTEM-ATTRIBUTES kann angegeben werden, dass das Subsystem automatisch geladen werden soll. Wird das Subsystem nicht geladen, so wird die PUBLIC-Slice bei Programmausführung in den Benutzeradressraum geladen.

Programmausführung

Das Programm kann mit dem folgenden Kommando ausgeführt werden:

```
/START-EXECUTABLE-PROGRAM -  
/      FROM-FILE=*LIBRARY-ELEMENT( -  
/          LIBRARY=<bibliothek>,ELEMENT-OR-SYMBOL=<element>), -  
/      DBL-PARAMETERS=(LOADING=(PROGRAM-MODE=*ANY,REP-FILE=<rep-dateiname>))
```

Die Angabe von <bibliothek> ist nur sinnvoll, wenn das Programm nicht als DSSM-Subsystem geladen wird.

Es ist auch möglich und empfehlenswert, ein spezifisches SDF-Kommando START-<programm> zu erstellen und z.B. über eine Prozedur zu implementieren. Auf diese Weise können die Parameter, die zum Starten der Programmausführung benötigt werden (z.B. der Name der REP-Datei) 'verborgen' werden. Außerdem wird damit Unabhängigkeit von den Dateinamen erreicht, die zum Laden des Programms benötigt werden.

8.5.2 Generierung als Modul

Ein Modul ist ein LLM, das in ein Programm eingefügt und dynamisch geladen werden kann (durch die dynamische Befriedigung von Extern- verweisen oder durch Aufruf des BIND-Makros). Ziel der in diesem Abschnitt beschriebenen Modelle ist es, alle Symbole des Externadressbuchs zu verbergen bzw. zu unterdrücken, die für den Aufruf des Moduls nicht benötigt werden. Damit wird eine Performance-Verbesserung beim Laden erreicht (ZE und E/A).

8.5.2.1 Nicht gemeinsam benutzbares Modul

Ein nicht gemeinsam benutzbares Modul wird in den Benutzeradressraum und nicht in den Shared Code geladen.

Es besteht die Möglichkeit, mit der Mischfunktion alle nicht benötigten Definitionen im Externadressbuch zu unterdrücken und nur die benötigten Definitionen zu erhalten. Dies wird in der Anweisung MERGE-MODULES angegeben. Die Anzahl der Symbole, die bei Verwendung dieser Funktion im Externadressbuch verbleiben können, ist jedoch auf 40 begrenzt. Diese Vorgehensweise ist in Variante 1 der Bindeprozedur dargestellt.

Soll das LLM mehr als 40 externe Namen enthalten, so kann die Mischfunktion nicht verwendet werden. Zur Vermeidung von Namenskonflikten und unerwarteter Befriedigung von Externverweisen muss der Benutzer in diesem Fall alle Symbole, die an der Modulschnittstelle nicht benötigt werden, maskieren. Auf diese Weise sind die ausschließlich intern verwendeten Symbole nicht sichtbar (Variante 2 der Bindeprozedur).

Bindeprozedur (Variante 1 für maximal 40 externe Namen)

```
//START-LLM-CREATION INTERNAL-NAME=<interner-name>, -
//                               INTERNAL-VERSION=<version>
//INCLUDE-MODULES ... _____ (1)
//INCLUDE-MODULES ...
//RESOLVE-BY-AUTOLINK ... _____ (2)
//MERGE-MODULES NAME=<interner-name>,ENTRY-LIST=<liste> _____ (3)
//SAVE-LLM LIBRARY=...,ELEMENT=...,
```

- (1) Alle Module des Programms (OMs oder LLMs) werden explizit eingefügt.
- (2) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (3) Es werden alle Symbole im Externadressbuch unterdrückt, die an der Modulschnittstelle nicht benötigt werden; mit <liste> werden die Symbole angegeben, die wahrscheinlich von externen Einheiten referenziert werden und daher im Externadressbuch verbleiben sollen.

Nach der Anweisung MERGE-MODULES enthält das LLM nur noch eine einzige CSECT. Das bedeutet, dass die Eigenschaften dieser CSECT für das ganze Programm gelten (z.B. READ-ONLY, RMODE, AMODE,...). Soll dies vermieden werden, besteht die Möglichkeit, verschiedene MERGE-MODULES-Anweisungen abzusetzen (diese Anweisung kann auf Sub-LLM- oder OM-Ebene verwendet werden).

Bindeprozedur (Variante 2 für Module mit mehr als 40 externen Namen)

```
//START-LLM-CREATION INTERNAL-NAME=<interner-name>, -
//                               INTERNAL-VERSION=<version>
//INCLUDE-MODULES ... _____ (1)
//INCLUDE-MODULES ...
//RESOLVE-BY-AUTOLINK ... _____ (2)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=*ALL,VISIBLE=*NO _____ (3)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=<liste 1>,VISIBLE=*YES _____ (4)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=<liste 2>,VISIBLE=*YES
//SAVE-LLM LIBRARY=...,ELEMENT=...,
```

- (1) Alle Module des Programms (OMs oder LLMs) werden explizit eingefügt.
- (2) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (3) Alle Symbole werden maskiert.
- (4) Die benötigten Symbole werden sichtbar gemacht; mit <liste 1>/<liste 2> werden die Symbole angegeben, die wahrscheinlich von externen Einheiten referenziert werden und daher sichtbar bleiben sollen. Falls erforderlich, können mehr als 2 Anweisungen angegeben werden.

8.5.2.2 Teilweise gemeinsam benutzbares Modul

Ein teilweise gemeinsam benutzbares Modul enthält sowohl gemeinsam benutzbaren als auch nicht gemeinsam benutzbaren Code. Der gemeinsam benutzbare Code kann in den Shared Code geladen werden. Er kann auch über die dynamische Befriedigung von Externverweisen dynamisch geladen werden. Ein teilweise gemeinsam benutzbares LLM muss in mehrere, nach den Attributen PUBLIC/PRIVATE gebildete Slices aufgeteilt sein. Das CSECT-Attribut generiert der Compiler automatisch (s.u. Übersetzung). Der BINDER fasst dann automatisch alle CSECTs, die das gleiche Attribut haben, in einer Slice zusammen (s.u. Bindeprozedur). Um die PUBLIC-Slice als DSSM-Subsystem zu laden, muss ein DSSM-Katalog generiert werden (s.u. DSSM-Deklaration). Das Modell basiert auf LLM-Format 2.

Übersetzung

Um CSECTs mit den Attributen PUBLIC bzw. PRIVATE zu erzeugen, müssen die Quellprogramme mit der folgenden Option übersetzt werden (Beispiel eines C-Compilers):

```
COMPILER-ACTION=*MODULE-GENERATION(SHAREABLE-CODE=*YES,  
MODULE-FORMAT=*OM/*LLM)
```

Wenn Symbolnamen mit mehr als 8 Zeichen verwendet werden, ist MODULE-FORMAT=*LLM anzugeben.

Bindeprozedur

```
//START-LLM-CREATION INTERNAL-NAME=<name>,
// INTERNAL-VERSION=<version>,SLICE-DEFINITION=*BY-ATTRIBUTES -
// (PUBLIC=*YES(SUB-ENTRIES=<symbolname>)) _____ (1)
//INCLUDE-MODULES ... _____ (2)
//INCLUDE-MODULES ...
//RESOLVE-BY-AUTOLINK ... _____ (3)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=*ALL,VISIBLE=*NO _____ (4)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=<liste>,VISIBLE=*YES _____ (5)
[/RENAME-SYMBOL SYMBOL-NAME=<public-definition>, -
// NEW-NAME=<symbolname>,SYMBOL-OCCURENCE=*PARAMETERS -
// (FIRST-OCCURENCE=1,OCCURENCE-NUMBER=*ALL)] _____ (6)
//SAVE-LLM LIBRARY=...,ELEMENT=...,
```

- (1) Die Slice-Definition kann mit der Anweisung START-LLM-CREATION (oder mit MODIFY-LLM-ATTRIBUTES) erfolgen. Für SUB-ENTRIES muss der Benutzer eine Programmdefinition (CSECT oder ENTRY) in der PUBLIC-Slice angeben, die in der PRIVATE-Slice verwendet wird. Der hier angegebene Name muss das Subsystem eindeutig identifizieren. Ist dies nicht möglich, muss der Symbolname geändert werden (siehe (6)).
- (2) Alle Module (OMs oder LLMs) werden explizit eingefügt.
- (3) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (4) Alle Symbole werden maskiert.
- (5) Die benötigten Symbole werden sichtbar gemacht; mit <liste> werden die Symbole angegeben, die wahrscheinlich von externen Einheiten referenziert werden und daher sichtbar bleiben sollen. Der für SUB-ENTRIES angegebene Symbolname muss sichtbar bleiben.
- (6) Optional: Kann keine der Programmdefinitionen in dem LLM als eindeutige Angabe für SUB-ENTRIES verwendet werden, so muss der Benutzer eine (sichtbare) PUBLIC-Definition, die in der PRIVATE-Slice verwendet wird, umbenennen, um einen eindeutigen Namen zu erzeugen.

DSSM-Deklaration

Um die PUBLIC-Slice des LLM zu laden, müssen die folgenden Definitionen im Subsystem-Katalog vorgenommen werden (mit dem Produkt SSCM):

```
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=<subsystem-name>, -  
//  LIBRARY=<bibliotheksnamen>,LINK-ENTRY=<symbolnamen>, -  
//    SUBSYSTEM-ENTRIES=(<symbolnamen> -  
//                          (CONNECTION-SCOPE=*PROGRAM)), -  
//  MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*LOW/*HIGH)
```

Der Subsystem-Zugriff muss in Abhängigkeit vom Adressierungsmodus der Programme definiert werden (Empfehlung: HIGH). Kann die PUBLIC-Slice von externen Einheiten über andere Namen als <symbolnamen> referenziert werden, müssen diese Namen ebenfalls mit dem Parameter SUBSYSTEM-ENTRIES definiert werden:

```
SUBSYSTEM-ENTRIES=(<symbolnamen>(CONNECTION-SCOPE=*PROGRAM),  
...<symbol 1>(CONNECTION-SCOPE=*PROGRAM),  
...<symbol n>(CONNECTION-SCOPE=*PROGRAM))
```

Vorladen der PUBLIC-Slice

Das Subsystem kann mit dem Kommando START-SUBSYSTEM geladen werden. Durch Angabe des Parameters CREATION-TIME=*AFTER-SYSTEM-READY in der SSCM-Anweisung SET-SUBSYSTEM-ATTRIBUTES kann angegeben werden, dass das Subsystem automatisch geladen werden soll. Wird das Subsystem nicht geladen, so wird die PUBLIC-Slice bei Programmausführung in den Benutzeradressraum geladen.

8.5.2.3 Vollständig gemeinsam benutzbares Modul

Ein vollständig gemeinsam benutzbares Modul enthält ausschließlich gemeinsam benutzbaren Code und kann daher vollständig in den Shared Code geladen werden. Ein solches LLM kann auch über die dynamische Befriedigung von Externverweisen dynamisch geladen werden. Um ein vollständig gemeinsam benutzbares LLM zu erzeugen, kann der Benutzer die Modelle für nicht gemeinsam benutzbare Module verwenden. (Hier wird die Variante mit mehr als 40 externen Namen beschrieben.) Um das LLM als DSSM-Subsystem zu laden, muss ein DSSM-Katalog generiert werden (s.u. DSSM-Deklaration).

Bindeprozedur

```
//START-LLM-CREATION INTERNAL-NAME=<interner-name>, -
//                      INTERNAL-VERSION=<version>
//INCLUDE-MODULES ... _____ (1)
//INCLUDE-MODULES ...
//RESOLVE-BY-AUTOLINK ... _____ (2)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=*ALL,VISIBLE=*NO _____ (3)
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=<liste>,VISIBLE=*YES _____ (4)
//SAVE-LLM LIBRARY=...,ELEMENT=...,
```

- (1) Alle Module des Programms (OMs oder LLMs) werden explizit eingefügt.
- (2) Die Autolink-Funktion wird für die angegebenen Bibliotheken (z.B. Laufzeitbibliotheken) aufgerufen.
- (3) Alle Symbole werden maskiert.
- (4) Die benötigten Symbole werden sichtbar gemacht; mit <liste> werden die Symbole angegeben, die wahrscheinlich von externen Einheiten referenziert werden und daher sichtbar bleiben sollen. <liste>=(<symbol 1>,...,<symbol n>).

DSSM-Deklaration

Um das LLM zu laden, müssen die folgenden Definitionen im Subsystem-Katalog vorgenommen werden (mit dem Produkt SSCM):

```
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=<subsystem-name>, -
// LIBRARY=<bibliothekensname>,LINK-ENTRY=<symbol 1>, -
// SUBSYSTEM-ENTRIES=(<symbol 1>(CONNECTION-SCOPE=*PROGRAM)), -
//                      ...<symbol n>(CONNECTION-SCOPE=*PROGRAM), -
// MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*LOW/*HIGH)
```

Der Subsystem-Zugriff muss in Abhängigkeit vom Adressierungsmodus der Programme definiert werden (Empfehlung: HIGH).

Vorladen des Moduls

Das Subsystem kann mit dem Kommando START-SUBSYSTEM geladen werden. Durch Angabe des Parameters CREATION-TIME=*AFTER-SYSTEM-READY in der SSCM-Anweisung SET-SUBSYSTEM-ATTRIBUTES kann angegeben werden, dass das Subsystem automatisch geladen werden soll. Wird das Subsystem nicht geladen, so wird die PUBLIC-Slice bei Programmausführung in den Benutzeradressraum geladen.

8.5.3 Generierung als Programmbibliothek

Eine Programmbibliothek besteht aus einem Satz von Programmen, die mit START-EXECUTABLE-PROGRAM geladen werden. Zweck einer Programmbibliothek ist die Definition eines aus mehreren Programmen bestehenden Produkts. Eine Programmbibliothek kann vollständig gemeinsam benutzbare Programme, teilweise gemeinsam benutzbare Programme und nicht gemeinsam benutzbare Programme enthalten. Wenn die Bibliothek gemeinsam benutzbare Programme enthält, sollten diese gemeinsam als ein DSSM-Subsystem geladen werden. Da sie mit START-EXECUTABLE-PROGRAM aufgerufen werden, müssen sie außerdem als selbstständige Programme generiert werden. Die in diesem Kapitel beschriebenen Modelle basieren auf LLM-Format 2.

Übersetzung und Binden

Die Übersetzung und das Binden der einzelnen Programme ist jeweils so durchzuführen, wie es in den Abschnitten „Nicht gemeinsam benutzbares Programm“ auf Seite 346, „Teilweise gemeinsam benutzbares Programm“ auf Seite 347 und „Vollständig gemeinsam benutzbares Programm“ auf Seite 350 beschrieben ist.

DSSM-Deklaration

Zum Laden der Programmbibliothek (PUBLIC-Slices aller teilweise gemeinsam benutzbaren LLMs und alle vollständig gemeinsam benutzbaren LLMs) müssen die folgenden Definitionen im Subsystem-Katalog vorgenommen werden (mit dem Produkt SSCM):

```
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=<symbolname>, -
//  LIBRARY=<programmname>,LINK-ENTRY=<symbolname 1>,-
//  SUBSYSTEM-ENTRIES=(<symbolname 1> -
//                      (CONNECTION-SCOPE=*PROGRAM), -
//                      ...<symbolname n>(CONNECTION-SCOPE=*PROGRAM), -
//  MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*LOW/*HIGH)
```

<symbolname 1> ... <symbolname n> sind hierbei die Namen, die beim Binden der einzelnen Programme im folgenden Operanden angegeben wurden:

```
SLICE-DEFINITION=*BY-ATTRIBUTES(PUBLIC=*YES(SUB-ENTRIES=<symbolname>)
```

Der Subsystem-Zugriff muss in Abhängigkeit vom Adressierungsmodus der Programme definiert werden (Empfehlung: HIGH).

Vorladen der PUBLIC-Slices

Das Subsystem kann mit dem Kommando START-SUBSYSTEM geladen werden. Durch Angabe des Parameters CREATION-TIME=*AFTER-SYSTEM-READY in der SSCM-Anweisung SET-SUBSYSTEM-ATTRIBUTES kann angegeben werden, dass das Subsystem automatisch geladen werden soll. Wird das Subsystem nicht geladen, so werden alle gemeinsam benutzbaren Teile in den Benutzeradressraum geladen.

Ausführung

Zur Ausführung eines bestimmten Programms der Programmbibliothek kann das folgende Kommando verwendet werden:

```
/START-EXECUTABLE-PROGRAM -  
/ FROM-FILE=*LIBRARY-ELEMENT( -  
/ LIBRARY=<bibliothek>,ELEMENT-OR-SYMBOL=<element>), -  
/ DBL-PARAMETERS=(LOADING=(PROGRAM-MODE=*ANY,REP-FILE=<rep-dateiname>))
```

Es ist auch möglich und empfehlenswert, ein spezifisches SDF- Kommando START-<programm> zu erstellen und z.B. über eine BS2000- Prozedur zu implementieren. Auf diese Weise können die Parameter, die zum Starten der Programmausführung benötigt werden (z.B. der Name der REP-Datei) 'verborgen' werden. Außerdem wird damit Unabhängigkeit von den Dateinamen erreicht, die zum Laden des Programms benötigt werden. Für ein teilweise gemeinsam benutzbares Programm lädt BLS dann die PRIVATE-Slice in den Benutzeradressraum und verknüpft sie mit der dazugehörigen PUBLIC-Slice (über den Symbolnamen). Dabei überprüft BLS zur Vermeidung von Inkonsistenzen anhand eines Zeitstempels, ob die beiden Slices tatsächlich zum selben LLM gehören. Ist das nicht der Fall, so lädt BLS die PUBLIC-Slice in den Benutzeradressraum.

8.5.4 Generierung als Modulbibliothek

Eine Modulbibliothek besteht aus einem Satz von Modulen, die statisch in eine Anwendung eingebunden oder dynamisch geladen werden können. Zweck einer Modulbibliothek ist die Definition eines aus mehreren Modulen bestehenden Produkts. Eine Modulbibliothek kann vollständig gemeinsam benutzbare Module, teilweise gemeinsam benutzbare Module und nicht gemeinsam benutzbare Module enthalten. Wenn die Bibliothek gemeinsam benutzbare Module enthält, sollten diese gemeinsam als ein DSSM-Subsystem geladen werden. Die in diesem Kapitel beschriebenen Modelle basieren auf LLM-Format 2.

Übersetzung und Binden

Die Übersetzung und das Binden der einzelnen Module ist jeweils so durchzuführen, wie es in den Abschnitten „Nicht gemeinsam benutzbares Modul“ auf Seite 352, „Teilweise gemeinsam benutzbares Modul“ auf Seite 354 und „Vollständig gemeinsam benutzbares Modul“ auf Seite 356 beschrieben ist.

DSSM-Deklaration

Zum Laden der Modulbibliothek (PUBLIC-Slices aller teilweise gemeinsam benutzbaren LLMs und alle vollständig gemeinsam benutzbaren LLMs) müssen die folgenden Definitionen im Subsystem-Katalog vorgenommen werden (mit dem Produkt SSCM):

```
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=<subsystem-name>, -  
//  LIBRARY=<programmname>,LINK-ENTRY=<symbol 1>, -  
//  SUBSYSTEM-ENTRIES=(<symbol 1>(CONNECTION-SCOPE=*PROGRAM), -  
//      ...<symbol k>(CONNECTION-SCOPE=*PROGRAM), -  
//  MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*LOW/*HIGH)
```

<symbolname 1> ... <symbolname n> sind hierbei die Namen, die beim Binden der einzelnen Module im folgenden Operanden angegeben wurden:

```
SLICE-DEFINITION=*BY-ATTRIBUTES(PUBLIC=*YES(SUB-ENTRIES=<symbolname>)
```

Der Subsystem-Zugriff muss in Abhängigkeit vom Adressierungsmodus der Programme definiert werden (Empfehlung: HIGH).

Vorladen der PUBLIC-Slices

Das Subsystem kann mit dem Kommando START-SUBSYSTEM geladen werden. Durch Angabe des Parameters CREATION-TIME=*AFTER-SYSTEM-READY in der SSCM-Anweisung SET-SUBSYSTEM-ATTRIBUTES kann angegeben werden, dass das Subsystem automatisch geladen werden soll. Wird das Subsystem nicht geladen, so werden alle gemeinsam benutzbaren Teile in den Benutzeradressraum geladen.

8.6 Weitere Hinweise zu den Nutzungsmodellen

8.6.1 LLM-Format 1

Alle in diesem Kapitel genannten Modelle beziehen sich auf LLM-Format 2. LLM-Format 1 unterscheidet sich von LLM-Format 2 vor allem in den folgenden Punkten:

Bei LLM-Format 1 ruft BLS DSSM für jedes Symbol in der PUBLIC-Slice auf, das von der PRIVATE-Slice referenziert wird. Deshalb müssen alle Namen im DSSM-Katalog als SUBSYSTEM-Entries definiert werden. Dies bewirkt eine erheblich eingeschränkte Verwendung solcher Objekte. (Die einem gemeinsam benutzbaren und in verschiedenen LLMs verwendeten Laufzeitmodul entsprechenden Symbole würden in allen betreffenden Subsystemen definiert.)

Bei LLM-Format 2 kann der Benutzer während des BINDER-Laufs ein Symbol definieren (Operand SUBSYSTEM-ENTRIES in START-LLM-CREATION), das für die Verknüpfung der beiden Slices zum Ladezeitpunkt verwendet werden muss. Dieses Symbol muss sich in der PUBLIC-Slice befinden und von der PRIVATE-Slice referenziert werden. In diesem Fall ruft BLS DSSM nur für dieses Symbol auf und befriedigt alle anderen Externverweise automatisch ohne Aufruf von DSSM. Der Benutzer muss daher nur ein Symbol im DSSM-Katalog deklarieren. Seit BINDER V1.3A ist es möglich, LLM-Format 1 in dieser Hinsicht mit derselben Eigenschaft wie LLM-Format 2 zu erzeugen.

8.6.2 Vorladen von PUBLIC-Slices über die ASHARE-Schnittstelle

In diesem Kapitel werden nur die DSSM-Funktionen für das Vorladen des Shared Code beschrieben.

Eine weitere Möglichkeit zum Laden von Shared Code wird als Shared Code des Benutzers bezeichnet. In diesem Fall kann der Shared Code über die BLS-Programmschnittstelle ASHARE in einen Common Memory Pool im Klasse-6-Speicher geladen werden. Ein DSSM-Katalog ist für diese Option nicht erforderlich.

Memory Pools, die von der ASHARE-Schnittstelle zum Laden von Shared Code verwendet werden, werden vom Anwenderprogramm angefordert. Daher müssen diese Memory Pools auch vom Anwenderprogramm selbst verwaltet werden (im Gegensatz zu Subsystemen, für die DSSM diese Aufgabe übernimmt).

Außerdem ist zu beachten, dass ein Memory-Pool vom System aufgelöst wird, wenn keine Task mehr damit verknüpft ist, d.h. nachdem die letzte Task die Verbindung entweder explizit mit DISMP oder implizit bei Programmbeendigung gelöst hat.

Daher sind folgende Maßnahmen erforderlich:

1. Damit der Memory Pool während des gesamten Laufes einer Anwendung verfügbar ist, sollte zur Memory-Pool-Verwaltung ein Programm erstellt werden, das in einer eigenen Task abläuft. Dieses Programm startet die Anwendung folgendermaßen:
 - es legt den Memory Pool mit ENAMP an,
 - es lädt mit ASHARE den Code der Anwendung.Anschließend wechselt dieses Programm in den Wartezustand, bis die Anwendung beendet werden kann. Dann kann es die Beendigung der Anwendung vornehmen:
 - Es entlädt den Code mit DSHARE
 - es löst den Memory-Pool mit DISMP auf
2. Falls Modulaustausch stattfinden soll, empfiehlt es sich, den ILE-Mechanismus des DBL zu verwenden (siehe Handbuch „BLSSERV Bindelader-Starter“ [1]). Dieser ermöglicht es, ein Modul, das als Shared Code geladen ist, ohne Modifikation des PRIVATE-Teils der Anwendung auszutauschen. Außerdem ist es empfehlenswert, die Verbindungen zu den PUBLIC-Modulen zu verwalten, falls die ausgetauschten Module entladen werden sollen. Dies kann durch einen „Benutzungszähler“ erreicht werden, der in der Indirect-Linkage-Routine (mittels CS-Anweisungen) erhöht bzw. vermindert wird. Dann kann DSHARE ausgeführt werden, sobald der Zähler anzeigt, dass keine weitere Task den Code verwendet, der entladen werden soll.

9 Migration

Dieses Kapitel stellt die Unterschiede zwischen dem alten Binder-Lader-System (bis BS2000 V9.5) und dem aktuellen Binder-Lader-System (seit BS2000 V10.0) heraus und soll dem Benutzer eine Umstiegshilfe geben.



Informationen zum Binder TSOSLNK und zum statischen Lader ELDE sowie zur Migration vom TSOSLNK zum BINDER finden Sie im Vorgängerhandbuch „BINDER V2.3“.

9.1 Alte und aktuelle Begriffe

In diesem Abschnitt sind Begriffe des alten und des aktuellen Binder-Lader-Systems gegenübergestellt (siehe auch Kapitel „Fachwörter“).

9.1.1 Alte Begriffe

Bindemodul (OM)

Object Module (OM)

Ladbare Einheit, die durch Übersetzen eines Quellprogramms von einem Sprachübersetzer erzeugt wird.

Bindemodulbibliothek (OML)

Object Module Library (OML)

PAM-Datei, die Bindemodule als Bibliothekselemente enthält. Bindemodulbibliotheken werden ab LMS V2.0A nicht mehr unterstützt. Sie sind durch Programmbibliotheken zu ersetzen.

EAM-Bindemoduldatei

EAM Object Module File

Temporäre System-Bindemodulbibliothek, in die von einem Sprachübersetzer Bindemodule (OMs) abgespeichert werden.

Programmbibliothek

Program Library

PAM-Datei, die mit der Bibliotheks-Zugriffsmethode PLAM bearbeitet wird. Enthält Bibliothekselemente, die durch den Elementtyp und die Elementbezeichnung eindeutig bestimmt sind.

Segment

Programmteil, der getrennt geladen werden kann und unabhängig von anderen Programmteilen ablaufen kann.

DLL

Dynamischer Bindelader des bisherigen Binder-Lader-Systems. Bindet Bindemodule (OMs) und Großmodule (GMs) zu einem ablauffähigen Programm und lädt dieses.

9.1.2 Aktuelle Begriffe

Bindelademodul (LLM)

Link and Load Module (LLM)

Ladbare Einheit mit einer logischen und einer physischen Struktur. Wird vom BINDER erzeugt und als Bibliothekselement vom Elementtyp L in einer Programmbibliothek gespeichert.

Modul

Oberbegriff für Bindemodul (OM) und Bindelademodul (LLM).

Ladeeinheit

Load Unit

Enthält alle Module, die mit *einem* Ladeaufruf geladen werden. Jede Ladeeinheit liegt in einem Kontext.

Kontext

Ein Kontext kann sein:

- ein Satz von Ladeeinheiten,
- ein Umfeld für Binden und Laden,
- ein Umfeld für Entladen und Entbinden.

Ein Kontext hat einen Geltungsbereich und eine Zugriffsberechtigung.

Slice

Ladbare Einheit, in der alle Programmabschnitte (CSECTs) zusammengefasst sind, die zusammenhängend geladen werden. Slices bilden die physische Struktur eines Bindelademoduls (LLM).

Binder-Lader-Starter (BLS)

Bezeichnung des aktuellen Binder-Lader-Systems.

BINDER

Binder des aktuellen Binder-Lader-Systems. Bindet Module zu einem Bindelademodul (LLM).

DBL

Dynamischer Bindelader des aktuellen Binder-Lader-Systems. Bindet Module zu einer Ladeeinheit und lädt diese.

9.2 Merkmale des aktuellen Binder-Lader-Systems

Das Konzept für den aktuellen Binder-Lader-Starter (BLS) unterscheidet sich wesentlich vom Konzept für das alte Binder-Lader-System. Im folgenden sind die Hauptmerkmale des aktuellen Konzepts zusammengestellt.

9.2.1 Bindelademodul (LLM) und BINDER

- Das Format des Bindelademoduls (LLM) ermöglicht ein optimales Laden gegenüber den bisherigen Bindemodulen (OMs) und Großmodulen.
- Die logische Struktur eines LLM ermöglicht dem Benutzer, seine Anwendung zu strukturieren. Der Benutzer kann beim Erzeugen oder Ändern eines LLM einen Knoten entfernen oder ersetzen oder in ein LLM den Knoten eines anderen LLM einfügen.
- Ein LLM, das in einer Programmbibliothek gespeichert ist, kann geändert werden. Dabei können Module ausgetauscht oder zusätzlich in das LLM aufgenommen werden.
- Der Benutzer kann vereinbaren, dass sämtliche CSECTs, die die gleichen Attribute oder die gleiche Kombination von Attributen haben, vom BINDER zu Slices zusammengefasst werden. Dadurch wird beim Laden des LLM die Ladezeit und der Hauptspeicherbedarf wesentlich verringert.
- Symbole können mit Platzhalter (Wildcards) ausgewählt werden, z.B. beim Ändern der Maskierung von Symbolen (MODIFY-SYMBOL-VISIBILITY).
- In ein LLM kann Test- und Diagnoseinformation (LSD) eingefügt werden. Der Benutzer kann dabei einzelne Bindemodule (OMs) oder Sub-LLMs auswählen, in die Test- und Diagnoseinformation eingefügt wird.
- Der BINDER benutzt eine SDF-Schnittstelle. Die Anweisungen können sowohl im Dialogbetrieb wie im Stapelbetrieb eingegeben werden. Jede Anweisung wird nach der Eingabe *sofort* verarbeitet.
- Beim Erzeugen eines LLM kann sich der Benutzer zu jedem Zeitpunkt Listen mit Informationen über den Zustand des aktuellen LLM ausgeben lassen (SHOW-MAP). Der Benutzer kann dann entscheiden, ob in das aktuelle LLM weitere Module eingefügt oder ob Module entfernt werden sollen. Die Listen werden auf SYSLST oder SYSOUT ausgegeben. Der BINDER benutzt dabei implizit das SDF-Kommando SHOW-FILE. In einem BINDER-Lauf stehen dem Benutzer neben der Listenausgabe noch andere komfortable Informationsfunktionen zur Verfügung.

9.2.2 Dynamischer Bindelader DBL

- Der Benutzer kann Kontexte verwenden. Dies bringt folgende Vorteile:
 - Mehrere Kopien des gleichen Programms können in verschiedene Kontexte geladen werden.
 - Teile einer umfangreichen Anwendung können in verschiedene Kontexte geladen werden. In jedem einzelnen Kontext werden Externverweise getrennt befriedigt. Jede Teilanwendung in einem Kontext kann daher als selbstständige „Sub-Anwendung“ geladen und gestartet werden. So ist es z.B. möglich, die einzelnen Module eines Laufzeitsystems in getrennten Kontexten zu laden und zu starten.
 - Teile einer Anwendung, die zu einem Kontext gehören, können mit *einem* Aufruf entladen werden.
 - Symbole mit gleichen Namen in verschiedenen Kontexten bewirken keine Namenskonflikte, da jeder Kontext seine eigene Symboltabelle hat.
- Beim Aufruf des DBL kann eine Liste mit Informationen über die logische Struktur und den Inhalt der geladenen Ladeeinheit angefordert werden.
- Für die Autolink-Funktion können bis zu 100 alternative Bibliotheken angegeben werden.
- Der Benutzer kann nach seinen Erfordernissen festlegen, wie Namenskonflikte und unbefriedigte Externverweise behandelt werden.
- Beim Laden einer Ladeeinheit mit dem Makro BIND kann der Benutzer festlegen, dass:
 - REP-Sätze aus einer REP-Datei auf die Module der Ladeeinheit angewendet werden,
 - vom DBL benutzte Bibliotheken am Ende der Bearbeitung des DBL-Aufrufs eröffnet bleiben. Dadurch kann die Verarbeitung beschleunigt werden, wenn der DBL mehrmals mit derselben Bibliothek aufgerufen wird.

9.2.3 DBL und BINDER

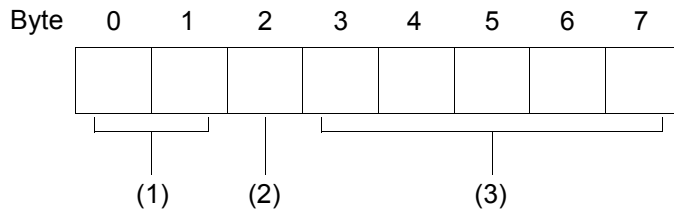
- Eines der Hauptmerkmale des aktuellen Binder-Lader-Starters (BLS) ist die weitgehende Harmonisierung zwischen BINDER und DBL. Das bedeutet:
 - Alle Informationen eines LLM, die vom BINDER beim Erzeugen des LLM festgelegt wurden, können unmittelbar vom neuen DBL ausgewertet werden.
 - Das Befriedigen von Externverweisen wird beim BINDER und beim DBL nach den gleichen Regeln durchgeführt.
- Funktionen, die der DLL nur auf dynamisch geladene Bindemodule (OMs) anwenden konnte (z.B. Möglichkeiten von Änderungen, Ausgabe von Informationen, Entladen), kann der DBL auf die gesamte Ladeeinheit anwenden.
- Sowohl DBL wie BINDER verarbeiten bis zu 32 Zeichen lange Symbolnamen. Diese Symbolnamen können vom BINDER durch Umbenennen der ursprünglichen Symbolnamen in einem LLM erzeugt werden (RENAME-SYMBOLS).
- Der BINDER kann LLMs mit Slices erzeugen, die nach dem Attribut PUBLIC oder PRIVATE der CSECTs gebildet werden. Die PUBLIC-Slice kann der Benutzer für gemeinsam benutzbar erklären, indem er die PUBLIC-Slice mit dem DBL-Makro ASHARE in einem Common Memory Pool lädt.

10 Anhang: Beschreibung der ISAM-Schlüssel

In den BINDER-Anweisungen SHOW-MAP, MODIFY-MAP-DEFAULTS und SHOW-LIBRARY-ELEMENTS kann der Benutzer auswählen, auf welchem Ausgabemedium die gewünschten Informationen erscheinen sollen. Bei Auswahl von OUTPUT \neq *SYSLST gibt BINDER die Informationen in Form einer ISAM-Datei aus. Die Schlüssel dieser ISAM-Dateien sind acht Byte lang und werden in den folgenden Abschnitten beschrieben.

ISAM-Schlüssel in BINDER-Listen (MAPs)

Der ISAM-Schlüssel besteht aus drei Teilen:



Byte 0 bis 1: kennzeichnet den Teil der BINDER-MAP:

Dezimalwert	Bedeutung
05	HELP INFORMATION
10	GLOBAL INFORMATION
15	LOGICAL STRUCTURE
17	SCOPE PATH INFORMATION
20	PHYSICAL STRUCTURE, wenn das LLM benutzerdefinierte Slices hat
25	PHYSICAL STRUCTURE, wenn das LLM nach Attributen gebildete Slices oder nur eine Einzel-Slice hat
30	PROGRAM MAP
31	COMMON LIST
35	UNRESOLVED REFERENCES
36	UNRESOLVED LONG NAMES
37	NOT REFERENCED SYMBOLS
40	SORTED SYMBOLS
45	PSEUDO REGISTERS
50	UNUSED MODULES
55	DUPLICATE SYMBOLS
60	INPUT INFORMATION
61	LINKNAME CONVERSION

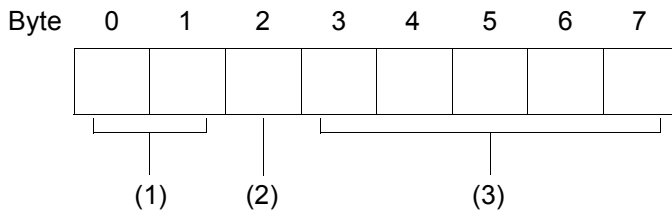
(1) (2) Byte 2: kennzeichnet die Satzart im dem Teil der BINDER-MAP:

Dezimalwert	Bedeutung
0	header record
1	information record
9	end of section

Bytes 3 bis 7: sequenzielle Nummer des Satzes der jeweiligen Satzart, dezimal, bei neuer Satzart wieder bei 0 beginnend

(3) ISAM-Schlüssel in den Listen von SHOW-LIBRARY-ELEMENTS

Der ISAM-Schlüssel besteht aus drei Teilen:



- (1) Byte 0 bis 1: kennzeichnet die Art der Liste, die bei SHOW-LIBRARY-ELEMENTS ausgegeben wird:

Dezimalwert	Bedeutung
90	DUPLICATE SYMBOLS
50	SYMBOLS IN LIBRARY
00	LIBRARY CONTENT

- (2) Byte 2: kennzeichnet die Satzart in der gewählten Liste:

Dezimalwert	Bedeutung
0	header record
1	information record
9	end of section

- (3) Bytes 3 bis 7: sequenzielle Nummer des Satzes der jeweiligen Satzart, dezimal, bei neuer Satzart wieder bei 0 beginnend

Fachwörter

In diesem Fachwörterverzeichnis sind die wichtigsten Begriffe des Binder-Lader-Systems (BLS) aufgenommen. Querverweise auf andere Fachwörter sind *kursiv* dargestellt.

Ablaufinvariantes Programm

Reentrant program

Programm, dessen Code während der Ausführung nicht verändert wird. Dies ist eine Voraussetzung für die Verwendung als *Shared Code*.

Adressierungsmodus

AMODE

Attribut eines *Programmabschnitts* (*CSECT*). Hardware-Adressierungsmodus, den ein *Programm* oder eine *Ladeinheit* beim Ablauf erwartet. Er kann eingestellt werden auf:

- 24-Bit-Adressierung (AMODE=24)
- 31-Bit-Adressierung (AMODE=31)
- 24- oder 31-Bit-Adressierung (AMODE=ANY).

Aktuelles LLM

Current LLM

Erzeugtes oder geändertes *Bindelademodul* (LLM), das im Arbeitsbereich des BINDER aufgebaut wird.

Aktuelle Slice

Current Slice

Slice, in die *Module* eingefügt oder in der *Module* ersetzt werden, wenn keine Angaben über die Position in der *physischen Struktur* des LLM gemacht werden. Gilt nur für *benutzerdefinierte Slices*.

AMODE

Adressierungsmodus

Attribut

Merkmal, das einem *Programmabschnitt (CSECT)* beim Übersetzen zugeordnet werden kann. Eine CSECT kann folgende Attribute haben:

- Lesezugriff (READ-ONLY)
- Hauptspeicherresident (RESIDENT)
- Gemeinsam benutzbar (PUBLIC)
- *Residenzmodus (RMODE)*
- Ausrichtung (ALIGNMENT)
- *Adressierungsmodus (AMODE)*

Autolink

Automatisches Suchen und Einfügen von *Modulen*.

Bedingter Externverweis

WXTRN

Hat die Eigenschaften eines *Externverweises (EXTRN)*, wird jedoch nur bedingt befriedigt. *Autolink* kann auf WXTRNs nicht angewendet werden.

Benutzerdefinierte SLices

User defined Slices

physische Struktur eines Bindelademoduls (LLM), bei der die *Slices* mit Anweisungen SET-USER-SLICE-POSITION vom Benutzer festgelegt werden. Dabei können Überlagerungsstrukturen gebildet werden.

Bindelademodul

Link and Load Module (LLM)

Ladbare Einheit mit einer *logischen Struktur* und einer *physischen Struktur*. Wird vom BINDER erzeugt und als Bibliothekselement vom *Elementtyp L* in einer *Programmbibliothek* oder in einer PAM-Datei (*PAM-LLM*) gespeichert.

Bindemodul

Object Module (OM)

Ladbare Einheit, die durch Übersetzen eines Quellprogramms mit Hilfe eines Sprachübersetzers erzeugt wird.

Bindemodulbibliothek

Object Module Library (OML)

PAM-Datei, die *Bindemodule* als Bibliothekselemente enthält.

BINDER-Lauf

Folge von BINDER-Anweisungen, die nach dem Ladeaufruf für den BINDER beginnt und mit der END-Anweisung endet.

COMMON-Bereich

Common Section

Datenbereich, der von mehreren *Programmabschnitten (CSECTs)* gemeinsam zur Kommunikation benutzt werden kann.

Common Memory Pool

Speicherbereich im Klasse-6-Speicher (Benutzerspeicher), an den sich mehrere Benutzer anschließen und darauf zugreifen können.

CSECT

Programmabschnitt

EAM-Bindemoduldatei

EAM Object Module File

Temporäre System-Bindemodulbibliothek, in die von einem Compiler *Bindemodule (OMs)* abgespeichert werden.

Edit-Lauf

Umfasst eine Folge von BINDER-Anweisungen, die mit der Anweisung START-LLM-CREATION oder START-LLM-UPDATE beginnt und mit der nächsten Anweisung START-LLM-CREATION oder START-LLM-UPDATE oder mit der END-Anweisung endet.

Einsprungstelle

ENTRY

Symbolische Verknüpfungsadresse, die in einem *Modul* definiert ist, jedoch auch von einem anderen Modul benutzt werden kann.

Einzel-Slice

Single Slice

physische Struktur eines Bindelademoduls (LLM), bei der das LLM aus einer einzigen *Slice* besteht.

Elementbezeichnung

Legt ein Bibliothekselement in einer *Programmbibliothek* fest. Setzt sich zusammen aus *Elementname* und *Elementversion*.

Elementname

Name eines Bibliothekselements in einer *Programmbibliothek* oder *Bindemodulbibliothek*. Die Anweisungen des BINDER und die Kommandos des DBL nehmen darauf Bezug.

Elementtyp

Typ eines Bibliothekselements in einer *Programmbibliothek*.
Für Programmbibliotheken gelten folgende Elementtypen.

Typ C *Programm (Lademodul)*

Typ L *Bindelademodul (LLM)*

Typ R *Bindemodul (OM)*

Elementversion

Versionsbezeichnung eines Bibliothekselements in einer *Programmbibliothek*.
Die Anweisungen des BINDER und die Kommandos des DBL nehmen darauf Bezug.

ENTRY

Einsprungstelle

ESD

Externadressbuch

ESV

Externadressbuch

Externadressbuch

External Symbol Dictionary (ESD)

External Symbols Vector (ESV)

Enthält alle *Programmdefinitionen* und *Referenzen* in einem *Modul*. Wird benötigt zum Befriedigen von Referenzen. Bindemodule (OMs) enthalten das Externadressbuch in Form von ESD-Sätzen. In Bindelademodulen (LLMs) sind ESV-Sätze enthalten.

Externer Pseudoabschnitt

XDSEC

Programmteil, für den in der *Textinformation* eines *Moduls* kein Abbild besteht. Ein externer Pseudoabschnitt kann eine *Referenz (XDSEC-R)* oder eine *Programmdefinition (XDSEC-D)* sein.

Externverweis

EXTRN

Symbolische Verknüpfungsadresse, die in einem *Modul* verwendet wird, jedoch in einem anderen Modul definiert ist. Wird unbedingt befriedigt, entweder explizit oder durch *Autolink*.

EXTRN

Externverweis

Geltungsbereich eines Kontext

Legt die Speicherklasse fest, in der ein *Kontext* liegt. Der Kontext kann im Benutzeradressraum (USER) oder im Systemadressraum (SYSTEM) liegen.

Großmodul

Synonym für *vorgebundenen Bindemodul*

ILE

Indirect Linkage Entry

Einsprungstelle (ENTRY), die den Aufrufer über eine *IL-Routine* an einen *ILE-Server* weiterleitet. Ein ILE hat folgende Merkmale:

- Name
- Adresse der *IL-Routine*
- Adresse des *ILE-Servers*
- Distanz der ILE-Serveradresse in der *IL-Routine*
- Status des *ILE-Servers* (aktiv oder nicht aktiv)
- Steuerungsanzeige (system- oder benutzergesteuert)

ILE-Server

Modul, das Programmcode wie ein Unterprogramm enthält, das aber über ein *ILE* und eine *IL-Routine* angesprochen werden kann.

IL-Routine

Indirect Linkage Routine

Routine, die einen *ILE-Server* aufruft. Eine IL-Routine kann vom Benutzer auch selbst definiert werden, wenn er nicht die vom System standardmäßig bereitgestellte IL-Routine verwenden will.

Indirektes Binden

Indirect Linkage

Bindemechanismus, bei dem ein *Externverweis* nicht wie bisher direkt mit einer Programmdefinition, sondern über eine zwischengeschaltete *IL-Routine* durch ein *ILE* befriedigt wird.

Interner Name

Internal Name

Wird beim Erzeugen eines *Bindelademoduls (LLM)* festgelegt und kennzeichnet die Root in der *logischen Struktur* des LLM.

Kontext

Ein Kontext kann sein:

- ein Satz von Objekten mit einer *logischen Struktur*,
- ein Umfeld für Binden und Laden,
- ein Umfeld für Entladen und Entbinden.

Ein Kontext hat einen *Geltungsbereich* und eine *Zugriffsberechtigung*.

Ladeinheit

Load Unit

Enthält alle *Module*, die mit *einem* Ladeaufruf geladen werden. Jede Ladeinheit liegt in einem *Kontext*.

LLM

Bindelademodul

Logische Struktur eines Kontext

Logical Structure of a Context

Hierarchische Struktur eines *Kontext* als Satz von Objekten. Objekte sind *Programmabschnitte (CSECTs)*, *Module* und *Ladeeinheiten*.

Logische Struktur eines LLM

Logical Structure of a LLM

Legt die Baumstruktur eines *Bindelademoduls (LLM)* fest. Die Wurzel ist der *interne Name*, die Knoten sind *Sub-LLMs* und die Blätter sind *Bindemodule (OMs)* und leere *Sub-LLMs*.

Logische Strukturinformation

Information in einem *Bindelademodul (LLM)*, die die *logische Struktur* des LLM festlegt.

LRLD

Relativierungsinformation

LSD

Test- und Diagnoseinformation.

OM

Bindemodul

OML

Bindemodulbibliothek

Modul

Oberbegriff für *Bindemodul (OM)* und *Bindelademodul (LLM)*.

Nach Attributen gebildete Slices

Slices by Attributes

physische Struktur eines Bindelademoduls (LLM), bei der die Slices nach Attributen von Programmabschnitten (CSECTs) gebildet werden.

PAM-LLM

LLM, das vom Binder BINDER in eine PAM-Datei abgespeichert wurde.

Pfadname

Path Name

Name, mit dem *Sub-LLMs* in der *logischen Struktur* eines LLM adressiert werden. Besteht aus einer Folge von Einzelnamen, die durch einen Punkt voneinander getrennt sind.

Physische Struktur eines LLM

Physical Structure of a LLM

Legt fest, aus welchen *Slices* ein *Bindelademodul (LLM)* aufgebaut ist.

Man unterscheidet:

Einzel-Slices

nach Attributen gebildete Slices

benutzerdefinierte Slices.

Physische Strukturinformation

Information in einem *Bindelademodul (LLM)*, die die *physische Struktur* des LLM beschreibt.

Programmabschnitt

Control Section (CSECT)

Programmteil, der unabhängig von anderen Programmteilen geladen werden kann. Ein Programmabschnitt kann bestimmte *Attribute* haben.

Programmbibliothek

Program Library

PAM-Datei, die mit der Bibliotheks-Zugriffsmethode PLAM bearbeitet wird. Enthält Bibliothekselemente, die durch den *Elementtyp* und die *Elementbezeichnung* eindeutig bestimmt sind.

Programmdefinition

Program Definition

Oberbegriff für
Programmabschnitt (CSECT)
Einsprungstelle (ENTRY)
Indirect Linkage Entry (ILE)
COMMON-Bereich
Externer Pseudoabschnitt (XDSEC-D)

Pseudo-Register

Pseudo Register

Hauptspeicherbereich, der zur Kommunikation verschiedener Programmteile untereinander benutzt wird.

Pseudo-RMODE

Residenzmodus (RMODE) eines *Moduls*. Wird vom BINDER oder DBL aus dem Residenzmodus aller *CSECTs* des Moduls festgelegt.

Referenz

Reference

Oberbegriff für
Externverweis (EXTRN)
V-Konstante
Bedingter Externverweis (WXTRN)
Externer Pseudoabschnitt (XDSEC-R)

Relativierungsinformation

Relocation Linkage Dictionary (RLD) Local ReLocation Dictionary (LRLD)

Information in einem *Modul*, die festlegt, wie Adressen beim Binden und Laden auf eine gemeinsame Bezugsadresse ausgerichtet werden. *Bindemodule* enthalten die Relativierungsinformation in Form von RLD-Sätzen. In *Bindelademodulen* (LLMs) sind LRLD-Sätze enthalten.

Residenzmodus

RMODE

Attribut eines *Programmabschnitts (CSECT)*. Legt fest, ob die CSECT oberhalb und unterhalb 16Mbyte (RMODE=ANY) oder nur unterhalb 16Mbyte (RMODE=24) geladen wird.

RLD

Relativierungsinformation

RMODE

Residenzmodus

Shared Code

Code, der von mehreren Tasks gleichzeitig benutzt werden kann. Er kann im Klasse-4-Speicher oder in einem *Common Memory Pool* des Klasse-6-Speichers abgelegt werden. Voraussetzung für die gleichzeitige Benutzbarkeit ist, dass er als *ablaufinvariantes Programm* erstellt wurde.

Servermodul

ILE-Server

Slice

Ladbare Einheit, in der alle *Programmabschnitte (CSECTs)* zusammengefasst sind, die zusammenhängend geladen werden. Slices bilden die *physische Struktur* eines *Bindelademoduls (LLM)*.

Sub-LLM

Unterstruktur in der *logischen Struktur* eines LLM. Besteht aus *Bindemodulen (OMs)* oder weiteren Sub-LLMs und wird durch den *Pfadnamen* adressiert.

Symbol

Oberbegriff für *Programmdefinition* und *Referenz*. Ist durch einen Symbolnamen gekennzeichnet.

Test- und Diagnoseinformation

List for Symbolic Debugging (LSD)

Information in einem *Modul*, die von den Test- und Diagnosehilfen für das Testen auf Quellprogrammebene benötigt wird.

Textinformation

Information in einem *Modul*, die aus dem Code und den Daten besteht.

V-Konstante

Adresskonstante, die in einem *Modul* definiert ist, deren Adresse jedoch in einem anderen Modul benutzt wird. Wird unbedingt befriedigt, entweder explizit oder durch *Autolink*.

WXTRN

Bedingter Externverweis

XDSEC

Externer Pseudoabschnitt

Zugriffsberechtigung eines Kontext

Legt fest, welche Benutzer auf einen *Kontext* zugreifen dürfen. Der Kontext kann privilegiert oder nicht privilegiert sein.

Literatur

Die Handbücher finden Sie im Internet unter <http://manuals.ts.fujitsu.com>. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie auch in gedruckter Form bestellen.

- [1] **BLSSERV**
Bindelader-Starter in BS2000
Benutzerhandbuch
- [2] **ASSEMBH** (BS2000)
Benutzerhandbuch
- [3] **LMS** (BS2000)
SDF-Format
Benutzerhandbuch
- [4] **BS2000 OSD/BC**
Systeminstallation
Benutzerhandbuch
- [5] **BS2000 OSD/BC**
Kommandos
Benutzerhandbuch
- [6] **JV** (BS2000)
Jobvariablen
Benutzerhandbuch
- [7] **BS2000 OSD/BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch
- [8] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

- [9] **BS2000 OSD/BC**
Einführung in die Systembetreuung
Benutzerhandbuch
- [10] **BS2000 OSD/BC**
Systemmeldungen
Benutzerhandbuch
- [11] **Einführung in die XS-Programmierung**
(für ASSEMBLER-Programmierer) (BS2000)
Benutzerhandbuch
- [12] **BS2000**
Programmiersystem
Technische Beschreibung
- [13] **SDF (BS2000)**
Einführung in die Dialogschnittstelle SDF
Benutzerhandbuch
- [14] **SDF-A (BS2000)**
Benutzerhandbuch
- [15] **BS2000 OSD/BC**
Einführung in das DVS
Benutzerhandbuch

Stichwörter

A

Abkürzungsmöglichkeiten bei SDF 192
Adressierungsmodus 114, 255
aktuelle Programmbibliothek 43
aktuelle Slice 69
aktuelles LLM 37, 40, 43, 287
 speichern 331
 Stufe rücksetzen 214
aktuelles Sub-LLM 57, 211
Aliasname 196
ALIGNMENT 114, 255
alphanum-name (Datentyp) 197
AMODE 114, 255
ändern
 Attribute von Modulen 244
 Attribute von Symbolen 114, 255
 LLM 31, 40
 LLM (Link and Load Module) 331
 LLM, Beispiel 41
 logische Struktur eines LLM 66
 logischen Namen eines LLM 66
 Maskierung von Symbolen 117, 261
 Merkmale eines LLM 31, 43, 235
 Merkmale von Modulen 66
 Pfadname eines Moduls 66
 physische Struktur 43
 Struktur eines LLM 244
 Symbolnamen 265
 Symbolnamen, Beispiele 108
 Typ von Externverweisen 259
 Typ von Symbolen 119
Anweisung, Geltungsbereich 138
Anweisungen für BINDER 138, 189
 Einteilung nach Funktionen 189
 Übersicht 210

Arbeitsbereich des BINDER 37, 40, 43, 47, 331
 löschen 325
Attribute
 einer CSECT 18
 Slices nach COMMON-Behandlung 103
 von Symbolen 114
Attribute ändern
 von Modulen 244
 von Symbolen 114, 255
 von Symbolen, Beispiele 115
aufrufen, BINDER 169
Auftragsschalter 233
Ausgaben, Listen des BINDER 141, 314
Ausgabeziele für BINDER-Listen 134, 141
Ausrichtung (ALIGNMENT) 114, 255
Autolink-Funktion 80, 85, 279
 Regeln 85

B

backtracking 26
bedingter Externverweis (WXTRN) 80, 105, 107,
 265, 301
beenden, BINDER 169
Beendigungscode für BINDER-LAUF 170
befriedigen von Externverweisen 80, 279
 Beispiele 82
 Regeln 81
BEGIN-SUB-LLM-STATEMENTS 52, 57, 211,
 264
Beginn eines Sub-LLM 57
behandeln, Symbole 105
Benutzeroberfläche, Hinweise 192
Benutzerschalter 233
Bezugsadresse eines LLM 104
Bibliothekselement 15

Bindelademodul (LLM) [9](#), [15](#), [16](#), [140](#), [141](#)
Definition [9](#)

Bindemodul (OM) [9](#), [16](#), [140](#)

Bindemodulbibliothek (OML) [9](#)

bei INCLUDE-MODULES [47](#), [215](#)

bei REPLACE-MODULES [54](#), [270](#)

Eingabequelle für BINDER [140](#)

Binden [9](#)

Binder

BINDER [9](#)

BINDER-Anweisungen [35](#)

BINDER-Funktionen [35](#)

BINDER-Konzept [15](#)

BINDER-Lauf [15](#), [80](#), [135](#), [140](#), [169](#)

beenden [169](#), [214](#)

Beendigungscode [171](#)

Geltungsbereich [138](#)

Programminformation [171](#)

starten [169](#)

Überwachung mit Jobvariable [170](#)

Zustandsanzeige [170](#)

BINDER-Läufe

konkurrierende [140](#), [290](#), [331](#)

BINDER-Makroaufruf [173](#)

BINDER-MAPs, ISAM-Schlüssel [142](#)

BLSCOPYN (Systemparameter) [239](#), [329](#)

BLSLDPXS [298](#)

BLSLIBnn [80](#), [217](#), [280](#), [309](#)

bei REPLACE-MODULES [272](#)

BLSSEC [10](#)

C

c-string (Datentyp) [197](#)

cat (Zusatz zu Datentypen) [208](#)

cat-id (Datentyp) [197](#)

command-rest (Datentyp) [197](#)

COMMON-Bereich [103](#), [105](#)

Attribute ändern [114](#), [255](#)

bei Namenskonflikten [102](#)

umbenennen [107](#), [265](#)

COMMON-Promotion [103](#)

Compiler [9](#)

compl (Zusatz zu Datentypen) [203](#)

composed-name (Datentyp) [197](#)

Copyright-Information [37](#), [43](#), [239](#), [329](#)

corr (Zusatz zu Datentypen) [208](#), [209](#)

CSECT [105](#)

Attribute [18](#), [19](#)

Attribute ändern [114](#), [255](#)

Behandlung durch Autolink [279](#)

suchen [80](#)

umbenennen [107](#), [265](#)

D

date (Datentyp) [197](#)

Datentypen SDF [193](#), [197](#), [202](#)

Zusätze [194](#)

DBL (Dynamic Binding Loader) [10](#)

device (Datentyp) [197](#)

Dialogbetrieb [15](#), [81](#)

DUPLICATE-LIST bei BINDER-Listen [143](#)

dynamischer Bindelader DBL [18](#)

E

EAM-Bindemoduldatei [9](#)

bei INCLUDE-MODULES [47](#), [215](#)

bei REPLACE-MODULES [54](#), [270](#)

Eingabequelle für BINDER [140](#)

Edit-Lauf [38](#), [240](#)

Geltungsbereich [138](#)

einfügen, Module [31](#), [47](#), [215](#)

Beispiel [49](#)

Eingaben für BINDER [138](#)

Eingabequelle für BINDER [47](#), [54](#), [215](#), [270](#)

aktuelle [47](#)

eröffnen [140](#)

Einsprungstelle (ENTRY) [105](#)

suchen [80](#)

umbenennen [107](#), [265](#)

Einzel-Slice [19](#), [21](#), [237](#), [327](#)

COMMON-Behandlung [103](#)

ELEMENT

bei INCLUDE-MODULES [217](#)

bei REPLACE-MODULES [273](#)

bei SAVE-LLM [24](#), [290](#)

bei SHOW-LIBRARY-ELEMENT [309](#)

bei START-LLM-UPDATE [333](#)

- Elementname
 - bei INCLUDE-MODULES 217
 - bei REPLACE-MODULES 273
 - bei SAVE-LLM 37, 43, 290
 - bei SHOW-LIBRARY-ELEMENT 309
 - bei START-LLM-UPDATE 333
 - Definition 24
 - gleicher 43, 287
- Elementtyp L 9, 15, 16
 - bei INCLUDE-MODULES 215
 - bei REPLACE-MODULES 54, 270
 - bei SAVE-LLM 43, 325
 - bei START-LLM-UPDATE 331
- Elementtyp R 9
 - als BINDER-Eingabe 140
 - bei INCLUDE-MODULES 215
 - bei REPLACE-MODULES 270
- Elementversion 47, 54
 - bei INCLUDE-MODULES 217
 - bei REPLACE-MODULES 273
 - bei SAVE-LLM 37, 43, 290
 - bei SHOW-LIBRARY-ELEMENT 309
 - bei START-LLM-UPDATE 333
 - Definition 24
 - höchste 54
- END-Anweisung 140, 170, 173, 214
- END-SUB-LLM-STATEMENTS 52, 57, 211, 214
- Ende eines Sub-LLM 57
- entfernen, Module 52, 264
- ENTRY 105
 - Behandlung durch Autolink 279
 - suchen 80
 - umbenennen 107, 265
- ENTRY-POINT 299
- eröffnen, Eingabequelle 140
- ersetzen, Module 31, 54, 270
 - Beispiel 55
- erzeugen, eines LLM 31, 37, 325
 - Beispiel 39
- ESV (External Symbols Vector)
 - bei Namenskonflikten 102
 - bei SAVE-LLM 45, 296
 - beim Maskieren von Symbolen 117
 - Definition 30
- EXIT-ROUTINE bei SHOW-MAP 321
- exklusive Slice 68
- Externadressbuch (ESV)
 - bei Maskieren von Symbolen 117
 - bei Namenskonflikten 102
 - bei SAVE-LLM 45, 296
 - Definition 30
- externer Pseudoabschnitt (XDSEC-D) 105
- Externverweis 105
 - befriedigen 80, 279
 - befriedigen, Regeln 81
 - nicht referenziert 80, 119
 - umbenennen 107, 265
 - unbefriedigt 96, 301
- EXTRN
 - siehe Externverweis 80, 105, 107, 265, 301
- F**
- FATAL ERROR 233
 - Fehlerklasse 135
- Fehlerbehandlung steuern 135, 231
- Fehlerklasse 172, 231
 - FATAL ERROR 135, 233
 - für Meldungen 136
 - INFORMATION 135, 233
 - INTERNAL ERROR 135, 233
 - RECOVERABLE ERROR 135, 233
 - SYNTAX ERROR 135, 233
 - UNRESOLVED EXTERNS 135, 233
 - WARNING 135, 233
- filename (Datentyp) 198
- fixed (Datentyp) 197
- full-filename siehe Datentyp filename 198
- G**
- garantierte Abkürzungen 192
- Geltungsbereich
 - Anweisung 138
 - BINDER-Lauf 138
 - Edit-Lauf 138
- gemeinsam benutzbar (PUBLIC) 20, 75, 114, 255
- gen (Zusatz zu Datentypen) 208
- gleichzeitiger Zugriff auf LLMs 140, 290, 331

GLOBAL-INFORMATION 316

bei BINDER-Listen 142

Liste 146

Großmodul 15, 16

H

Hauptspeicherresident 19, 75

Hauptspeicherresident (RESIDENT) 114

HELP-INFORMATION 316

bei BINDER-Listen 142

Hinweise, Benutzeroberfläche 192

I

Identifikation eines LLM 24

INCLUDE-MODULES 31, 40, 47

Format 215

INCLUSION-DEFAULTS

bei MODIFY-LLM-ATTRIBUTES 240

bei START-LLM-CREATION 38, 329

bei START-LLM-UPDATE 334

Index

global 206

Konstruktionszeichenfolge 206

platzhalter-spezifisch 206

Schreibweise 207

INFORMATION 232

Fehlerklasse 135

Inhalt eines LLM 30

INPUT-INFORMATION 319

bei BINDER-Listen 143

integer (Datentyp) 199

INTERNAL ERROR, Fehlerklasse 135

INTERNAL-NAME

bei MODIFY-LLM-ATTRIBUTES 43, 236

bei START-LLM-CREATION 37, 326

Root in logischer Struktur eines LLM 17, 24

INTERNAL-VERSION

bei MODIFY-LLM-ATTRIBUTES 43, 237

bei START-LLM-CREATION 37, 327

Definition 24

interne Version

bei MODIFY-LLM-ATTRIBUTES 43, 237

bei SAVE-LLM 44, 290

bei START-LLM-CREATION 37, 327

Definition 24

interner Name

bei MODIFY-LLM-ATTRIBUTES 43, 236

bei SAVE-LLM 44, 290

bei START-LLM-CREATION 37, 326

Root in logischer Struktur eines LLM 17, 24

ISAM-Schlüssel 188

Beschreibung 369

BINDER-MAPs 142

J

Jobvariable

zur Überwachung des BINDER-Laufs 170

K

Klasse-2-Systemparameter

BLSCOPYN 239, 329

BLSLDPXS 298

Knoten 17, 25, 57, 211

konkurrierende BINDER-Läufe 140, 290, 331

Konstruktionsangabe 207

Konstruktionszeichenfolge 206

Kurzname 196

bei SDF 193

L

Ladeadresse

COMMON-Bereich 103

eines LLM 104, 298

Lademodul 15

Laden 9

Laden und Starten des BINDER 169

Lader 9

Laufzeitmodule bei RESOLVE-BY-

AUTOLINK 86

Lesezugriff (READ-ONLY) 19, 75, 114, 255

LIBRARY

bei INCLUDE-MODULES 216

bei REPLACE-MODULES 272

bei RESOLVE-BY-AUTOLINK 280

bei SAVE-LLM 289

bei START-LLM-UPDATE 332

- Listen 300
 - Ausgabeziel 311, 319
 - ausgeben 141, 314
 - benennen 142
 - der Abkürzungen 142, 144, 316
 - der COMMON-Bereiche 154
 - der mehrfachen Programmdefinitionen 143, 157, 319
 - der nicht benutzten Module 143, 319
 - der Pseudoregister 143, 156, 319
 - der unbefriedigten Externverweise 142, 318
 - der unbenutzten Module 156
 - des BINDER 134, 141
 - Gemischte Module 143
 - Globale Informationen 142, 146
 - ISAM-Schlüssel 142
 - Kopfzeile 142, 316
 - mit Eingabeinformationen 143, 157, 319
 - Programmübersicht 142, 151, 317
 - sortierte Liste der
 - Programmdefinitionen 143, 155, 318
 - Übersicht über die logische Struktur 142, 148, 316
 - Übersicht über die physische Struktur 142, 150, 317
- Listenausgabe für BINDER, Beispiel 143
- LLM (Link and Load Module) 15, 16, 140, 141
 - aktuelles 43, 287
 - ändern 31, 40, 331
 - ändern, Beispiel 41
 - bei BINDER-Listen 142
 - Bezugsadresse 104
 - Definition 9
 - erzeugen 31, 37, 325
 - erzeugen, Beispiel 39
 - Identifikation 24
 - Inhalt 30
 - Ladeadresse 104, 298
 - logische Struktur 17, 30
 - logische Struktur erzeugen 211, 214
 - logische Struktur erzeugen oder ändern 57
 - logische Struktur erzeugen, Beispiel 59
 - Merkmale ändern 31, 43, 235
 - physische Struktur 18, 37, 237, 327
 - physische Struktur erzeugen 68, 69
 - physische Struktur erzeugen, Beispiel 70
 - physische Struktur festlegen 303
 - physische Struktur, Beispiel 23
 - physische Struktur, Diagrammdarstellung 68
 - speichern 31, 37, 43, 287
 - speichern, Beispiel 45
 - Startadresse 299
 - Struktur ändern 244
- LOAD-ADDRESS 298
- LOGICAL-STRUCTURE
 - bei BINDER-Listen 142
 - bei INCLUDE-MODULES 220
 - bei MODIFY-LLM-ATTRIBUTES 43, 240
 - bei REPLACE-MODULES 275
 - bei RESOLVE-BY-AUTOLINK 282
 - bei SAVE-LLM 45, 297
 - bei SHOW-MAP 316
 - bei START-LLM-CREATION 38, 330
 - bei START-LLM-UPDATE 40, 334
- logische Struktur eines LLM 17, 30, 59
 - ändern 66
 - erzeugen 57, 211, 214
 - Root 17, 57
 - Strukturbaum 17, 57
 - Stufe 17, 57
- logische Strukturinformation eines LLM 30
- logischer Name eines LLM, ändern 66
- low (Zusatz zu Datentypen) 203
- LRLD
 - bei SAVE-LLM 45
 - Definition 30
- LSD (List for Symbolic Debugging)
 - abspeichern 31
 - bei INCLUDE-MODULES 220
 - bei MODIFY-LLM-ATTRIBUTES 43, 240
 - bei MODIFY-MODULE-ATTRIBUTES 66, 245
 - bei REPLACE-MODULES 275
 - bei RESOLVE-BY-AUTOLINK 282
 - bei SAVE-LLM 45, 297
 - bei START-LLM-CREATION 38, 330
 - bei START-LLM-UPDATE 40, 334
 - Definition 30

M

Makro BINDER 173
man (Zusatz zu Datentypen) 208, 209
mandatory (Zusatz zu Datentypen) 209
MAP 134, 141, 300
MAP-NAME 142
maskiertes Symbol 107
Maskierung von Symbolen 102
 ändern 117, 261
MAX-ERROR-WEIGHT 233
Mehrfachzugriff auf LLMs 140, 290, 331
Meldungsbehandlung 136
MERGE-MODULES 120, 225
 Beispiele 123
MERGED-MODULES, bei BINDER-Listen 143
Merkmale eines LLM ändern 43
Merkmale von Modulen ändern 66
MESSAGE-CONTROL 136, 232
Metasyntax SDF 193, 195
Mischen von Modulen 120, 225
MODE 304
MODIFY-ERROR-PROCESSING 135, 136, 231
MODIFY-LLM-ATTRIBUTES 24, 31, 43, 235
MODIFY-MAP-DEFAULTS 134, 241, 300, 314
MODIFY-MODULE-ATTRIBUTES 66, 244
MODIFY-SYMBOL-ATTRIBUTES 114, 255
MODIFY-SYMBOL-TYPE 119, 259
MODIFY-SYMBOL-VISIBILITY 107, 117, 261
Modul 16, 140
 Attribute ändern 244
 einfügen 31, 47, 215
 einfügen, Beispiel 49
 entfernen 52, 264
 entfernen, Beispiel 53
 ersetzen 31, 54, 270
 ersetzen, Beispiel 55
 Merkmale ändern 66
 Priorität 218, 274, 280, 310
MODULE-CONTAINER
 bei INCLUDE-MODULES 216
 bei REPLACE-MODULES 272
 bei SAVE-LLM 289
 bei START-LLM-UPDATE 332

N

NAME 264
 bei INCLUDE-MODULES 219
 bei REPLACE-MODULES 271
name (Datentyp) 199
NAME-COLLISION
 bei MODIFY-MODULE-ATTRIBUTES 66
Namenskonflikte 102
 bei MODIFY-MODULE-ATTRIBUTES 66
 bei MODIFY-SYMBOL-VISIBILITY 117
 bei RENAME-SYMBOLS 107
NEW-NAME 268
 bei REPLACE-MODULES 274
nicht referenzierte Externverweise 80, 119

O

odd (Zusatz zu Datentypen) 208
OM (Object Module) 9
OML (Object Module Library) 9
 bei INCLUDE-MODULES 47, 215
 bei REPLACE-MODULES 54, 270
 Eingabequelle für BINDER 140
OUTPUT 311, 319
OVERWRITE bei SAVE-LLM 43

P

Parameterbereich bei BINDER-Makroaufruf 175
partial-filename (Datentyp) 200
path-compl (Zusatz zu Datentypen) 203
PATH-NAME
 bei BEGIN-SUB-LLM-STATEMENTS 211
 bei INCLUDE-MODULES 219
 bei MERGE-MODULES 225
 bei MODIFY-MODULE-ATTRIBUTES 244
 bei REMOVE-MODULES 264
 bei REPLACE-MODULES 272
 bei RESOLVE-BY-AUTOLINK 86, 282
Pfadname 47, 57, 211, 264
 Abkürzung 26
 bei INCLUDE-MODULES 219
 bei MERGE-MODULES 226
 bei MODIFY-MODULE-ATTRIBUTES 245
 bei RESOLVE-BY-AUTOLINK 86, 282
Definition 25

Pfadname (Forts.)
 eines Moduls, ändern 66
 Struktur 25
 PHYSICAL-STRUCTURE 317
 bei BINDER-Listen 142
 physische Struktur eines LLM 18, 37, 237, 327
 ändern 43
 Beispiel 23
 Diagrammdarstellung 68
 Diagrammdarstellung, Beispiel 69
 erzeugen 68, 69
 erzeugen, Beispiel 70
 festlegen 303
 physische Strukturinformation eines LLM 30
 posix-filename (Datentyp) 200
 posix-pathname (Datentyp) 200
 POSIX-Platzhalter 204
 Priorität von Modulen 47, 218, 274, 280, 310
 private Slice 75
 product-version (Datentyp) 201
 PROGRAM-MAP 317
 bei BINDER-Listen 142
 Programm 15
 Programmabschnitt (CSECT) 18, 105
 Attribute ändern 114, 255
 suchen 80
 umbenennen 107, 265
 Programmbibliothek 9, 15
 aktuelle 43
 bei INCLUDE-MODULES 47, 215
 bei REPLACE-MODULES 54, 270
 bei SAVE-LLM 37, 43
 bei START-LLM-UPDATE 331
 Eingabequelle für BINDER 140
 Programmdefinition 105, 265
 Programminformation für BINDER-Lauf 170
 Protokollierung steuern 134
 PSEUDO-REGISTER 319
 bei BINDER-Listen 143
 Pseudoregister, Pseudoregistervektor 104
 PUBLIC 20, 75, 114, 255

Q

quotes (Zusatz zu Datentypen) 209

R

READ-ONLY 19, 75, 114, 255
 Readme-Datei 13
 RECOVERABLE ERROR 233
 Fehlerklasse 135
 Referenz 105, 265
 Region 69
 Relativierung der Adressen 104
 Relativierungsinformation (LRLD)
 bei SAVE-LLM 45
 Definition 30
 Relativierungsinformation (RLD)
 bei SAVE-LLM 297
 RELOCATION-DATA
 bei SAVE-LLM 45
 REMOVE-MODULES 40, 52, 264
 RENAME-SYMBOLS 107, 265
 REPLACE-MODULES 31, 40, 54, 270
 RESIDENT 19, 75, 114, 255
 Residenzmodus 20, 75, 114, 255
 RESOLUTION 302
 RESOLUTION-SCOPE
 bei BEGIN-SUB-LLM-STATEMENTS 212
 bei INCLUDE-MODULES 221
 bei MODIFY-MODULE-ATTRIBUTES 246
 bei REPLACE-MODULES 276
 bei RESOLVE-BY-AUTOLINK 283
 RESOLVE-BY-AUTOLINK 80, 85, 279
 Returncode, Makro BINDER 176
 RLD (Relocation Linkage Dictionary), bei SAVE-LLM 297
 RMODE 20, 75, 114, 255
 Root 17, 25
 Root-Slice 68, 69, 219, 275
 Rückkehrcode-Anzeige für BINDER-Lauf 170
 RUN-TIME-VISIBILITY
 bei MODIFY-MODULE-ATTRIBUTES 66
 bei RESOLVE-BY-AUTOLINK 86

S

- SAVE-LLM 31, 37
 - Format 287
 - für BINDER-Listen 134
 - nach START-LLM-CREATION 325
 - nach START-LLM-UPDATE 40, 43, 331
- Schachtelung bei Sub-LLMs 57
- Schlüsselwortoperanden bei SDF 193
- SCOPE
 - bei MODIFY-SYMBOL-ATTRIBUTES 114, 256
 - bei MODIFY-SYMBOL-TYPE 260
 - bei MODIFY-SYMBOL-VISIBILITY 117, 262
 - bei RENAME-SYMBOLS 268
 - bei RESOLVE-BY-AUTOLINK 281
 - bei SET-EXTERN-RESOLUTION 302
- sep (Zusatz zu Datentypen) 208
- SET-EXTERN-RESOLUTION 96, 301
- SET-USER-SLICE-POSITION 20, 69, 238, 303, 328
- SHOW-FILE-Kommando 134, 141, 311, 320
- SHOW-LIBRARY-ELEMENTS 307
 - ISAM-Schlüssel der Liste 132, 142
- SHOW-MAP 69, 134
 - Ausgabeziel 141
 - Format 314
 - Standardwerte ändern 241
- SHOW-SYMBOL-INFORMATION 323
- Sicherheitskomponente BLSSEC 10
- SLICE 219, 275
- Slice 18, 219, 303
 - benutzerdefiniert 20, 238, 328
 - benutzerdefiniert, Beispiel 22
 - Einzel- 19, 237, 327
 - Einzel-, Beispiel 21
 - exklusiv 68
 - nach Attributen gebildet 19, 237, 327
 - nach Attributen gebildet, Beispiel 22
 - Region 69
 - Stufennummer 69
- SLICE-DEFINITION 37, 69, 237, 327
 - bei MODIFY-LLM-ATTRIBUTES 43
- SLICE-NAME 303
- Slice-Name 77, 150
- SORTED-PROGRAM-MAP 318
 - bei BINDER-Listen 143
- SPECIAL-HANDLING 233
- speichern, eines LLM 31, 43, 287
 - Beispiel 45
- Speicherresidenz (RESIDENT) 255
- Standard-Dateikettungsname für BINDER-Listen 312, 321
- Standarddateiname für BINDER-Listen 312, 320
- Standardheader 176
- Standardwerte für Ausgabe von Listen ändern 241
- Stapelbetrieb 15, 81
- START-BINDER-Kommando 169
- START-LLM-CREATION 24, 31, 37, 40
 - Format 325
- START-LLM-UPDATE 31, 40
 - Format 331
- START-STATEMENT-RECORDING 335
- Startadresse des LLM 299
- Starter 10
- STATEMENT-LIST
 - bei BINDER-Listen 143
- Stellungsoperanden bei SDF 193
- Stern vor konstantem Operandenwert 192
- steuern
 - Fehlerbehandlung 135, 231
 - Protokollierung 134
- STOP-STATEMENT-RECORDING 336
- structured-name (Datentyp) 201
- Struktur des Pfadnamens 25
- Struktur eines LLM ändern 244
- Strukturinformation eines LLM 38
 - bei INCLUDE-MODULES 220
 - bei MODIFY-LLM-ATTRIBUTES 43, 240
 - bei REPLACE-MODULES 275
 - bei RESOLVE-BY-AUTOLINK 282
 - bei SAVE-LLM 297
 - bei START-LLM-CREATION 330
 - bei START-LLM-UPDATE 334
 - logische 30
 - logische, bei SAVE-LLM 45
 - physische 30

- Stufe in der logischen Struktur eines LLM 17, 25, 57
- Sub-LLM 15, 17, 47, 52, 211
 Beginn 57, 211
 Ende 57, 214
 Schachtelung 57
- Subsystem 20
 Symbole 238, 328
- SUBSYSTEM-ENTRIES 238
- Suchverfahren für Pfadname 26
- Symbol
 Attribute ändern 114, 255
 maskiert 107, 261
 Maskierung ändern 117, 261
 Typ ändern 119
- SYMBOL-DICTIONARY
 bei SAVE-LLM 45, 296
- SYMBOL-NAME 255, 261, 266, 281, 301
 bei SHOW-LIBRARY-ELEMENT 310
- SYMBOL-TYPE 266, 301
- Symbole
 Begriffsdefinition 105
 in Subsystemen 238, 328
 Informationen ausgeben 323
 umbenennen 107, 265
- Symbolnamen, ändern, Beispiele 108
- SYNTAX ERROR 233
 Fehlerklasse 135
- SYSPLAMALT 166
 FILE ID 166, 167
- T**
- temp-file (Zusatz zu Datentypen) 208
- Test- und Diagnoseinformation (LSD) 43, 66
 bei INCLUDE-MODULES 220
 bei MODIFY-LLM-ATTRIBUTES 240
 bei MODIFY-MODULE-ATTRIBUTES 245
 bei REPLACE-MODULES 275
 bei RESOLVE-BY-AUTOLINK 282
 bei SAVE-LLM 45, 297
 bei START-LLM-CREATION 38, 330
 bei START-LLM-UPDATE 40, 334
 Definition 30
- TEST-OPTIONS
 bei REPLACE-MODULES 275
- TEST-SUPPORT
 bei INCLUDE-MODULES 220
 bei MODIFY-LLM-ATTRIBUTES 43, 240
 bei MODIFY-MODULE-ATTRIBUTES 66, 245
 bei RESOLVE-BY-AUTOLINK 282
 bei SAVE-LLM 45, 297
 bei START-LLM-CREATION 38, 330
 bei START-LLM-UPDATE 40, 334
- text (Datentyp) 201
- Textinformation (TXT) 30
- time (Datentyp) 201
- TXT (Text) 30
- Typ von Symbolen 259
 ändern 119
- TYPE 218, 274, 280, 310
 bei INCLUDE-MODULES 47
- U**
- Überlagerungsstrukturen bei Slices 20
- Überschreiben
 LLM als Bibliothekselement 43
- umbenennen
 Symbole 107, 265
- unbefriedigte Externverweise 96, 301
 Beispiele 96
- under (Zusatz zu Datentypen) 203
- UNRESOLVED EXTERNS 233
 Fehlerklasse 135
- UNRESOLVED-LIST 318
 bei BINDER-Listen 142
- UNUSED-MODULE-LIST 319
 bei BINDER-Listen 143
- user (Zusatz zu Datentypen) 209
- USER-COMMENT 142, 316
- V**
- V-Konstante 105, 107, 265, 301
- vers (Zusatz zu Datentypen) 209

VERSION

- bei INCLUDE-MODULES [217](#)
- bei REPLACE-MODULES [273](#)
- bei SAVE-LLM [24](#)
- bei START-LLM-UPDATE [333](#)

VISIBLE [262](#)

- vorgebundenen Bindemodul [15, 16](#)
- vsn (Datentyp) [201](#)

W

WARNING [233](#)

- Fehlerklasse [135](#)

WARNING-MESSAGE [267](#)

wild(n) (Zusatz zu Datentypen) [204](#)

with (Zusatz zu Datentypen) [203](#)

without (Zusatz zu Datentypen) [208](#)

WXTRN [105](#)

- bei RENAME-SYMBOLS [107](#)
- bei SET-EXTERN-RESOLUTION [301](#)
- umbenennen [265](#)
- unbefriedigt [80](#)

X

x-string (Datentyp) [202](#)

x-text (Datentyp) [202](#)

XDSEC-D [105](#)

Z

Zusätze zu Datentypen [194, 203](#)

Zustandsanzeige, BINDER-Lauf [170](#)