BLSSERV V2.7

Bindelader-Starter in BS2000/OSD

Ausgabe Mai 2009

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 9001:2000

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

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

Copyright und Handelsmarken

Inhalt

1 Einleitung

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 **Binder-Lader-System**, 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 Programmlauf 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.

1.1 Kurzbeschreibung des Binder-Lader-Systems

Im **Binder-Lader-System (BLS)** stehen dem Benutzer folgende Funktionseinheiten zur Verfügung:

- der Binder BINDER,
- der alte Binder TSOSLNK,
- das Subsystem BLSSERV mit der Funktionalität des dynamischen Bindeladers DBL und des statischen Laders ELDE,
  BLSSERV V2.7A ist ablauffähig ab BS2000/OSD-BC V6.0.
- 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), die von Compilern erzeugt und in einer Bindemodulbibliothek (OML), einer Programmbibliothek (Typ R) oder in der temporären EAM-Bindemoduldatei gespeichert wurden,

– bereits gebundene oder von Compilern erzeugte LLMs aus einer Programmbibliothek (Typ L),

– bereits gebundene LLMs aus einer PAM-Datei (PAM-LLM),

– vorgebundene Bindemodule (Großmodule), die vom Binder TSOSLNK gebunden und in einer Bindemodulbibliothek (OML), in einer Programmbibliothek (Typ R) oder in der temporären EAM-Bindemoduldatei gespeichert wurden.

Binder TSOSLNK

Der Binder TSOSLNK bindet:

– ein oder mehrere Bindemodule (OMs) zu einem ablauffähigen Programm (Lademodul) und speichert dieses in einer katalogisierten Programmdatei oder als Bibliothekselement vom Typ C in einer Programmbibliothek,

– mehrere Bindemodule (OMs) zu einem einzigen vorgebundenen Modul (Großmodul) und hinterlegt dieses als Bibliothekselement vom Typ R in einer Programmbibliothek oder in der EAM-Bindemoduldatei.

Der Benutzer sollte an Stelle des Binders TSOSLNK den Binder BINDER verwenden, da TSOSLNK nicht weiterentwickelt und durch BINDER abgelöst wird.

BLSSERV mit dem Dynamischen Bindelader DBL und dem statischen Lader ELDE

Der **dynamische Bindelader DBL** hat die Aufgabe, Module zu einer Ladeeinheit zu binden und diese zu laden. Die Funktionalität von DBL ist ein Bestandteil des Subsystems BLSSERV.

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),

– Bindemodule (OMs), die von Compilern erzeugt und in einer Bindemodulbibliothek (OML), in einer Programmbibliothek (Typ R) oder in der temporären EAM-Bindemodul-datei gespeichert wurden,

– vorgebundene Bindemodule (Großmodule), die vom Binder TSOSLNK gebunden und in einer Bindemodulbibliothek (OML), in einer Programmbibliothek (PLAM, Elementtyp R) oder in der temporären EAM-Bindemoduldatei gespeichert wurden.

Der **statische Lader ELDE** hat die Aufgabe, ein ablauffähiges Programm zu laden, das vom Binder TSOSLNK gebunden und in einer Programmdatei oder als Bibliothekselement vom Typ C in einer Programmbibliothek gespeichert wurde. Die Funktionalität von ELDE ist ein Bestandteil des Subsystems BLSSERV.

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 oder von ELDE 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 werden. Bild 1 zeigt das Zusammenwirken dieser Funktionseinheiten.

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

Die Binder BINDER und TSOSLNK sind Dienstprogramme. Der dynamische Bindelader DBL und der statische Lader ELDE dagegen gehören zum Subsysstem BLSSERV, das ein Teil des BS2000-Organisationsprogrammes ist. Sie bieten ihre Funktionalität über BS2000-Kommandos und über Programmschnittstellen an. Den Anstoß zur Ausführung eines geladenen Programmes gibt ein Starter, der Teil des Subsysstems BLSSERV ist und für den Benutzer unsichtbar bleibt.

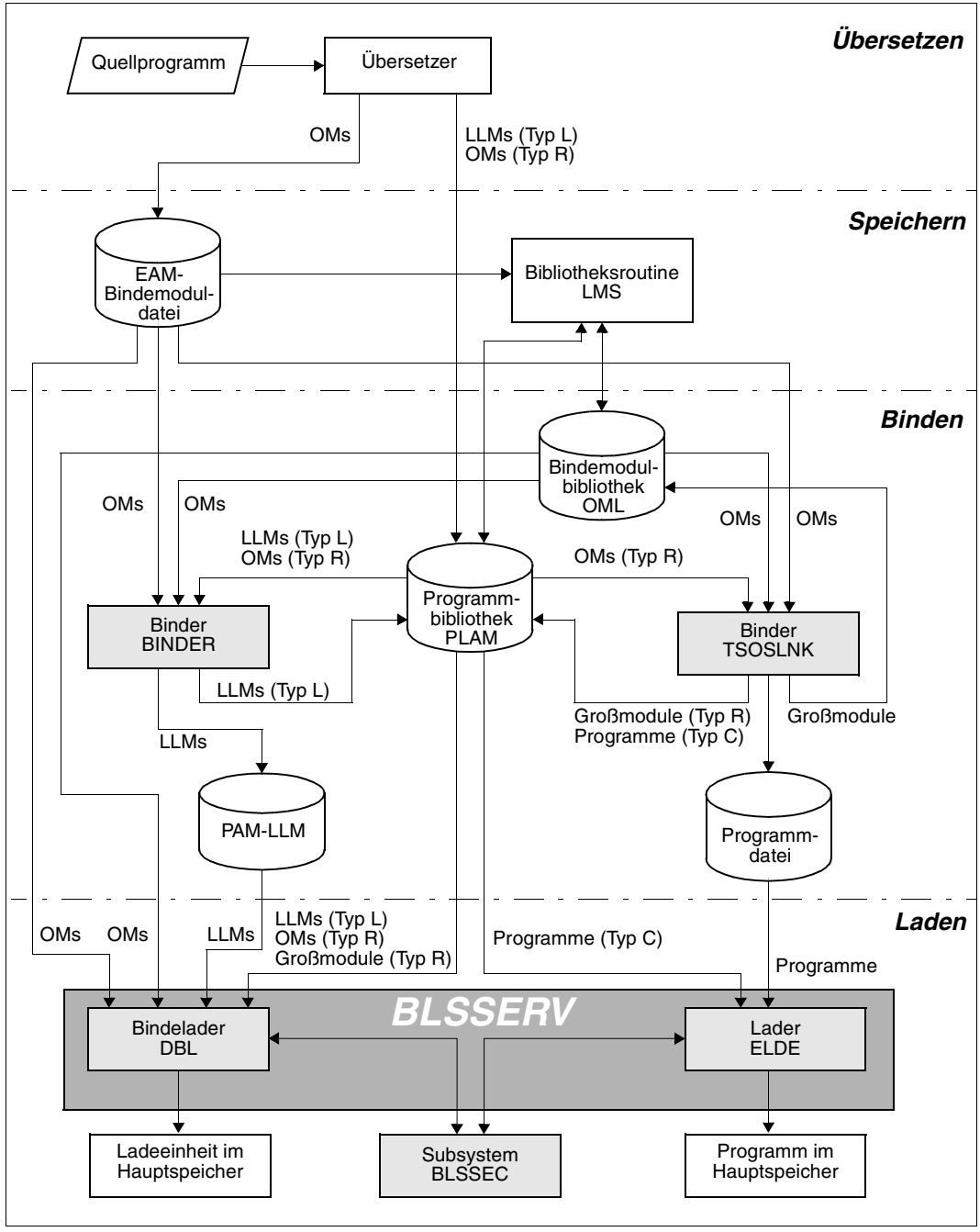


Bild 1: Zusammenwirken der Funktionseinheiten für das Binden und Laden

1.2 Zielsetzung und Zielgruppen des Handbuchs

Das Handbuch „Bindelader-Starter“ wendet sich an die Software-Entwicklung. Es dient zum einen der Beschreibung der Leistung und Anwendung des Dynamischen Bindeladers DBL und des statischen Laders ELDE als Bestandteil von BLSSERV, zum anderen dient es als Nachschlagewerk für deren Kommandos und Makros.

1.3 Konzept des Handbuchs

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

Das vorliegende Handbuch beschreibt den Dynamischen Bindelader DBL und den statischen Lader ELDE.

Das Benutzerhandbuch „BINDER“ [1] enthält die Beschreibung des Binders BINDER. Zum BINDER gibt es ein Tabellenheft mit den BINDER-Anweisungsformaten.

Das vorliegende Handbuch enthält im Einzelnen:

- im ersten Kapitel eine Übersicht über die Funktionseinheiten des Binder-Lader-Systems (BLS) und deren Zusammenwirken
- im zweiten Kapitel die Beschreibung des Dynamischen Bindeladers DBL mit den Ein-/Ausgaben, den Besonderheiten für gemeinsam benutzbare Programme, der Erläuterung des Kontextkonzeptes, zusätzlichen Funktionen des DBL, dem Ablauf des DBL sowie eine Übersicht über die Kommandos und Makros für den DBL-Aufruf
- im dritten Kapitel die XS-Unterstützung des Dynamischen Bindeladers
- im vierten Kapitel eine kurze Beschreibung des Laders ELDE
- im Kapitel „Migration“ die wesentlichen Unterschiede zwischen dem alten Binder-/Lader-Konzept (bis BS2000 V9.5) und dem aktuellen Binder-Lader-System (ab BS2000 V10.0). Es soll dem Benutzer eine Umstiegshilfe geben.
- im Kapitel „Performantes Laden von Programmen/Produkten“ Hinweise zur Beschleunigung des Ladevorgangs und zur Reduzierung des Betriebsmittelverbrauchs.

Readme-Dateien

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei. Sie finden die Readme-Datei auf Ihrem BS2000-Rechner unter dem Dateinamen SYSRME.BLSSERV.*version*.*sprache*. Die Benutzerkennung, unter der sich die Readme-Datei befindet, erfragen Sie bitte bei Ihrer zuständigen Systembetreuung. Die Readme-Datei können Sie mit dem Kommando SHOW-FILE oder mit einem Editor ansehen oder auf einem Standarddrucker mit folgendem Kommando ausdrucken:

```
/PRINT-DOCUMENT dateiname, LINE-SPACING=*BY-EBCDIC-CONTROL
```

1.4 Änderungen gegenüber der vorigen Version

Die vorliegende Ausgabe des Handbuchs „Bindelader-Starter“ enthält gegenüber dem Vorgängerhandbuch („BLSSERV V2.5, Bindelader-Starter“) folgende Änderungen:

- BLSSERV V2.7 unterstützt X86-Code.
- In der DBL-Liste sind zusätzliche Informationen enthalten:
  - für LLMs und Bindemodule (OM): Bibliotheksname, Elementname und Elementversion
  - für PAM-LLMs: vollständiger Dateiname.
- Beim BIND-Makro können neben den aus dem aktuellen Ladevorgang nicht befriedigt verbliebenen Externverweisen auch die aus früheren Ladevorgängen in einen benutzerdefinierten Datenbereich ausgegeben werden.
- Bei den Makros ASHARE und ILEMGT wurde die Maximalzahl paralleler Memory Pools für Shared Code auf 16 erhöht.
- Die redundanten Kommando- und Makrobeschreibungen wurden aus diesem Handbuch entfernt. Die Kommandos sind im Handbuch „Kommandos“ [5], die Makros im Handbuch „Makroaufrufe an den Ablaufteil“ [7] detailliert beschrieben.

1.5 Verwendete Metasprache

1.5.1 Darstellungsmittel

In diesem Handbuch werden folgende Darstellungsmittel verwendet:

für Hinweise

halbfett besonders wichtige Begriffe im Fließtext

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

Literaturhinweise werden im Text in Kurztiteln und eckigen Klammern [ ] angegeben. Der vollständige Titel jeder Druckschrift, auf die verwiesen wird, ist im Literaturverzeichnis aufgeführt.

2 Der dynamische Bindelader DBL

Der dynamische Bindelader ist ein Teil des Subsystems BLSSERV. BLSSERV wird unmittelbar nach Startup geladen. Damit steht die Funktionalität des DBL über die Programmschnittstelle und über einige BS2000-Kommandos zur Verfügung. DBL kann daher nicht wie ein Dienstprogramm aufgerufen werden.
Der DBL hat die Aufgabe, Module zu einer Ladeeinheit zu binden und diese zu laden. Zu diesem Zweck muss für das zu ladende Modul eine Umgebung aufgebaut werden, die es ermöglicht, alle Externverweise in diesem Modul zu befriedigen. Das wird durch das Kontextkonzept des DBL realisiert. Der DBL unterstützt außerdem die Mehrfachnutzung von Modulen, indem er Funktionen zum Laden und Entladen von gemeinsam benutzbaren Programmen (Shared Code) zur Verfügung stellt.

Zusätzlich führt der DBL folgende Funktionen aus:

- Entladen und Entbinden von Objekten,
- Ausgeben von Binde- und Ladeinformation.

2.1 Binden und Laden

Das Binden und Laden lässt sich in 3 Komplexe einteilen (siehe Bild 2):

- Eingaben für den DBL
- Bindevorgang
- Ausgaben des DBL

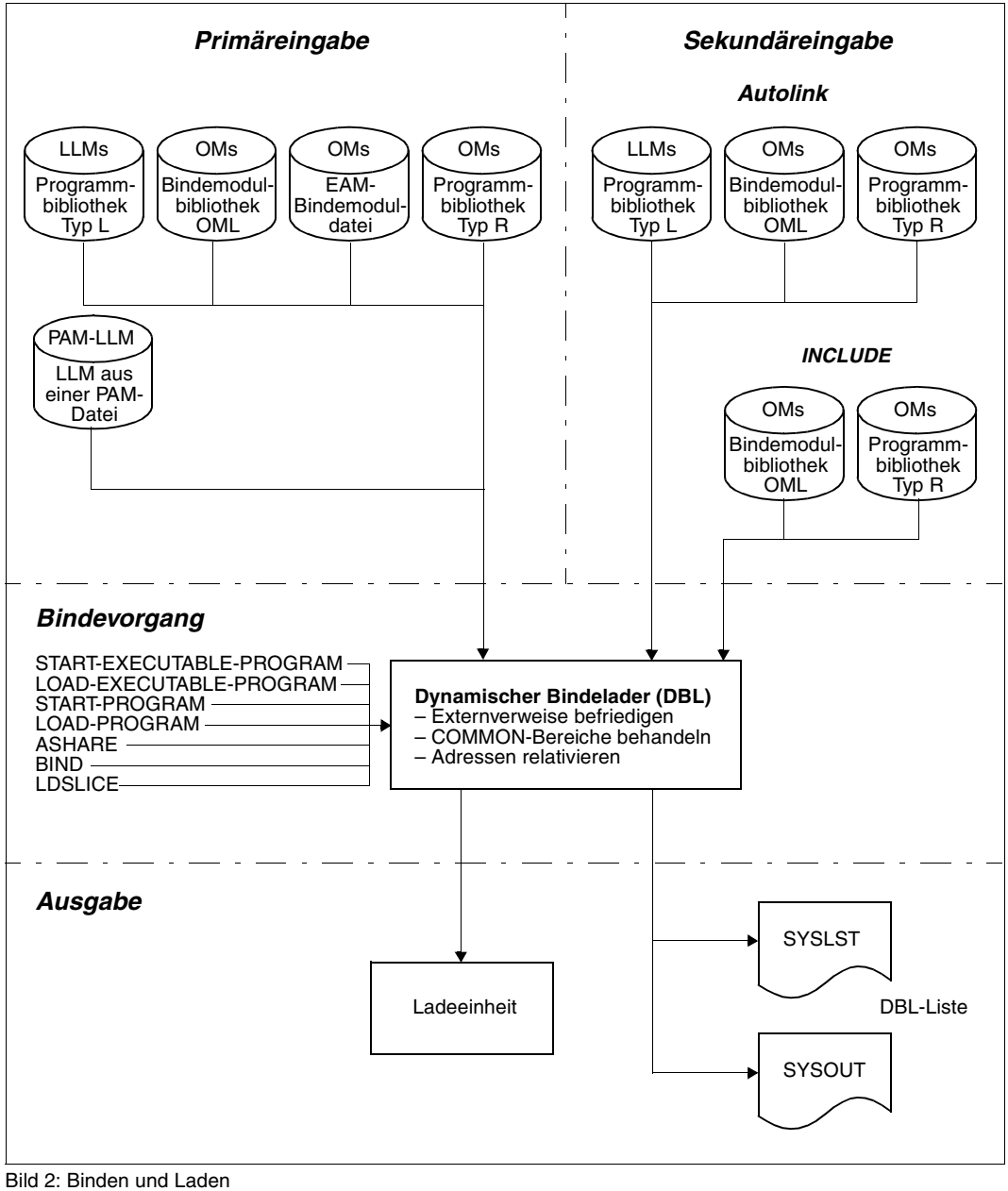


Bild 2: Binden und Laden

2.1.1 Eingaben für den DBL

Eingaben für den DBL können folgende **Module** sein:

– Bindelademodule (LLMs), die vom BINDER gebunden oder von Compilern erzeugt wurden,

– Bindemodule (OMs) und vorgebundene Bindemodule.

Bindelademodule

Bindelademodule (LLMs) werden vom Binder BINDER erzeugt und in Programmbibliotheken als Typ-L-Element oder in PAM-Dateien abgelegt. LLMs in PAM-Dateien werden als PAM-LLMs bezeichnet. Compiler wie z.B. der C-Compiler können ebenfalls LLMs erzeugen und diese in Bibliotheken ablegen (siehe Handbücher der Compiler).

Die logische Struktur eines LLM wird durch eine Baumstruktur realisiert. Die Wurzel in dieser Baumstruktur ist der interne Name des LLM, mit dem der BINDER das LLM identifiziert. Die Knoten (nodes) sind Sub-LLMs, die hierarchisch in Stufen (levels) angeordnet sind. Bindemodule bilden die letzte Stufe (Blätter) in der logischen Struktur des LLM.
Zur externen Identifizierung des LLM werden Elementname und Elementversion des LLM als Bibliothekselement (Typ L) genutzt. Bei PAM-LLMs wird der Name der PAM-Datei zur externen Identifizierung benutzt.

Ein LLM hat neben der logischen Struktur auch eine physische Struktur. Die Bindemodule eines LLM bestehen aus Programmabschnitten, den CSECTs, die getrennt in den Hauptspeicher geladen werden können. Die CSECTs eines oder mehrerer OMs werden zu einer Slice zusammengefasst. Eine Slice ist damit eine ladbare Einheit für den DBL.

Es gibt drei Typen der physischen Struktur eines LLM:

1. LLM mit einer einzelnen Slice (das LLM wird in einem Ladevorgang vollständig in den Speicher geladen).
2. LLM mit nach Attributen gebildeten Slices:
   CSECTs, die die gleichen Attribute (READ-ONLY, RESIDENT, PUBLIC, RMODE) haben, werden zu einer Slice zusammengefasst (bei PAM-LLMs nicht erlaubt).
3. LLMs mit benutzerdefinierten Slices:
   Der Benutzer legt die Positionen der Slices selbst fest. Er kann damit Overlay-Strukturen erzeugen (bei PAM-LLMs max. 16 Slices).

Die genaue Beschreibung der LLMs, ihrer Identifikation sowie der logischen und physischen Struktur ist im Handbuch „BINDER“ [1] zu finden.

Bindemodule

Bindemodule (OM) werden von Compilern erzeugt. Vorgebundene Bindemodule, die der Binder TSOSLNK erzeugt, haben dasselbe Format wie OMs und werden daher im Folgenden auch als OMs betrachtet.

Eingabequellen für Bindemodule können sein:

– Programmbibliotheken (PLAM, Elementtyp R),

– Bindemodulbibliotheken (OML),

– die EAM-Bindemoduldatei.

Primäreingabe und Sekundäreingabe

Man unterscheidet je nach Eingabezeitpunkt die Primär- und die Sekundäreingabe.

Die **Primäreingabe** wird in dem Kommando oder Makroaufruf angegeben, der den DBL aufruft. Der Benutzer gibt im Kommando oder Makroaufruf das Modul und wahlweise auch die Eingabequelle an. Das Modul kann ein Bindemodul oder ein LLM sein.

Die **Sekundäreingabe** wird in folgenden Fällen durchgeführt:

– falls der DBL auf Grund unbefriedigter Externverweise und V-Konstanten weitere Module aus einer Eingabequelle einfügen muss (Autolink-Funktion, siehe Seite 24f).

– falls INCLUDE-Anweisungen in Bindemodulen vorhanden sind, die weitere Bindemodule aus einer Eingabequelle anfordern. Dies gilt nur für Bindemodule. LLMs enthalten keine INCLUDE-Anweisungen.

2.1.2 Bindevorgang

Der DBL bindet die Module einer Ladeeinheit in folgenden Schritten:

– Primäreingabe.
Der DBL sucht das Modul, das im Ladeaufruf angegeben ist.

– Befriedigen von Externverweisen.
Der DBL versucht, zu allen Externverweisen in diesem Modul pmnde Namen von Programmabschnitten und Einsprungstellen zu finden. Dazu sucht der DBL in der Sekundäreingabe die entsprechenden Module und bindet sie ein. Ergeben sich durch die neu dazugekommenen Module wieder unbefriedigte Externverweise, so wird dieser Prozess iterativ solange wiederholt, bis alle Externverweise befriedigt sind bzw. bis ein bestimmter vereinbarter Status für noch unbefriedigte Externverweise vorhanden ist.

– Behandlung der COMMON-Bereiche.
Der DBL prüft in allen Modulen, ob COMMON-Bereiche definiert sind und reserviert für sie Speicherplätze.

– Relativierung der Adressen.
Alle Adressen in den Modulen werden so umgesetzt, dass sie sich auf dieselbe Bezugsadresse beziehen.

Diese Schritte des Bindevorgangs werden im Folgenden detailliert beschrieben. Besonderheiten beim Laden von List-Name-Units und von PAM-LLMs finden Sie in Abschnitt „Verwaltung von List-Name-Units“ auf Seite 38 und in Abschnitt „Laden von LLMs aus Dateien (PAM-LLMs)“ auf Seite 41.

Suchvorgang der Primäreingabe

Der DBL sucht das Modul, das in den Kommandos START/LOAD-EXECUTABLE-PROGRAM (bzw. START/LOAD-PROGRAM) oder im Makroaufruf BIND angegeben wurde, in verschiedenen Containern.

Der angegebene Modulname kann sein:

– der Name einer CSECT oder eines ENTRY, die nicht maskiert sein dürfen,

– der externe Name eines Bindemoduls oder eines LLMs,

– der Name eines Programms im Shared Code im Klasse-3/4/5-Speicher (geladen als unprivilegiertes Subsystem),

– der Name eines Programmes im Shared Code des Benutzers.

Der Suchvorgang verläuft in folgenden Stufen:

1. Suche im Link-Kontext, der vom Benutzer angegeben wurde (siehe Seite 56). Findet der DBL nicht maskierte CSECTs oder ENTRYs, die dem angegebenen Symbolnamen im Ladeaufruf entsprechen, dann übergibt er die Startadresse an die Benutzertask und der Ladevorgang wird beendet.

   – Die Suche im Link-Kontext erfolgt nur beim Makroaufruf BIND.

2. Suche im Shared Code des Benutzers, der mit dem ASHARE-Makro des DBL in den Common Memory Pool geladen wurde. Der Suchvorgang endet, wenn der DBL ein gemeinsam benutzbares Programm (definiert mit dem Parameter PROGRAM im ASHARE-Makro) oder eine nicht maskierte CSECT bzw. ein ENTRY mit diesem Namen findet. Eine Startadresse wird übergeben und der Ladevorgang wird abgeschlossen.

   – Die Suche im Shared Code des Benutzers wird nur im RUN-MODE=*ADVANCED ausgeführt.

   – Die Suche im Shared Code des Benutzers wird **nicht** ausgeführt, wenn beim Kommando START/LOAD-EXECUTABLE-PROGRAMM NAME-SCOPE=*ELEMENT angegeben ist.

   – In Common Memory Pools wird nur dann gesucht, wenn der Benutzer SHARE-SCOPE=*MEMORY-POOL(...)/*ALL angegeben hat, da Default-Wert für SHARE-SCOPE=*SYSTEM-MEMORY ist. Die Suche kann auf Common Memory Pools mit einem definierten Geltungsbereich eingeschränkt oder ganz unterdrückt werden (SHARE-SCOPE=*NONE).

3. Suche im Shared Code des Systems, der aus den unprivilegierten Subsystemen im Klasse-3/4/5-Speicher besteht (siehe Handbuch „Verwaltung von Subsystemen“ [10]). Werden nicht maskierte CSECTs oder ENTRYs gefunden, die dem angegebenen Symbolnamen im Ladeaufruf entsprechen, übergibt der DBL die Ladeadresse an die Benutzertask und der Ladevorgang wird beendet. Der DBL baut also eine Verbindung zwischen der Benutzertask und dem Subsystem auf.
   Im RUN-MODE=*ADVANCED kann der Benutzer im Ladeaufruf das Durchsuchen der unprivilegierten Subsysteme unterdrücken (Operand SHARE[-SCOPE]=*NONE).

   – Die Suche im Shared Code des Systems wird **nicht** ausgeführt, wenn beim Kommando START/LOAD-EXECUTABLE-PROGRAMM NAME-SCOPE=*ELEMENT angegeben ist.

4. Durchsuchen der Bibliothek, die der Benutzer im Ladeaufruf angegeben hat (Operand LIBRARY bzw. LIBNAM@/LIBNAM/LIBLINK). Wenn im RUN-MODE= *ADVANCED eine solche Bibliothek nicht angegeben wurde, aber der Link-Name BLSLIB einer Bibliothek zugewiesen wurde, dann verwendet der DBL diese Standardbibliothek mit dem Link-Namen BLSLIB. Ist die angegebene Bibliothek fehlerhaft oder existiert sie nicht, so wird die Verarbeitung mit einer Fehlermeldung abgebrochen.

5. Durchsuchen von **alternativen Bibliotheken**.

   Es gibt zwei Gruppen von alternativen Bibliotheken:

   a) Alternative Bibliotheken, die über Dateikettungsnamen vereinbart wurden.

      Hierbei unterscheidet man zwischen

      – alternativen Benutzerbibliotheken mit den Dateikettungsnamen BLSLIBnn und
      – alternativen Systembibliotheken mit den Dateikettungsnamen $BLSLBnn
        Alternative Systembibliotheken werden für Komponenten des von der Fujitsu ausgelieferten Laufzeitsystems benötigt. Sie werden über den Dateikettungsnamen $BLSLBnn (00≤nn≤99) von der jeweiligen Laufzeitkomponente selbst zugewiesen. Der Benutzer hat darauf keinen Einfluss. Die Dateikettungsnamen der alternativen Systembibliotheken werden von der Software-Entwicklung der Fujitsu Technology Solutions GmbH verwaltet und sind ausschließlich für deren Softwareprodukte reserviert.

   b) System- und Benutzer-Tasklibs

      Hierbei handelt es sich um Bibliotheken mit dem Namen TASKLIB oder um eine Bibliothek, die mit dem Kommando SET-TASKLIB zugewiesen wurde.

   Über Dateikettungsnamen vereinbarte alternative Bibliotheken werden in folgender Reihenfolge durchsucht:

   1. Alternative Systembibliotheken mit den Dateikettungsnamen $BLSLB00 .. 49
   2. Alternative Bibliotheken, die vom Benutzer mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99) zugewiesen wurden.
   3. Alternative Systembibliotheken mit den Dateikettungsnamen $BLSLB50 .. 99

   Innerhalb einer Gruppe von Bibliotheken werden diese nach aufsteigenden Nummern „nn“ durchsucht.

   System- und Benutzertasklibs werden in folgender Reihenfolge durchsucht:

   1. Die Bibliothek, die vom Benutzer mit dem Kommando SET-TASKLIB zugewiesen wurde
   2. Die Bibliothek „$userid.TASKLIB“
      oder, falls diese nicht existiert:
      Die Bibliothek „$defluid.TASKLIB“.
      Dabei ist „defluid“ der Name der System-Standardkennung (Wert des Klasse-2-Systemparameters DEFLUID, siehe Handbuch „Einführung in die Systembetreuung“ [9]).

Welche alternativen Bibliotheken durchsucht werden, hängt davon ab, auf welche Weise der Ladevorgang durchgeführt wird:

– Laden mit START/LOAD-EXECUTABLE-PROGRAM
oder mit dem BIND-Makro (mit INTVERS=SRVxxx, wobei xxx ≥ 002)

  Der Benutzer kann mit dem Operanden ALTERNATE-LIBRARIES bzw. ALTLIB festlegen, welche der genannten Bibliotheken in welcher Reihenfolge durchsucht werden.

– Laden mit START/LOAD-PROGRAM und RUN-MODE=*ADVANCED
oder mit dem BIND-Makro (INTVERS=BLSP2/SRV001)

  Nur die mit den Dateikettungsnamen BLSLIBnn bzw. $BLSLBnn vereinbarten alternativen Bibliotheken werden durchsucht, falls ALTERNATE-LIBRARIES=*YES angegeben wurde.

– Laden mit START/LOAD-PROGRAM und RUN-MODE=*STD

  Nur die System- und Benutzer-Tasklibs werden durchsucht.

Auswahl und Zuweisung einer Programmversion

Ab BS2000/OSD-BC V3.0 ist es möglich, eine Programmversion auszuwählen, die der DBL bei Suchen nach der Primäreingabe und bei der Befriedigung von Externverweisen verwenden soll. Ein Programm ist aus der Sicht von DBL eine Ladeeinheit mit einer bestimmten Version. Alle Programmdefinitionen (CSECT, ENTRY, COMMON, ...), die in der Ladeeinheit enthalten sind, erben diese Version.

Die Auswahl einer Programmversion ist auf zwei Arten möglich:

1. Versionsauswahl vor dem Laden und Starten:

   Mit dem Kommando SELECT-PROGRAM-VERSION oder dem Makroaufruf SELPRGV wird die Version einer Ladeeinheit ausgewählt, die DBL bei späteren Ladeaufrufen verwenden soll. Die Ladeeinheit muss zum Zeitpunkt der Versionsauswahl noch nicht geladen sein; es können aber auch schon mehrere Versionen dieser Ladeeinheit geladen sein.

2. Versionsauswahl im Ladeaufruf:

   Beim Ladeaufruf mit den Kommandos LOAD- und START-EXECUTABLE-PROGRAM (bzw. LOAD- und START-PROGRAM) oder mit den Makros BIND und ASHARE kann eine Version angegeben werden (Operand PROGRAM-VERSION bzw. PGMVERS). Der DBL sucht dann unter den bereits geladenen Objekten eine Ladeeinheit mit diesem Namen und genau dieser Version. In diese Suche werden auch DSSM-Subsysteme einbezogen. Der DBL betrachtet die Subsystemversion als Programmversion und berücksichtigt alle „Connectable Entries“ (siehe Seite 51) des Subsystems.

   Die Versionsauswahl im Ladeaufruf überschreibt eine vorhergehende Versionsauswahl.

Der DBL weist einer zu ladenden Ladeeinheit eine Version zu, wenn er unter den bereits geladenen Objekten nicht die ausgewählte Version (siehe 1. und 2.) findet.

Mit dem Makro GETPRGV kann der Benutzer abfragen, welche Version für ein Programm ausgewählt ist.

Die Programmversion kann auch beim Entladen und Entbinden angegeben oder als Binde- und Ladeinformation ausgegeben werden.

Befriedigen von Externverweisen

Der DBL versucht, alle nicht befriedigten Externverweise in den Modulen einer Ladeeinheit zu befriedigen. Dazu durchsucht er zuerst die bereits eingebundenen Module nach CSECTs oder ENTRYs mit dem gleichen Namen. Findet er dort keine passenden Symbole, dann sucht er nach neuen Modulen mit solchen CSECTs oder ENTRYs und fügt diese Module in die Ladeeinheit ein. Diesen Ladevorgang bezeichnet man als **Autolink-Funktion** des DBL. Die Autolink-Funktion behandelt nur CSECTs und ENTRYs, die nicht maskiert sind (siehe Handbuch „BINDER“ [1]).

Suchstrategie

Der DBL sucht die Module, die Externverweise befriedigen, in verschiedenen Containern. Wird ein Modul mit passenden CSECTs oder ENTRYs in einem Container gefunden, wird das Modul in die Ladeeinheit eingefügt und der Suchvorgang beendet. Der Suchvorgang verläuft standardmäßig in folgenden Stufen, wobei die Reihenfolge der Stufen 1 bis 4 vom Benutzer verändert werden kann (siehe Hinweis „Benutzerdefinierte Suchreihenfolge“ auf Seite 27):

1. Suche im Link-Kontext (siehe Seite 56).
2. Suche im Shared Code des Benutzers, der mit dem ASHARE-Makro geladen wurde.
   - Die Suche im Shared Code des Benutzers wird nur im RUN-MODE=*ADVANCED ausgeführt.
   - In Common Memory Pools wird nur dann gesucht, wenn der Benutzer SHARE-SCOPE=*MEMORY-POOL(...)/*ALL angegeben hat, da der Default-Wert für SHARE-SCOPE=*SYSTEM-MEMORY ist. Die Suche kann auf Common Memory Pools mit einem definierten Geltungsbereich eingeschränkt oder ganz unterdrückt werden (SHARE-SCOPE=*NONE).
3. Suche im Shared Code des Systems, der aus den unprivilegierten Subsystemen im Klasse-3/4/5-Speicher besteht (siehe Handbuch „Verwaltung von Subsystemen“ [10]). Wird ein Externverweis durch ein Symbol eines Subsystems befriedigt, dann baut der DBL eine Verbindung zu diesem Subsystem auf. Das Durchsuchen der unprivilegierten Subsysteme kann der Benutzer im Ladeaufruf unterdrücken (Operand SHARE[-SCOPE]=*NONE).
4. Suche in Referenz-Kontexten (siehe Seite 56), die der Benutzer festgelegt hat. Bei mehreren Referenz-Kontexten erfolgt die Suche in der vorliegenden Reihenfolge. Gibt es keine Referenz-Kontexte, wird dieser Schritt übergangen. Außerdem ist er nur für den Makro BIND möglich.

5. Durchsuchen der Bibliothek, die der Benutzer im Ladeaufruf angegeben hat (Operand LIBRARY bzw. LIBNAM@/LIBNAM/LIBLINK). Wenn im RUN-MODE= *ADVANCED eine solche Bibliothek nicht angegeben wurde, aber der Link-Name BLSLIB einer Bibliothek zugewiesen wurde, dann verwendet der DBL diese Standardbibliothek mit dem Link-Namen BLSLIB. Ist die angegebene Bibliothek fehlerhaft oder existiert sie nicht, so wird die Verarbeitung mit einer Fehlermeldung abgebrochen.
   Diese Suche wird nicht durchgeführt, wenn AUTOLINK=ALTLIB angegeben wurde.
6. Durchsuchen von **alternativen Bibliotheken**.

   Es gibt zwei Gruppen von alternativen Bibliotheken:

   a) Alternative Bibliotheken, die über Dateikettungsnamen vereinbart wurden.

      Hierbei unterscheidet man zwischen

      – alternativen Benutzerbibliotheken mit den Dateikettungsnamen BLSLIBnn und
      – alternativen Systembibliotheken mit den Dateikettungsnamen $BLSLBnn
        Alternative Systembibliotheken werden für Komponenten des von Fujitsu ausgelieferten Laufzeitsystems benötigt. Sie werden über den Dateikettungsnamen $BLSLBnn (00≤nn≤99) von der jeweiligen Laufzeitkomponente selbst zugewiesen. Der Benutzer hat darauf keinen Einfluss. Die Dateikettungsnamen der alternativen Systembibliotheken werden von der Software-Entwicklung der Fujitsu Technology Solutions GmbH verwaltet und sind ausschließlich für deren Softwareprodukte reserviert.

   b) System- und Benutzer-Tasklibs

      Hierbei handelt es sich um Bibliotheken mit dem Namen TASKLIB oder um eine Bibliothek, die mit dem Kommando SET-TASKLIB zugewiesen wurde.

   Über Dateikettungsnamen vereinbarte alternative Bibliotheken werden in folgender Reihenfolge durchsucht:

   1. Alternative Systembibliotheken mit den Dateikettungsnamen $BLSLB00 .. 49
   2. Alternative Bibliotheken, die vom Benutzer mit dem Dateikettungsnamen BLSLIBnn (00≤nn≤99) zugewiesen wurden.
   3. Alternative Systembibliotheken mit den Dateikettungsnamen $BLSLB50 .. 99

   Innerhalb einer Gruppe von Bibliotheken werden diese nach aufsteigenden Nummern „nn“ durchsucht.

System- und Benutzertasklibs werden in folgender Reihenfolge durchsucht:

1. Die Bibliothek, die vom Benutzer mit dem Kommando SET-TASKLIB zugewiesen wurde
2. Die Bibliothek „$userid.TASKLIB“
   oder, falls diese nicht existiert:
   Die Bibliothek „$defluid.TASKLIB“.
   Dabei ist „defluid“ der Name der System-Standardkennung (Wert des Klasse-2-Systemparameters DEFLUID, siehe Handbuch „Einführung in die Systembetreuung“ [9]).

Welche alternativen Bibliotheken durchsucht werden, hängt davon ab, auf welche Weise der Ladevorgang durchgeführt wurde:

– Laden mit START/LOAD-EXECUTABLE-PROGRAM
  oder mit dem BIND-Makro (mit INTVERS=SRVxxx, wobei xxx ≥ 002)

  Der Benutzer kann mit dem Operanden ALTERNATE-LIBRARIES bzw. ALTLIB festlegen, welche der genannten Bibliotheken in welcher Reihenfolge durchsucht werden.

– Laden mit START/LOAD-PROGRAM und RUN-MODE=*ADVANCED
  oder mit dem BIND-Makro (INTVERS=BLSP2/SRV001)

  Nur die mit den Dateikettungsnamen BLSLIBnn bzw. $BLSLBnn vereinbarten alternativen Bibliotheken werden durchsucht, falls ALTERNATE-LIBRARIES=*YES angegeben wurde.

– Laden mit START/LOAD-PROGRAM und RUN-MODE=*STD

  Nur die System- und Benutzer-Tasklibs werden durchsucht.

Die Stufen 5 und 6 können im Ladeaufruf mit START/LOAD-EXECUTABLE-PROGRAM oder mit dem BIND-Makro unterdrückt werden, indem der Operand AUTOLINK=*NO angegeben wird. In diesem Fall durchsucht der DBL nur bereits geladenen privaten und gemeinsam benutzbaren Code. Damit kann das unbeabsichtigte Laden von Laufzeitmodulen verhindert werden. Falls einige Externverweise ungelöst bleiben, behandelt sie der DBL so, wie es auf Seite 27 beschrieben ist.

Treten Namenskonflikte auf, behandelt sie der DBL wie auf den Seiten 32ff beschrieben.

Die Autolink-Funktion bezieht sich *nicht* auf die temporäre EAM-Bindemoduldatei. Sollen mehrere Module aus der EAM-Bindemoduldatei geladen werden (Operand *OMF im Ladeaufruf), muss der Benutzer darauf achten, dass das erste Modul in der EAM-Bindemoduldatei die Startadresse der Ladeeinheit enthält. Im Ladeaufruf muss dabei angegeben werden, dass alle Module aus der EAM-Bindemoduldatei geholt werden sollen (Operand FROM-FILE=*MODULE(LIBRARY=*OMF,ELEMENT=*ALL,...)).

Aufgrund bedingter Externverweise (WXTRNs) wird kein Nachladen ausgelöst. Sie werden nur befriedigt:

– in einem unprivilegierten Subsystem oder im Shared Code,

– durch Module, die z.B. mit der INCLUDE-Anweisung geholt werden,

– durch die Autolink-Funktion, die durch andere EXTRNs oder V-Konstanten aufgerufen wird.

Ein COMMON-Bereich wird durch die Autolink-Funktion nicht gebunden. Das bedeutet, wenn sich ein Externverweis auf einen COMMON-Bereich bezieht, kann das Modul, das diesen COMMON-Bereich enthält, nicht automatisch geladen werden. Die Externverweise werden nicht befriedigt.

Benutzerdefinierte Suchreihenfolge

Die Suchreihenfolge innerhalb der Stufen 1 bis 4 kann der Benutzer seinen Bedürfnissen entsprechend anpassen. Hierzu stehen folgende Operanden zur Verfügung:

– Beim Kommando MODIFY-DBL-DEFAULTS:

  RESOL-TYPE=*USER(ORDER=...) und
  PUBLIC-RESOL-TYPE=*USER(ORDER=...)

– Beim Makro BIND:

  RESORD (mit RESTYP=USER) und PURESOR (mit PURESTY=USER)

Behandlung der unbefriedigten Externverweise

Die Behandlung von Externverweisen, die beim Durchsuchen der alternativen Bibliotheken (siehe 6. auf Seite 25) nicht befriedigt wurden, kann der Benutzer mit dem Operanden UNRESOLVED-EXTRNS im Ladeaufruf steuern.

Folgende Möglichkeiten gibt es:

– Nicht befriedigte Externverweise sind unzulässig
  (Operand UNRESOLVED-EXTRNS=*ABORT)
  Das Laden der aktuellen Ladeeinheit wird dann abgebrochen.

– nicht befriedigte Externverweise erhalten eine vom Benutzer festgelegte Adresse
  (Operand UNRESOLVED-EXTRNS=*STD)
  Die Adresse wird vom Benutzer im Operanden ERROR-EXIT im Ladeaufruf angegeben. Standardwert ist die Adresse X’FFFFFFFF’.

– nicht befriedigte Externverweise werden zu einem späteren Zeitpunkt befriedigt (Operand UNRESOLVED-EXTRNS=*DELAY).
Der DBL speichert die nicht befriedigten Externverweise im Link-Kontext (siehe Seite 56). Wird eine neue Ladeeinheit im Kontext geladen, versucht der DBL am Ende des Ladens, die gespeicherten Externverweise mit CSECTs und ENTRYs der neuen Ladeeinheit zu befriedigen. Dieser Vorgang wiederholt sich beim Laden weiterer Ladeeinheiten, solange der Kontext besteht. Diese Möglichkeit gilt nicht für externe Pseudoabschnitte (XDSECs-R). Externverweise, die zu einem späteren Zeitpunkt befriedigt werden sollen, erhalten beim Speichern im Kontext eine vorläufige Adresse, die vom Benutzer mit dem Operand ERROR-EXIT im Ladeaufruf festgelegt wird. Voreinstellung ist *NONE, was intern der Adresse X'FFFFFFFF' entspricht.

– Die Angabe UNRES=DELAYWARN beim BIND-Makro hat im Wesentlichen dieselbe Wirkung wie UNRES=DELAY. Die beiden Angaben unterscheiden sich lediglich im Returncode. Wenn Externverweise existieren, die zu einem späteren Zeitpunkt befriedigt werden müssen, wird bei DELAYWARN ein entsprechender Returncode zurückgegeben, während der Returncode bei DELAY Null ist. Außerdem ist die Angabe UNRES=DELAYWARN Voraussetzung für die Ausgabe von aus früheren Ladevorgängen nicht befriedigt verbliebenen Externverweise in den benutzerdefinierten Datenbereich.

Bei RUN-MODE=*STD wird die Verarbeitung entsprechend dem Operanden UNRESOLVED-EXTRNS=*STD durchgeführt und alle unbefriedigten Externverweise erhalten die Adresse X'FFFFFFFF'.

Alle nicht befriedigten Externverweise protokolliert der DBL in die Systemdatei SYSOUT. Im Dialogbetrieb mit UNRESOLVED-EXTERNS=*STD kann der Benutzer dann entscheiden, ob bei unbefriedigten Externverweisen die Verarbeitung fortgesetzt oder abgebrochen werden soll.
Im Stapelbetrieb wird die Verarbeitung immer fortgesetzt.

Nicht befriedigte Pseudoabschnitte (XDSECs-R) werden dabei getrennt von den übrigen Externverweisen aufgelistet. XDSEC-R werden nur mit XDSEC-D im gleichen Kontext befriedigt.

Der Operand USNUNR@ des BIND-Makros (nur zusammen mit INTVER=SRVxxx, wobei xxx ≥ 005) ermöglicht die Ausgabe einer Liste unbefriedigter Externverweise in einen benutzerdefinierten Datenbereich. Dieser Operand und der Operand UNRES sind voneinander vollkommen unabhängig. Näheres hierzu finden Sie in der Beschreibung des BIND-Makros im Handbuch „Makroaufrufe an den Ablaufteil“ [7]. Bei Angabe von INTVERS=SRVxxx (xxx ≥ 006) und UNRES=DELAYWARN im BIND-Makro kann alternativ oder zusätzlich die Liste der aus früheren Ladevorgängen nicht befriedigt verbliebenen Externverweise in den benutzerdefinierter Datenbereich ausgegeben werden.

Behandlung von COMMON-Bereichen

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 der Ladeeinheit als Daten-Kommunikationsbereiche zwischen verschiedenen Modulen verwendet oder als Platz für CSECTs eingesetzt werden.

COMMON-Bereichen, die in verschiedenen Modulen der Ladeeinheit definiert sind, und alle den gleichen Namen haben, weist DBL *einen* gemeinschaftlichen Speicherbereich zu. Diesen Bereich wählt er so groß, dass der längste COMMON-Bereich dieses Namens hineinpasst. Es ist nicht möglich, diesen Speicherbereich während des Programmlaufes durch einen BIND-Makroaufruf zu vergrößern. Gibt es keinen Programmabschnitt (CSECT) mit dem Namen des COMMON-Bereichs, so verzögert der DBL die Bearbeitung, bis alle Module geladen sind. Hat eine CSECT den gleichen Namen wie COMMON-Bereiche, gibt der DBL dieser CSECT die Adresse des COMMON-Bereichs, d.h. der COMMON-Bereich wird beim Laden der Ladeeinheit mit dieser CSECT initialisiert. Beim Initialisieren werden außerdem Attribute und Inhalt des COMMON-Bereiches mit denen der CSECT in Übereinstimmung gebracht.

Wenn ein Programmabschnitt und ein COMMON-Bereich den gleichen Namen haben, berücksichtigt der DBL die COMMON-Anweisung bereits bei der Bearbeitung des betreffenden Programmabschnitts.

– Erscheint in der Eingabe der COMMON-Bereich vor dem gleichnamigen Programmabschnitt, gibt der DBL dem COMMON-Bereich die größere der beiden Längen.

– Befindet sich bei der Eingabe der Programmabschnitt vor dem gleichnamigen COMMON-Bereich, so wird dieser vom DBL so groß wie der Programmabschnitt gewählt, unabhängig von der Länge, die in der COMMON-Definition angegeben ist.

Für COMMON-Bereiche ist die Autolink-Funktion nicht wirksam. Deshalb wird ein Modul mit einem COMMON-Bereich, auf den ein EXTRN oder VCON verweist, nicht automatisch nachgeladen.

Namenskonflikte behandelt DBL wie auf den Seiten 32ff beschrieben.

Relativierung der Adressen

Die wichtigste Aufgabe des dynamischen Bindeladers DBL besteht darin, aus den Bindemodulen (OMs) und LLMs der Eingabe eine ablauffähige Einheit zu bilden. Er muss also die Adressen in den einzelnen Programmabschnitten dem Gesamtprogramm anpassen (Relativierung) und außerdem für noch unbefriedigte Externverweise und V-Konstanten die passenden Einsprungadressen suchen.

Ein Bindemodul (OM) und ein LLM enthalten Relativierungssätze, in denen festgelegt ist, wie sich die Programmadressen relativ zueinander verhalten. Mit ihrer Hilfe vergibt der DBL die endgültigen Adressen für das gesamte Programm.

Relativierung der Adressen bei Bindemodulen (OMs)

Bei Programmabschnitten (CSECT) hängt die Anfangsadresse von den beiden Attributen READ-ONLY und PAGE ab, die vom Sprachübersetzer eingetragen wurden:

READ-ONLY bedeutet, dass der Abschnitt während des Programmlaufs nur gelesen, d.h. nicht überschrieben werden darf. Der DBL muss den Abschnitt also in eine neue Seite laden, falls die aktuelle Seite nicht auch das Attribut READ-ONLY hat.

PAGE vereinbart, dass der Programmabschnitt an den Anfang einer neuen Seite zu laden ist, d.h. an eine Adresse, die ein Vielfaches von 4096 Byte (X'1000') ist.

Beispiele für die Verarbeitung von Programmabschnitten durch den DBL

1.
```
X   CSECT   PAGE,READ
.....................
Y   CSECT   READ
.....................
```

   Programmabschnitt X soll auf Seitengrenze ausgerichtet und nur lesbar sein. Der nachfolgende Abschnitt Y benötigt keine Ausrichtung auf Seitengrenze, ist ebenfalls nur lesbar. Der DBL lädt Y deshalb in dieselbe Seite wie X, und zwar an die nächste Doppelwortgrenze.

2.
```
ABC   CSECT   READ
..................
XYZ   CSECT
..................
```

   Den Programmabschnitt ABC richtet der DBL, falls er der erste Programmabschnitt des Bindemoduls ist, automatisch auf Seitengrenze aus. Er lädt ihn in eine Seite, die nur gelesen werden darf. Abschnitt XYZ wird von DBL auf die Seitengrenze einer neuen Seite ausgerichtet, weil Schreib- und Lesezugriff auf die CSECT erlaubt ist.

Relativierung der Adressen bei LLMs

Die Anfangsadresse ist von folgenden Eigenschaften des LLM abhängig:

1. LLM mit CSECT-Attributen PAGE und READ-ONLY

   Wurden die Attribute PAGE und READ-ONLY vom BINDER übergeben, dann gilt:

   – PAGE vereinbart, dass die CSECT an den Anfang einer neuen Seite zu laden ist, d.h. an einer Adresse, die ein Vielfaches von 4096 Byte (X'1000') ist.

   – READ-ONLY bedeutet, dass die CSECT während des Programmlaufs nur gelesen, d.h. nicht überschrieben werden darf. Der DBL muss die CSECT also in eine neue Seite laden, falls die aktuelle Seite nicht auch das Attribut READ-ONLY hat.
   Dies gilt für LLMs mit folgender physischer Struktur (siehe Handbuch „BINDER“ [1]):
     – Einzel-Slices
     – benutzerdefinierte Slices
     – nach Attributen gebildete Slices, bei denen das Attribut READ-ONLY nicht zur Bildung von Slices verwendet wurde.

2. LLM ohne Relativierungsinformation (LRLD)

   Wenn der LLM ohne Relativierungsinformation gespeichert wurde (siehe BINDER-Anweisung SAVE-LLM, Operand RELOCATION-DATA=*NO), bestimmt der DBL die Anfangsadresse, die mit dem Operanden LOAD-ADDRESS in der BINDER-Anweisung SAVE-LLM festgelegt wurde. Ist dies nicht möglich, wird der Ladevorgang abgebrochen.

3. LLM mit Relativierungsinformation (LRLD)

   Wenn der LLM mit Relativierungsinformation gespeichert wurde (siehe BINDER-Anweisung SAVE-LLM, Operand RELOCATION-DATA=*YES), bestimmt der DBL die Anfangsadresse wie folgt:

   – Er versucht zuerst, die mit dem Operanden LOAD-ADDRESS in der BINDER-Anweisung SAVE-LLM festgelegte Anfangsadresse zuzuordnen. Ist dies möglich, wird die Relativierung auf die unbefriedigten Externverweise und auf die Referenzen zu COMMON-Bereichen beschränkt. (Die LLM-interne Relativierung wurde schon während des BINDER-Laufs durchgeführt.)

   – Kann die mit dem Operanden LOAD-ADDRESS festgelegte Adresse nicht als Anfangsadresse zugeordnet werden, bestimmt der DBL eine beliebige Adresse und führt die vollständige Relativierung durch.

Unterstützung von EEN-Namen (extended external names)

BLSSERV kann Module mit EEN-Namen verarbeiten.

Compiler für objektorientierte Programmiersprachen erzeugen Symbolnamen, die nicht den Konventionen für EN-Namen (External Names) entsprechen. Solche Namen werden EEN-Namen (Extended External Names) genannt.

Symbole mit EEN-Namen können nur in LLMs mit Format 4 (oder höher) enthalten sein.

– Diese LLMs können von BINDER ab V2.0 erzeugt werden.

– Sie können mit BLSSERV ab V2.0 in den Benutzeradressraum geladen werden.

– Mit BLSSERV ab V2.1 können sie auch als Shared Code oder als Subsystem geladen werden.

– Die Eingabe von EEN-Namen über Benutzerschnittstellen zum Binden und Laden (START/LOAD-EXECUTABLE-PROGRAM, START/LOAD-PROGRAM, BIND, ASHARE) ist nicht möglich. Die Module müssen über den Namen des Bibliothekselements spezifiziert werden, in dem sie enthalten sind.

– Informationen über EENs können mit der VSVI1-Schnittstelle ausgegeben werden. In allen anderen Ausgaben des BLS werden keine EENs angezeigt, sie werden durch interne Namen ersetzt.

Behandlung von Namenskonflikten

Namenskonflikte können entstehen, wenn im Externadressbuch einer Ladeeinheit Symbole mit gleichen Namen vorkommen. Symbole sind CSECTs, ENTRYs, COMMON-Bereiche und XDSECs-D. Nicht jede Namensgleichheit ist ein Namenskonflikt.

Der DBL behandelt Namenskonflikte im Betriebsmodus RUN-MODE=*STD und RUN-MODE=*ADVANCED (siehe Seite 91) unterschiedlich.

Behandlung von Namenskonflikten bei RUN-MODE=*STD

Bei RUN-MODE=*STD werden Namenskonflikte entdeckt, unabhängig davon, ob die Symbole in einem Modul maskiert sind oder nicht. Der Benutzer hat keine Möglichkeit, die Behandlung von Namenskonflikten zu steuern.

In der folgenden Tabelle ist dargestellt, wie der DBL Namenskonflikte behandelt. Dabei bedeutet „Eintrag 1“ ein Symbol in einem Modul, das bereits geladen ist, oder ein COMMON-Bereich, der in der aktuellen Ladeeinheit bereits bekannt ist. Ein nicht initialisierter COMMON-Bereich am Ende der Ladeeinheit wird in der Symboltabelle vermerkt und bei weiteren Ladeaufrufen als CSECT betrachtet. Ein Namenskonflikt tritt auf, wenn ein Modul geladen wird, in dem ein Symbol „Eintrag 2“ mit gleichem Namen gefunden wird.

	Eintrag 2			
Eintrag 1	**CSECT**	**ENTRY**	**COMMON**	**XDSEC-D**
CSECT	(1)	–	(2)	–
ENTRY	–	–	(3)	–
COMMON	(3)	(4)	(5)	–
XDSEC-D	–	–	–	(6)

Bedeutung

(1) Ein Namenskonflikt wurde entdeckt.
Das Laden des Moduls, das die CSECT mit dem gleichen Namen enthält, wird abgebrochen.
Die Autolink-Funktion wird abgebrochen. Wurde das Laden des Moduls durch eine INCLUDE-Anweisung in einem Bindemodul verursacht (siehe Anhang), so wird das Modul ignoriert und die Verarbeitung wird fortgesetzt.

(2) Der COMMON-Bereich wird mit der CSECT initialisiert.

(3) Ein Namenskonflikt wurde entdeckt.
Das Laden wird abgebrochen.

(4) Der Namenskonflikt wird **nicht** entdeckt.

(5) Der zweite COMMON-Bereich wird ignoriert.

(6) Behebbarer Fehler
Die zweite XDSEC-D wird übergangen.

(-) kein Namenskonflikt (nur Namensgleichheit).

Behandlung von Namenskonflikten bei RUN-MODE=*ADVANCED

Bei RUN-MODE=*ADVANCED werden Namenskonflikte nur entdeckt, wenn die Symbole in einem Modul *nicht* maskiert sind. Der Benutzer wird über jeden Namenskonflikt, bei dem Fehler auftreten können, durch eine Meldung benachrichtigt. Mit dem Operanden NAME-COLLISION im Ladeaufruf hat der Benutzer die Möglichkeit, die Behandlung von Namenskonflikten zu steuern. Folgende Möglichkeiten sind vorhanden:

– Operand NAME-COLLISION=*STD
  Namenskonflikte zwischen nicht maskierten Symbolen werden durch Warnungsmeldungen angezeigt. Das Modul, das das Symbol mit dem gleichen Namen enthält, wird geladen. Die neue Ausprägung des Symbols wird maskiert, d.h. es wird nicht mehr benutzt, um Externverweise zu befriedigen.

– Operand NAME-COLLISION=*ABORT
  Das Laden der aktuellen Ladeeinheit wird abgebrochen, sobald ein Namenskonflikt zwischen nicht maskierten Symbolen entdeckt wird.

In der folgenden Tabelle ist dargestellt, wie der DBL Namenskonflikte bei NAME-COLLISION=*STD behandelt. Dabei bezeichnet „Eintrag 1“ ein nicht maskiertes Symbol in einem Modul, das bereits geladen ist, oder ein COMMON-Bereich, der in der aktuellen Ladeeinheit bereits bekannt ist. Ein nicht initialisierter COMMON-Bereich am Ende der Ladeeinheit wird in der Symboltabelle vermerkt und bei weiteren Ladeaufrufen als CSECT betrachtet. Ein Namenskonflikt tritt auf, wenn ein Modul geladen wird, in dem ein nicht maskiertes Symbol „Eintrag 2“ mit gleichen Namen gefunden wird.

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

Bedeutung

(1) Ein Namenskonflikt wurde entdeckt. Das Symbol „Eintrag 1“ wird nicht maskiert und kann für die Befriedigung von Externverweisen verwendet werden. Das Symbol „Eintrag 2“ wird maskiert und kann nicht mehr für die Befriedigung von Externverweisen verwendet werden. Eine Warnungsmeldung wird ausgegeben.

(2) Der COMMON-Bereich wird mit der CSECT initialisiert (wie bei RUN-MODE=*STD).

(3) Der zweite COMMON-Bereich wird ignoriert (wie bei RUN-MODE=*STD).

(4) Behebbarer Fehler
Die zweite XDSEC-D wird übergangen. Die Behandlung ist die gleiche wie bei RUN-MODE=*STD.

(-) kein Namenskonflikt (nur Namensgleichheit).

Indirektes Binden

Mit DBL ab BS2000/OSD-BC V3.0 wird ein neuer Mechanismus eingeführt, der im weiteren als „Indirektes Binden“ bezeichnet wird. Beim indirekten Binden (Indirect Linkage) wird ein Externverweis nicht wie bisher direkt mit einer Programmdefinition befriedigt, sondern über eine zwischengeschaltete Indirect-Linkage-Routine. In dieser IL-Routine erfolgt der Aufruf eines Servermoduls, das die gesuchte Programmdefinition enthält.

Das indirekte Binden hat folgende Vorteile:

– Beim Austausch des Servermoduls wird nur zwischen der IL-Routine und dem Server entbunden. Beim Einsetzen des neuen Servermoduls für dieselbe Programmdefinition muss nur der Externverweis zwischen der IL-Routine und dem Servermodul befriedigt werden. Das vermeidet aufwändige konsistenzerhaltende Maßnahmen zwischen dem Server und seinen möglichen Aufrufern.

– Die Befriedigung von Externverweisen wird nur einmal durchgeführt und ändert sich nicht, solange der Aufrufer geladen ist. Dadurch wird das Entbinden über mehrere Kontexte oder Tasks hinweg vermieden und die Befriedigung von Externverweisen muss nicht bei jedem Aufruf des Servers erneut durchgeführt werden.

Symbole des Typs ILE

Für das indirekte Binden wurde der neue Symboltyp ILE (Indirect Linkage Entry) eingeführt. Ein ILE hat folgende Attribute:

- Name:
  entspricht dem Namen des ILE-Servers
- Adresse der IL-Routine
- Adresse des ILE-Servers:
  ein Feld, das die ILE-Serveradresse enthält
- Distanz der Serveradresse in der IL-Routine:
  lokalisiert das Feld mit der ILE-Serveradresse innerhalb der IL-Routine
- Status des Servers (aktiv oder nicht aktiv):
  zeigt an, ob der ILE-Server zu diesem Zeitpunkt aufgerufen werden kann.
  Der Status kann sowohl vom aufrufenden Programm als auch von einer benutzerdefinierten IL-Routine verwendet werden.
- Steuerung (durch das System oder durch den Benutzer)

Die Deklaration von ILEs und die Verwaltung von ILE-Listen werden durch die Makroaufrufe ILEMGT und ILEMIT ermöglicht. Informationen über ILEs können mit dem Makroaufruf VSVI1 angefordert werden.

In Bild 3 ist dargestellt, wie Externverweise befriedigt werden, wenn der ILE-Server geladen ist.

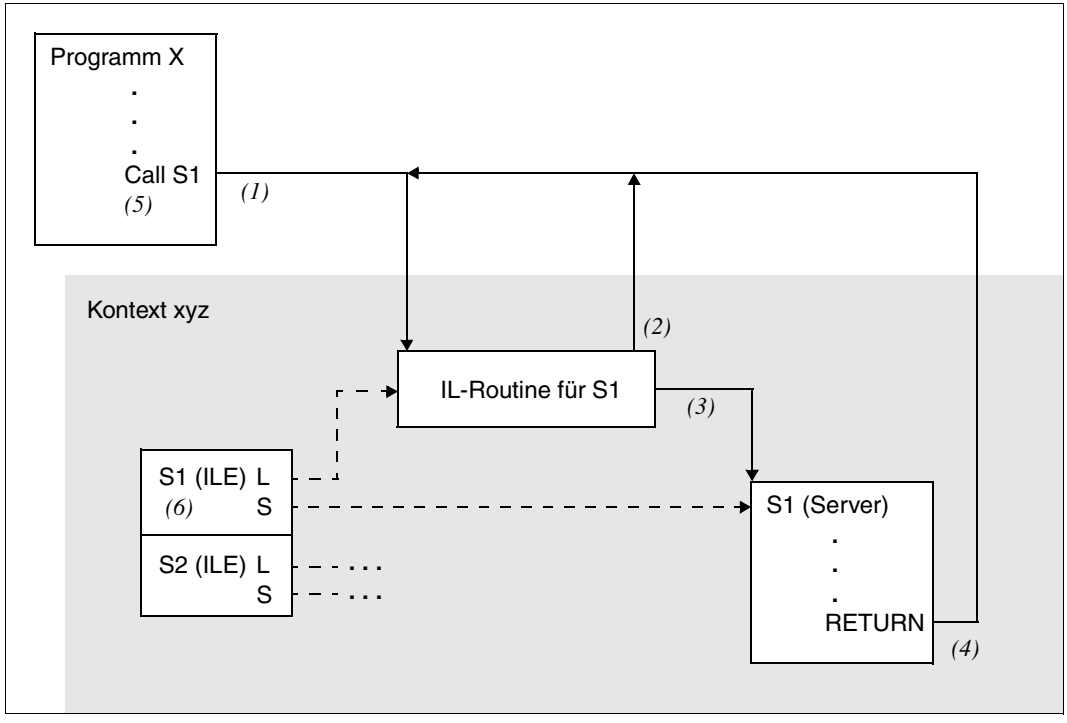


Bild 3: Indirektes Binden

(1) Der Aufruf von S1 verzweigt zur IL-Routine für S1.

(2) Wenn das Servermodul S1 nicht geladen ist, erfolgt der Rücksprung zum aufrufenden Programm X bereits in der IL-Routine.

(3) Wenn das Servermodul S1 geladen ist, verzweigt die IL-Routine zum Servermodul S1.

(4) Ist das Servermodul S1 abgearbeitet, erfolgt der Rücksprung zum aufrufenden Programm X.

(5) Der Aufrufer kann den Returncode überprüfen, um Informationen über die Abarbeitung des Servermoduls zu erhalten.

(6) Die Symboltabelle des Kontextes xyz enthält den Eintrag für ein ILE-Symbol mit dem Namen S1, in dem die Adressen der IL-Routine (L) und des Servermoduls (S) abgelegt sind.

Verwaltung von List-Name-Units

Ab BLSSERV V2.4A können mehrere oder alle Objekte einer Bibliothek mit einem einzigen Aufruf des BIND-Makros geladen werden. Dadurch entsteht eine so genannte List-Name-Unit und es werden mehrfache DBL-Aufrufe beim Laden einer Symbol-Liste derselben Ladeeinheit vermieden.

Zum Laden in eine List-Name-Unit sind die Angabe `INTVERS=SRVxxx` (`xxx` ≥ 004) und `SYMBOL=*ALL` oder `SYMBOL=name` erforderlich, wobei das letzte Zeichen von `name` das Wildcard-Symbol „*“ ist. Näheres siehe Makro BIND im Handbuch „Makroaufrufe an den Ablaufteil“ [7].

Das Laden in eine List-Name-Unit läuft folgendermaßen ab:

- Abhängig vom Operanden SYMBOL wird eine Liste der aus der Hauptbibliothek zu ladenden Objekte erstellt:
  - Bei `SYMBOL=*ALL` wird zunächst die jeweils höchste Version aller Elemente des Typs L in diese Liste aufgenommen. Anschließend wird die jeweils höchste Version aller Elemente des Typs R in die Liste aufgenommen, jedoch nur dann, wenn nicht bereits ein gleichnamiges Element des Typs L in der Liste enthalten ist.
  - Ist der Operand SYMBOL mit einem teilqualifizierten Name angegeben, so wird die Liste ebenso erzeugt. Allerdings werden dabei nur die Elemente berücksichtigt, die dem teilqualifizierten Namen entsprechen.
  - Die Namensliste enthält ausschließlich Namen von PLAM-Bibliothekselementen (d.h. SYMTYP=MODULE). Es werden keine gleichnamigen CSECTs oder ENTRYs in der Bibliothek gesucht.

  Dabei gelten folgende Voraussetzungen:
  - Die Hauptbibliothek muss eine PLAM-Bibliothek sein.
  - Alternative Bibliotheken werden nur für den Autolink-Prozess verwendet, nicht jedoch bei der Erstellung der Namensliste.
  - LLM mit benutzerdefinierten Slices können nicht in eine List-Name-Unit geladen werden. Der Ladevorgangs wird mit RC=X'0C400430' abgebrochen.
  - Die Operanden VERS und VERS@ werden beim Laden in eine List-Name-Unit ignoriert.
  - Vor dem Laden findet keine Suche nach Namen im bereits geladenen Code statt, da es sich bei den Namen um Elementnamen und nicht um Symblonamen handelt. Im geladenen Code sind jedoch nur Symbolnamen hinterlegt.

- Alle Objekte in der Liste werden in derselben Einheit (List-Name-Unit) und in dem Kontext geladen, der durch UNIT oder UNIT@ und LNKCTX oder LNKCTX@ festgelegt ist. Ist kein Name für die Ladeeinheit angegeben, wird der Name des ersten Objekts in der Liste als Name für die Ladeeinheit angenommen.
- Die Version einer List-Name-Unit wird mit dem Operanden PGMVERS/PGMVERS@ bestimmt. Bei `PGMVERS=*STD` wird die PLAM-Version des ersten geladenen Objekts als Version der Ladeeinheit eingesetzt.
- In der DBL-Liste wird für jedes in der Liste enthaltene Objekt als Binde-Verfahren EXPLICIT angegeben.
- Sowohl die zurückgelieferte Startadresse (Operand SYMBLAD bei BIND) als auch der zurückgemeldete AMODE entsprechen denen des ersten Objekts, das in die List-Name-Unit geladen wird. Ebenso wird bei Angabe von `BRANCH=YES` die Steuerung an die Startadresse des ersten Objekts übergeben, das in die List-Name-Unit geladen wurde.
- Beim Laden in eine List-Name-Unit werden Externverweise erst befriedigt, wenn alle Objekte der List-Name-Unit geladen sind. Auch die Autolink-Funktion wird erst zu diesem Zeitpunkt ausgeführt. Dadurch erhöht sich die Performance beim Befriedigen von Externverweisen: Ein Symbol, das in mehreren Modulen der Liste referenziert wird, muss nur einmal gesucht werden und nicht nach dem Laden jedes einzelnen Objekts.

  Beispiel

  Drei Objekte einer Liste sollen geladen werden:
  – Modul M1 mit Externverweisen E1 und E2
  – Modul M2 mit Externverweisen E2 und E3
  – Modul M3 mit ENTRY E1 und Externverweis E2.

  a) Ladevorgang beim Laden **ohne** List-Name-Unit:
     – M1 laden
     – Externverweise E1 und E2 befriedigen und Autolink (E1 in M3 gefunden)
     – M3 laden (wegen Autolink)
     – Externverweis E2 befriedigen und Autolink (ohne Wirkung)
     – M2 laden
     – Externverweis E2 und E3 befriedigen und Autolink (ohne Wirkung).

     Für den Externverweis E2 wird der Prozess der Adressbefriedigung dreimal durchlaufen.

b) Ladevorgang beim Laden **in** List-Name-Unit:
   - M1 laden
   - M2 laden
   - M3 laden
   - Externverweise E1, E2 und E3 und Autolink (ohne Wirkung)

   Für den Externverweis E2 wird der Prozess der Adressbefriedigung nur einmal durchlaufen.

Das Laden in eine List-Name-Unit ist somit performanter, kann jedoch die Ladereihenfolge beeinflussen (Modul M3 wird ohne List-Name-Unit vor M2 geladen).

Bei Laden von PUBLIC/PRIVATE LLMs in List-Name-Units wird der Public-Teil erst geladen, nachdem alle Objekte der List-Name-Unit geladen sind.

Für das Entladen von List-Name-Units gelten folgende Einschränkungen:

- Eine List-Name-Unit kann nur als Ganzes entladen werden. Das Entladen eines Teils oder nur eines einzelnen Moduls aus einer List-Name-Unit ist nicht möglich.
- Falls bei einem der Objekte in der Liste Fehler auftreten, werden alle bereits geladenen Module der Liste (=aktueller Stand der List-Name-Unit) entladen. Dies entspricht dem derzeitigen Verhalten beim Laden einer Einheit.

Eigenschaften einer List-Name-Unit

Die Eigenschaften einer List-Name-Unit lassen sich wie folgt zusammenfassen:

- Die Eingabeobjekte werden nur in der Hauptbibliothek gesucht (alternative Bibliotheken werden nur beim Autolink verwendet).
- Eingabeobjekte werden grundsätzlich geladen (keine Suche im Speicher vor dem Laden).
- Das Befriedigen von Externverweisen und die Durchführung der Autolink-Funktion erfolgen erst dann, wenn alle Objekte der List-Name-Unit geladen wurden.
- Ein Modul einer List-Name-Unit kann nicht nicht unabhängig von der List-Name-Unit entladen werden.

Laden von LLMs aus Dateien (PAM-LLMs)

Um dem Benutzer die Umstellung von Anwendungen mit Lademodulen auf solche mit LLMs zu erleichtern, ermöglicht es BLSSERV, LLMs zu laden, die mit BINDER in eine PAM-Datei abgespeichert wurden. Diese LLMs werden im Gegensatz zu den in Programmbibliotheken abgespeichertern LLMs als PAM-LLMs bezeichnet.

PAM-LLMs unterscheiden sich von den herkömmlichen LLMs in folgender Weise:

- PAM-LLMs können nur mit Kommandos geladen werden, nicht jedoch mit Programmschnittstellen wie z.B. dem BIND-Makro. (Ausnahme: Slices eines PAM-LLMs mit benutzerdefinierten Slices können mit dem LDSLICE-Makro geladen werden.)
- Der Dateikettungsname für PAM-LLMs ist derselbe wie für Programmdateien: ECERDLOD.
- Da ein PAM-LLM im Ausführungsmodus ADVANCED abläuft, wird es standardmäßig in den Kontext LOCAL#DEFAULT geladen, nicht in den Kontext CTXPHASE, der für Lademodule reserviert ist.
- Der Name beim Ladeaufruf wird grundsätzlich als Dateiname und nicht als Symbolname interpretiert. Daher findet beim Laden keine Suche nach bereits geladenen Symbolen im Speicher statt.
- Beim Laden werden keine Externverweise befriedigt und es wird keine Autolink-Funktion durchgeführt.
- PUBLIC/PRIVATE-PAM-LLMs können nicht geladen werden.

2.1.3 Ausgaben des DBL

Ausgaben des DBL sind:

– eine Ladeeinheit, die in einem Kontext liegt und gestartet werden kann,

– eine DBL-Liste mit Information über die logische Struktur und den Inhalt der geladenen Ladeeinheit.

Ladeeinheit

Wenn der DBL mit einem Kommando START/LOAD-EXECUTABLE-PROGRAM bzw. START/LOAD-PROGRAM oder durch einen Makroaufruf BIND bzw. ASHARE aufgerufen wird, erzeugt er eine **Ladeeinheit** und lädt sie in den Hauptspeicher. Den Aufbau einer Ladeeinheit zeigt Bild 4.

Eine Ladeeinheit enthält alle Module, die mit einem einzigen Ladeaufruf geladen wurden. Dies sind alle Module, die im Ladeaufruf angegeben sind (Primäreingabe) und zusätzlich Module, die durch Autolink oder INCLUDE-Anweisungen eingefügt werden (Sekundäreingabe).

Die Module können Bindemodule (OMs) oder LLMs sein. Sie enthalten die Programmdefinitionen (CSECTs und ENTRYs). Jede Ladeeinheit liegt in einem Kontext, der vom Benutzer festgelegt wird (siehe Seite 53ff). Gibt der Benutzer keinen Kontext an, so wird der Standardkontext mit dem Namen „LOCAL#DEFAULT“ verwendet.

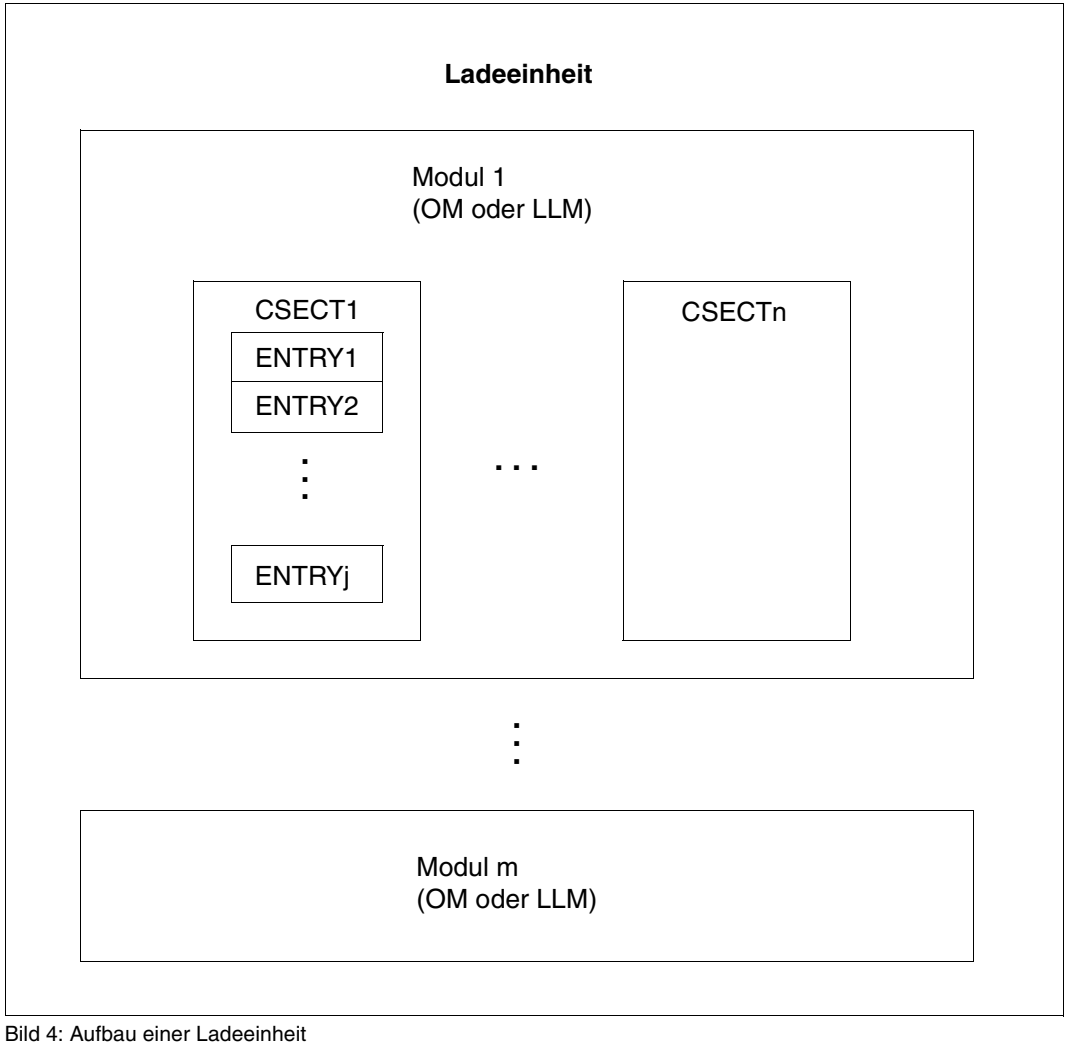


Bild 4: Aufbau einer Ladeeinheit

Ladeinformationen einer Ladeeinheit

Der Benutzer kann die Ladeinformation einer Ladeeinheit selbst im Ladeaufruf festlegen (Operanden LOAD-INFORMATION und TEST-OPTIONS). Er kann auswählen, ob ein Externadressbuch enthalten sein soll oder nicht. Bei LLMs können diese Informationen nur dann geladen werden, wenn sie vom BINDER erzeugt wurden (BINDER-Anweisung SAVE-LLM, Operand SYMBOL-DICTIONARY=*YES).
Bei Bindemodulen (OM) kann die Ladeinformation immer geladen werden. Bei Großmodulen (GM) kann die Ladeinformation mit dem Binder TSOSLNK reduziert werden. Soll in der Ladeeinheit ein Externadressbuch enthalten sein, können folgende Informationen für die Ladeeinheit ausgewählt werden:

- Externadressbuch mit Programmdefinitionen
  (Operand LOAD-INFORMATION=*DEFINITIONS)
  Ein Externadressbuch, das die Programmdefinitionen aller Module der Ladeeinheit enthält, wird geladen. Programmdefinitionen sind Programmabschnitte (CSECTs), Einsprungstellen (ENTRYs), COMMON-Bereiche, Pseudoabschnitte (DSECTs), externe Pseudoabschnitte (XDSECs-D) und Modulnamen. Diese Informationen benötigt der DBL, um
  - Externverweise zwischen Modulen zu befriedigen, die durch den aktuellen Ladeaufruf geladen werden, und Modulen, die durch einen späteren Ladeaufruf geladen werden,
  - Binde- und Startinformation an den Benutzer auszugeben,
  - Test- und Diagnosehilfen (AID-Kommandos) zu unterstützen.
- Externadressbuch mit Referenzen (Operand LOAD-INFORMATION=*REFERENCES)
  Ein Externadressbuch, das zusätzlich zu der Programmdefinition die befriedigten Referenzen aller Module der Ladeeinheit enthält, wird geladen. Referenzen sind Externverweise (EXTRNs), V-Konstanten, bedingte Externverweise (WXTRNs) und Externe Pseudoabschnitte (XDSECs-R).
  Diese Informationen benötigt der DBL, um Ladeeinheiten nach dem Laden zu entbinden und verzögerte Bearbeitung von Externverweisen durchzuführen.
- Externadressbuch für den Aufbau der DBL-Liste
  (Operand LOAD-INFORMATION=*MAP)
  Nur ein Externadressbuch, das für den Aufbau der DBL-Liste notwendig ist, wird temporär geladen. Das Externadressbuch wird entladen, sobald die DBL-Liste aufgebaut ist.
- LSD-Information (Operand TEST-OPTIONS=*AID)
  LSD-Information wird geladen. Diese Information benötigen die Test- und Diagnosehilfen (AID-Kommandos). LSD-Information ist nur sinnvoll, wenn Programmdefinitionen in der Ladeeinheit enthalten sind. Die LSD-Information kann nicht geladen werden, wenn der Operand LOAD-INFORMATION den Wert *NONE oder *MAP hat.

DBL-Liste

Eine DBL-Liste mit Informationen über den Aufbau und den Inhalt der geladenen Ladeeinheit wird mit dem Operanden PROGRAM-MAP im Ladeaufruf per Kommando oder mit dem Operanden MAP beim BIND-Makro ausgegeben. Die Ausgabe der DBL-Liste ist nur im Betriebsmodus ADVANCED möglich (siehe Seite 91ff). Als Ausgabeziel kann die Systemdatei SYSOUT, die Systemdatei SYSLST oder beide gewählt werden. Der BIND-Makro bietet zusätzlich die Möglichkeit der Ausgabe in einen benutzerdefinierten Datenbereich.

Die folgende Tabelle gibt für eine Ladeeinheit und die geladenen Module einen Überblick über die ausgegebenen Informationen.

Information	**Objekt**								
	Lade-einheit	**LLM**	**PAM-LLM**	**OM in LLM**	**OM**	**CSECT**	**ENTRY**	**COMMON**	**XDSEC-D**
Objektname	X	X	X	X	X	X	X	X	X
Name des Link-Kontext	X								
Ladeinformationen 1)	X								
Typ des LLM / PAM-LLM 2)		X	X						
Typ des OM 3)				X	X				
Objektadresse						X	X	X	
Objektgröße						X		X	X
Bindemethode 4)		X	X		X				
Dateikettungsname 5)		X	X		X				
Bibliotheks-/ Dateiname 6)		X	X		X				
Elementname		X							
Elementversion		X							

Fußnoten

1) Als Ladeinformationen werden ausgegeben:
   – Wert des Operanden LOAD-INFORMATION
   – Programmversion der Ladeeinheit. Wenn im Ladeaufruf keine Programmversion angegeben wurde, gibt der DBL die PLAM-Version des geladenen Bibliothekselements aus (~ steht für die höchste PLAM-Version).
   – Ladezeitpunkt

- Wert des Operanden TEST-OPTIONS
- Startadresse der Ladeeinheit (Load Unit Starting Point)
- Adressierungsmodus (AMODE):
  24, 31 oder 32
- Als Code-Typ wird bei HSI= ausgegeben:

/7500	Das Symbol aus dem Ladeaufruf verweist auf /390-Code.
/4000	Das Symbol aus dem Ladeaufruf verweist auf RISC(MIPS)-Code.
/SP04	Das Symbol aus dem Ladeaufruf verweist auf SPARC-Code.
/X86	Das Symbol aus dem Ladeaufruf verweist auf X86-Code.

2) Als Typ des LLM/PAM-LLM (siehe Handbuch „BINDER“ [1]) wird ausgegeben:

STANDARD	Das LLM besteht aus einer Einzel-Slice.
BY-USER	Das LLM besteht aus benutzerdefinierten Slices.
BY-ATTR	Das LLM besteht aus Slices, die nach Attributen gebildet sind.

3) Als Typ des OM wird ausgegeben:

STANDARD	Das OM wurde von einem Compiler erzeugt.
PRELINK	Das OM wurde in einem Binderlauf mit TSOSLNK als vorgebundenes Modul erzeugt.

4) Als Bindemethode wird ausgegeben:

EXPLICIT	Das Modul wurde als Primäreingabe durch explizite Angabe im Ladeaufruf geladen.
AUTOLINK	Das Modul wurde als Sekundäreingabe mit Hilfe der Autolink-Funktion geladen, um Externverweise zu befriedigen.
INCLUDE	Das Modul wurde als Sekundäreingabe durch INCLUDE-Anweisungen geladen, die in bereits geladenen Modulen eingefügt werden. Dies ist nur bei Bindemodulen (OMs) möglich.
PUBLIC	Ein LLM besteht aus Slices, die nach dem Attribut PUBLIC gebildet wurden (siehe Handbuch „BINDER“ [1]). Beim Laden des LLM wurden die Slices mit dem Attribut PUBLIC in den Klasse-6-Speicher geladen.

5) Der Dateikettungsname der Bibliothek, in der das LLM/OM gefunden wurde, wird ausgegeben. Handelt es sich dabei um die Benutzer- oder System-Tasklib, wird USERTSKL bzw. SYSTTSKL ausgegeben. Für PAM-LLMs wird ECERDLOD ausgegeben.

Beispiel für eine DBL-Liste (Ausschnitt)

```
############################## A D B L   M A P ##############################
#                                                                            #
# LOAD UNIT: %UNIT. . . . . . . . . . . . . LOAD INFO  =DEF                 #
# VERSION  : V02.7A40 . . . . . . . . . . . LOAD TIME  =2009-01-30 09:37:27 #
# CONTEXT  : LOCAL#DEFAULT. . . . . . . . . TEST OPTION=NONE                 #
#                                                                            #
#  LLM     : BLSSYS . . . . . . . . . . . . STANDARD / EXPLICIT  BLSLIB     #
#  ELE-LIB : :2OSH:$TSOS.SYSLNK.BLSSERV.027                                  #
#  ELE-NAME: BLSSYS                                                          #
#  ELE-VERS: V02.7A40                                                        #
#   OM     : PBSMAIN. . . . . . . . . . . . STANDARD                        #
#    CSECT : PBSMAIN. . . . . . . . . . . . @=       0 L=     8C0           #
#     ENTRY: NLKRES28 . . . . . . . . . . . @=      18                       #
#     ENTRY: NLKRES45 . . . . . . . . . . . @=      28                       #
#     ENTRY: PBSINI . . . . . . . . . . . . @=      38                       #
#     ENTRY: PBSGOV . . . . . . . . . . . . @=      48                       #
#    CSECT : PBSMAIN@ . . . . . . . . . . . @=     8C0 L=     E00           #
#    CSECT : PBSSSID. . . . . . . . . . . . @=    16C0 L=      80           #
#    CSECT : PBSINIV. . . . . . . . . . . . @=    1740 L=      40           #
#    CSECT : PBSOSDV. . . . . . . . . . . . @=    1780 L=      40           #
#    CSECT : PBSSXTP. . . . . . . . . . . . @=    17C0 L=      40           #
#    CSECT : PBSHWPS. . . . . . . . . . . . @=    1800 L=      40           #
#   OM     : PBSDM. . . . . . . . . . . . . STANDARD                        #
#    CSECT : PBSDM. . . . . . . . . . . . . @=    1840 L=     940           #
#     ENTRY: PBDMUCD. . . . . . . . . . . . @=    1858                      #
#    CSECT : PBSDM@@@ . . . . . . . . . . . @=    2180 L=     280           #
#   OM     : PBSSYSA. . . . . . . . . . . . STANDARD                        #
#    CSECT : PBSSYSA. . . . . . . . . . . . @=    2400 L=     940           #
#     ENTRY: PBSANCH. . . . . . . . . . . . @=    2418                      #
#     ENTRY: PBSTIME. . . . . . . . . . . . @=    2428                      #
#     ENTRY: PBSTERM. . . . . . . . . . . . @=    2438                      #
#     ENTRY: PBSMEMO. . . . . . . . . . . . @=    2448                      #
#     ENTRY: PBSWA. . . . . . . . . . . . . @=    2458                      #
#     ENTRY: PBSSINF. . . . . . . . . . . . @=    2468                      #
#     ENTRY: PBSTINF. . . . . . . . . . . . @=    2478                      #
#     ENTRY: PBSMSG . . . . . . . . . . . . @=    2488                      #
#     ENTRY: PBSBLSP. . . . . . . . . . . . @=    2498                      #
#    CSECT : PBSSYSA@ . . . . . . . . . . . @=    2D40 L=    2FC0           #
#   OM     : PBSBUFD. . . . . . . . . . . . STANDARD                        #
#    CSECT : PBSBUFD. . . . . . . . . . . . @=    5D00 L=    2500           #
#     ENTRY: PBSBDSP. . . . . . . . . . . . @=    5DE0                      #
#     ENTRY: PBSBINI. . . . . . . . . . . . @=    5D00                      #
#   OM     : PBSBUF . . . . . . . . . . . . STANDARD                        #
#    CSECT : PBSBUF . . . . . . . . . . . . @=    8200 L=    2040           #
#     ENTRY: SVPTBUF. . . . . . . . . . . . @=    8208                      #
#     ENTRY: SVSVC183 . . . . . . . . . . . @=    8250                      #
#     ENTRY: SVSVC117 . . . . . . . . . . . @=    8340                      #
#     ENTRY: SVSVC111 . . . . . . . . . . . @=    8588                      #
#     ENTRY: SVSVC110 . . . . . . . . . . . @=    83E8                      #
#     ENTRY: SVSVC109 . . . . . . . . . . . @=    8500                      #
#     ENTRY: SVSVC106 . . . . . . . . . . . @=    8478                      #
#     ENTRY: $SVBDAS. . . . . . . . . . . . @=    8A50                      #
#     ENTRY: $SVBDSP. . . . . . . . . . . . @=    8B20                      #
#     ENTRY: $SVUNAS. . . . . . . . . . . . @=    8A50                      #
#     ENTRY: $SVUNSP. . . . . . . . . . . . @=    8B20                      #
#     ENTRY: $SVVIAS. . . . . . . . . . . . @=    8A50                      #
#     ENTRY: $SVVISP. . . . . . . . . . . . @=    8B20                      #
.
.
.
# LOAD UNIT STARTING POINT. . . . . . . . . @=       0 AMODE=24 HSI=/7500   #
#                                                MMODE=TU-4K               #
#                                                                          #
######################## E N D   O F   A D B L   M A P ######################
```

Layout der DBL-Liste in einem benutzerdefinierten Datenbereich

Mit `BIND ...,USRMAPI=...,USRMAP@=...,USRMAPL=...` (wobei `USRMAPI≠NONE`) kann eine DBL-Liste in einen benutzerdefinierten Datenbereich ausgegeben werden. Diese ist ähnlich aufgebaut wie die auf SYSOUT/SYSLST ausgegebene Liste. Sie hat folgendes Layout:

- Listenkopf zur Identifikation des Bereichs
  Dieser enthält den Returncode X'08010129' falls die Länge des Datenbereichs nicht ausreicht, um die gewünschte Information aufzunehmen.
  Das Layout des Listenkopfs hängt von der Angabe INTVERS beim BIND-Makro ab: INTVERS=SRV005 erzeugt einen Listenkopf der Version 1, INTVERS=SRV006 erzeugt einen Listenkopf der Version 2. Damit wird angezeigt, ob Bibliotheksname, Elementname und Elementversion in der ausgegebenen Information enthalten sind.
- Gesamtlänge der ausgegebenen Information einschließlich Listenkopf
- Datensätze mit folgendem Aufbau:
  - Satzidentifikation
  - Länge der im Satz enthaltenen Information
  - die Information selbst

Der benutzerdefinierte Datenbereich muss bei INTVERS=SRV005 mindestens 248 Byte und bei INTVERS=SRV006 mindestens 336 Byte lang sein um die minimale Information aufzunehmen.

Wenn die Länge des Datenbereichs nicht ausreicht, um die gewünschte Information aufzunehmen, wird die Ausgabe abgebrochen.

Wenn die Länge des Datenbereichs ausreicht, wird die Ausgabe durch einen End-Satz abgeschlossen.

Das Layout des Datenbereichs kann mit mit `BIND MF=D,XPAND=USRMAP,INTVERS=SRVxxx` (xxx ≥ 005) generiert werden.

2.1.4 Gemeinsam benutzbare Programme (Shared Code)

Jedes Modul, das ein Benutzer ausführt, wird entweder in den tasklokalen Klasse-6-Speicher oder in den Speicher für gemeinsam benutzbare Programme (Shared Code) geladen. Die Module im tasklokalen Klasse-6-Speicher können nur durch die Task ausgeführt werden, die deren Laden veranlasst hat. Diese Module bilden den so genannten privaten Teil eines Programmes.

Der Speicher für Shared Code ist ein Speicherbereich, in dem ein- und diesselbe Kopie der auszuführenden Module von mehreren Tasks gleichzeitig ausgeführt werden kann. Gemeinsam benutzbare Module bilden den PUBLIC-Teil eines Programmes und müssen ablaufinvariant programmiert sein. Ein solches Programm muss Datenbereiche, die variable benutzereigene Daten aufnehmen sollen, dynamisch anfordern und im Klasse-6-Speicher des jeweiligen Benutzers anlegen. Der Benutzer kann den Shared Code entweder direkt ausführen (mit START-EXECUTABLE-PROGRAM bzw. START-PROGRAM) oder über ein privates Programm darauf zugreifen. Die Externverweise im privaten Programm können automatisch durch Programmdefinitionen (CSECTs, ENTRYs) im Shared Code befriedigt werden.

Vorteile gemeinsam benutzbarer Programme

- Da ein Modul nur einmal geladen wird, spart man für alle folgenden Aufrufe die Ladezeit.
- Haupt- und Seitenspeicher werden entlastet, da es nur eine einzige Kopie des Moduls im Speicher für Shared Code gibt (nicht jede Task besitzt eine eigene Kopie in ihrem Klasse-6-Speicher, siehe Bild 5 am Beispiel von Shared Code des Systems).
- Die Seitenwechselrate verringert sich, da sich nur eine Kopie im Seitenwechselbereich befindet und weil „nur lesbare“ Seiten nicht auf den Seitenspeicher zurückgeschrieben werden, wenn sie im Hauptspeicher nicht mehr benötigt werden, d.h. eine Aktualisierung des Seitenspeichers ist für sie unnötig.

Der Einsatz von gemeinsam benutzbaren Programmen ist also für große Programme vorteilhaft, die längere Zeit im Speicher stehen (Dialog-, Transaktionsbetrieb) und evtl. von mehreren Benutzern gleichzeitig verwendet werden.

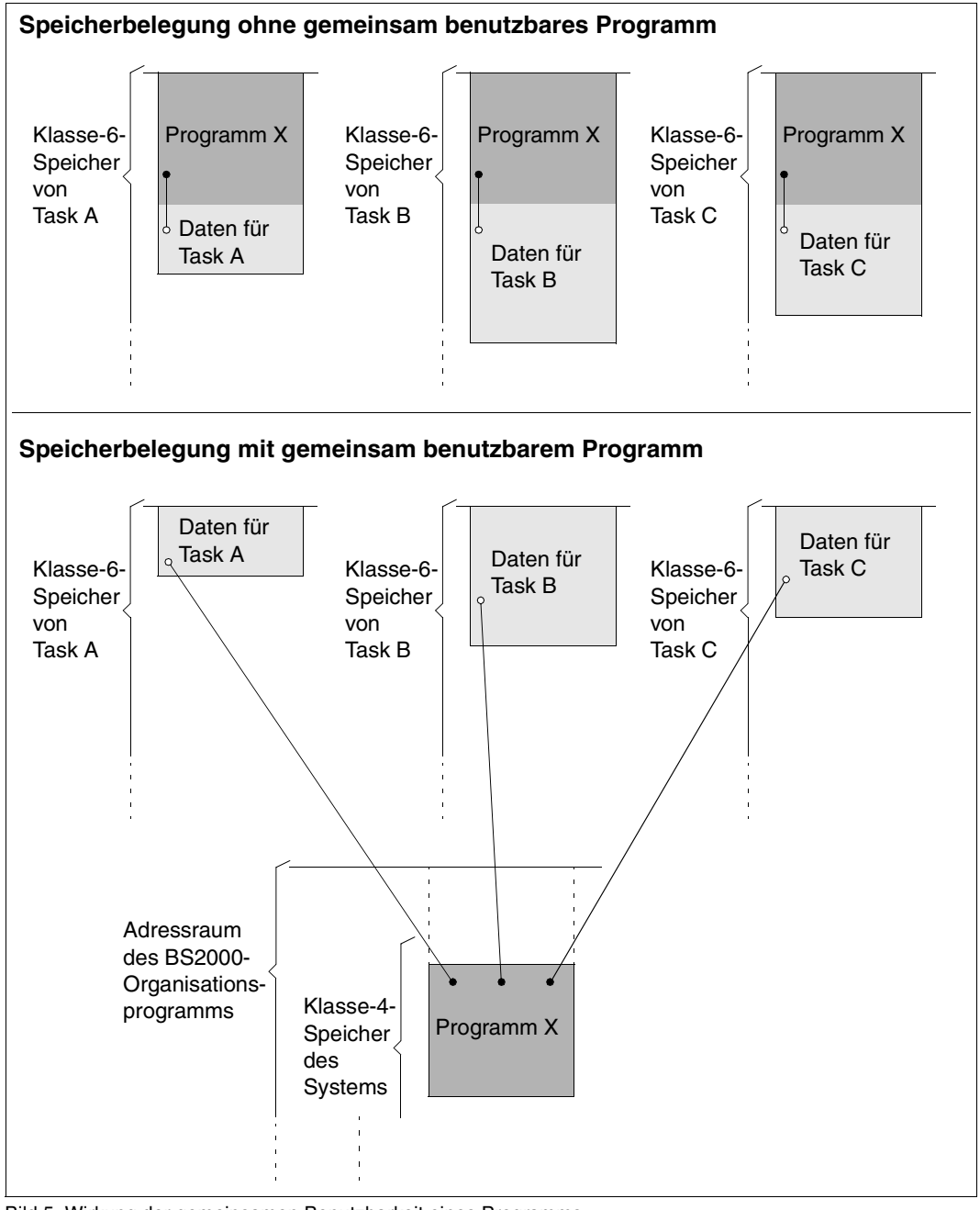


Bild 5: Wirkung der gemeinsamen Benutzbarkeit eines Programms

Der DBL bietet verschiedene Möglichkeiten für das Laden und die Verwaltung von Shared Code, die in den folgenden Abschnitten beschrieben sind.

Shared Code des Systems

Der Shared Code des Systems wird in den Systemadressraum (Klasse-3/4-Speicher oder privilegierter Klasse-5-Speicher) geladen und alle Tasks bzw. Benutzer können gemeinsam darauf zugreifen. Der DBL sorgt dafür, dass dieser Shared Code immer schreibgeschützt ist, unabhängig davon, welchen Wert das Attribut READ-ONLY hat.

Kontrolliert wird dieser Speicherbereich durch die Subsystemverwaltung (DSSM), da der Shared Code in Form von unprivilegierten Subsystemen geladen wird (siehe Handbuch „Verwaltung von Subsystemen“ [10]). Für jedes Subsystem muss im DSSM eine Liste von sichtbaren Symbolen festgelegt werden, die außerhalb des Subsystems bekannt sein sollen. Nur diese Symbole, die als „Connectable Entries“ bezeichnet werden, können in START/LOAD-EXECUTABLE-PROGRAM bzw. START/LOAD-PROGRAM oder im BIND-Makro angegeben werden und sind zur Befriedigung von Externverweisen nutzbar.

Shared Code des Benutzers

Shared Code, der von einem Benutzer ohne besondere Privilegien verwaltet werden soll, wird in Common Memory Pools abgelegt. Common Memory Pools werden im Klasse-6-Speicher angelegt. Die dort geladenen Module sind für alle die Tasks gemeinsam benutzbar, die sich an den Memory Pool angeschlossen haben.

Der Shared Code kann mit dem Makro ASHARE in den Common Memory Pool geladen werden. Diese Funktion ist ab BS2000/OSD-BC V1.0 verfügbar. Jede Task, die sich an den Common Memory Pool angeschlossen hat, kann:

– ein gemeinsam benutzbares Programm (bestehend aus einer Menge von Modulen) in den Memory Pool laden

– ein gemeinsam benutzbares Programm entladen

– Informationen über die geladenen Module abfragen

Der Name des Programmes und jedes sichtbare Symbol im Shared Code des Benutzers können im Operanden SYMBOL in den Kommandos START/LOAD-EXECUTABLE-PROGRAM bzw. START/LOAD-PROGRAM sowie im Makroaufruf BIND angegeben bzw. zur Befriedigung von Externverweisen genutzt werden. Wird in einer Task, die noch nicht an den Common Memory Pool angeschlossen ist, auf Programmdefinitionen im Shared Code in diesem Memory Pool verwiesen, dann führt der DBL automatisch den ENAMP-Makro aus (ENAble Memory Pool, siehe „Makroaufrufe an den Ablaufteil“ [7]).

Mit dem Makro BIND (Parameter MPID) kann das Laden in den Common Memory Pool ebenfalls erreicht werden. Auf Programme, die so geladen wurden, kann jedoch nur die Task zugreifen, die das Laden in den Memory Pool veranlasst hat. Das bedeutet, dass diese Task für die Verwaltung des Memory Pools selbst verantwortlich ist. Die Informationen über ein so geladenes Programm (z.B. die Ausgaben des VSVI1-Makros) sind ebenfalls nur für die Task verfügbar, die das Programm geladen hat.

Im Folgenden wird immer davon ausgegangen, dass der Shared Code mit dem ASHARE-Makro in ein- und denselben Common Memory Pool geladen wurde.

Ein Programm kann entweder nur mit dem BIND-Makro oder nur mit dem ASHARE-Makro in ein- und denselben Memory Pool geladen werden.

LLMs mit PUBLIC-Slices

Beim Erzeugen eines LLM durch den BINDER kann der Benutzer festlegen, ob Slices nach dem Attribut PUBLIC gebildet werden sollen. Das LLM wird dann in zwei Teile (Slices) aufgespalten, die beide das Format eines einfachen LLMs haben. Die Slice mit den CSECTs mit dem Attribut PUBLIC=YES kann als Shared Code geladen werden. Die andere Slice, in der alle CSECTs mit dem Attribut PUBLIC=NO zusammengefasst sind, ist nur im tasklokalen Benutzerspeicher ladbar und bildet den privaten Teil des LLM.

Der DBL lädt nur den PUBLIC-Teil des LLM, wenn das LLM mit DSSM als unprivilegiertes Subsystem oder mit dem ASHARE-Makro geladen wird.

Beim Laden des LLM mit den Kommandos START/LOAD-EXECUTABLE-PROGRAM (bzw. START/LOAD-PROGRAM) oder mit dem BIND-Makro wird zuerst der private Teil des LLM geladen. Enthält der private Teil Externverweise auf den PUBLIC-Teil, dann können zwei Fälle eintreten:

1. Der PUBLIC-Teil des LLM war bereits im Shared-Code-Speicher geladen. Die Externverweise im privaten Teil werden durch den Shared Code befriedigt.
2. Der PUBLIC-Teil befand sich noch nicht im Shared-Code-Speicher und wird als privater Code in den tasklokalen Benutzerspeicher geladen. Dasselbe passiert, wenn der Benutzer SHARE[-SCOPE]=NONE im Ladeaufruf angegeben hat und den PUBLIC-Teil nicht als Shared Code nutzen will.

Beim Laden des privaten Teils eines LLM, in dem Externverweise auf den PUBLIC-Teil auftreten, überprüft der DBL, ob auch wirklich beide Teile zu ein- und demselben LLM gehören. Ist das nicht der Fall, dann wird der PUBLIC-Teil des LLM als privater Code in den tasklokalen Benutzerspeicher geladen.

Soll der PUBLIC-Teil des LLM über das DSSM geladen werden, so muss der Benutzer bereits im BINDER-Lauf in den Anweisungen START-LLM-CREATION oder MODIFY-LLM-ATTRIBUTES die Liste der SUBSYSTEM-ENTRIES angeben, die nach außen hin für den PUBLIC-Teil des LLM bekannt sein sollen. Diese Möglichkeit besteht ab BINDER V1.1A.

2.2 Kontextkonzept

Der Begriff **Kontext** hat im Folgenden drei verschiedene Bedeutungen.

Ein Kontext kann sein:

– ein Satz von Objekten mit einer logischen Struktur,

– eine Umgebung für das Binden und Laden,

– eine Umgebung für das Entladen und Entbinden.

Die Verwendung von Kontexten durch den DBL 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 einzigen Aufruf entladen werden.

– Symbole mit gleichen Namen in verschiedenen Kontexten bewirken keinen Namenskonflikt, da jeder Kontext seine eigene Symboltabelle hat.

Im Folgenden sind die einzelnen Kontextbegriffe näher erläutert.

2.2.1 Kontext als Satz von Objekten

Ein Kontext als Satz von Objekten hat eine logische Struktur. Er ist wie folgt hierarchisch strukturiert (siehe Bild 6):

1. Objekte der niedrigsten Stufe sind **Programmdefinitionen**. Dazu zählen folgende Objekte:
   - Programmabschnitte (CSECTs)
   - Einsprungstellen (ENTRYs)
   - COMMON-Bereiche
   - Pseudoabschnitte (DSECTs)
   - Externe Pseudoabschnitte (XDSECs-D)
2. Objekte der nächst höheren Stufe sind **Module**. In ihnen sind die Programmdefinitionen enthalten. Module können Bindemodule (OMs) oder LLMs sein.
3. Objekte der nächst höheren Stufe werden als **Ladeeinheiten** bezeichnet. Eine Ladeeinheit enthält alle Module, die mit einem einzigen Ladeaufruf gebunden und geladen werden sollen.
4. Der Kontext selbst besteht aus einer oder mehreren Ladeeinheiten. Er ist das Objekt der höchsten Stufe.

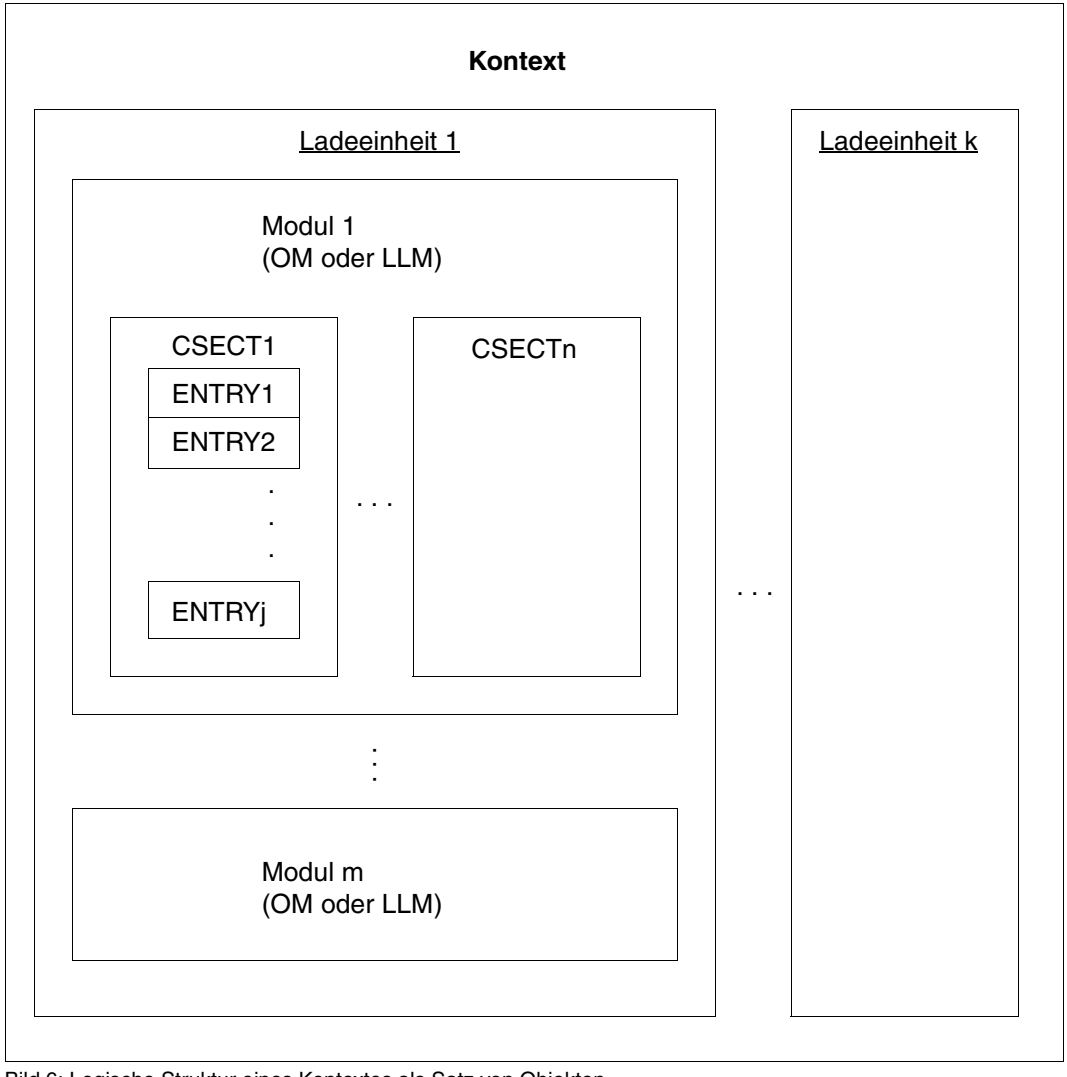


Bild 6: Logische Struktur eines Kontextes als Satz von Objekten

2.2.2 Kontext als Umgebung für das Binden und Laden

Dieser Kontext ist ein Hauptspeicherbereich, in den die Module einer Ladeeinheit geladen werden. Den Hauptspeicherbereich baut der DBL dynamisch auf. Der Benutzer kann also keinen festen Speicherplatz dafür reservieren. Der Hauptspeicherbereich kann vollständig in einem Memory Pool liegen oder kann sich auf mehrere Memory Pools verteilen. Er darf jedoch nur innerhalb derselben Speicherklasse liegen. Für die in diesem Hauptspeicherbereich geladenen Module existiert eine Symboltabelle, in der jedem Symbol eine Adresse zugeordnet ist.

Der Kontext für das Binden und Laden kann als Link-Kontext oder Referenz-Kontext benutzt werden:

Link-Kontext

In den Link-Kontext werden die Module einer Ladeeinheit geladen. Alle Symbole eines Moduls, das in diesen Kontext geladen wird, werden in die Kontext-Symboltabelle eingetragen. Der Link-Kontext wird von der Autolink-Funktion des DBL benutzt, um die Externverweise in den geladenen Modulen zu befriedigen (siehe Seite 24).

Außerdem wird er benutzt, um nicht befriedigte Externverweise zwischenzuspeichern, falls dies im Ladeaufruf gefordert wird (Operand UNRESOLVED-EXTRNS=*DELAY). Wird eine neue Ladeeinheit im Link-Kontext geladen, versucht der DBL am Ende des Ladens, die gespeicherten Externverweise mit CSECTs und ENTRYs der neuen Ladeeinheit zu befriedigen. Dieser Vorgang wiederholt sich beim Laden weiterer Ladeeinheiten, solange der Kontext besteht.

Den Namen des Link-Kontextes legt der Benutzer im Makroaufruf BIND fest (Operand LNKCTX@ oder LNKCTX). Wird kein Name angegeben, wählt der DBL als Standardname LOCAL#DEFAULT. Wird der DBL über ein Kommando LOAD- und START-EXECUTABLE-PROGRAM (bzw. LOAD- und START-PROGRAM) aufgerufen, bestimmt der DBL als Standardname LOCAL#DEFAULT.

Referenz-Kontext

Der Referenz-Kontext dient zur Befriedigung von Externverweisen, wenn diese im Link-Kontext nicht gefunden werden. Der Referenzkontext entsteht dadurch, dass er wechselweise Link-Kontext in einem anderen Ladevorgang ist. Mehrere Referenz-Kontexte können gleichzeitig vorhanden sein (maximal 200). Den Namen und die Anzahl der möglichen Referenz-Kontexte legt der Benutzer im Makroaufruf BIND fest (Operand REFCTX@ und REFCTX#).

2.2.3 Kontext als Umfeld für das Entladen und Entbinden

Für das Entladen (siehe Seite 60) ist der Kontext die größte Einheit, die entladen werden kann. Als Objekte niederer Stufen können Ladeeinheiten und Module (LLMs oder OMs) entladen werden.

2.2.4 Merkmale eines Kontextes

Jeder Kontext hat als Merkmale einen Geltungsbereich und eine Zugriffsberechtigung.

Der **Geltungsbereich** des Kontextes legt die Klasse des Speichers fest, in dem der Kontext physisch liegen soll. Man unterscheidet folgende Geltungsbereiche:

– Geltungsbereich USER
  Der Kontext liegt im Benutzeradressraum (Klasse-6-Speicher). Der Kontextname beginnt mit einem Buchstaben. Die Anzahl der USER-Kontexte ist auf maximal 201 beschränkt.
  Für Shared Code in Common Memory Pools ist der Geltungsbereich ebenfalls USER. Der Name dieses Kontexts beginnt mit „#“. Die Anzahl der USER-Pool-Kontexte in einem Memory Pool ist auf maximal 15 beschränkt.

– Geltungsbereich SYSTEM
  Der Kontext liegt im Systemadressraum (Klasse-4-Speicher oder Klasse-3-Speicher). Der Kontextname beginnt mit dem Zeichen „$“.

– Geltungsbereich SSLOCAL
  Der Kontext gehört zu einem lokalen Subsystem und liegt im nichtprivilegierten Klasse-5-Speicher. Der Kontextname beginnt mit dem Zeichen „*“.

Link-Kontext und Referenz-Kontext müssen für nicht-privilegierte Benutzer immer im Geltungsbereich USER liegen.

Die **Zugriffsberechtigung** legt fest, welche Benutzer auf den Inhalt des Kontextes zugreifen dürfen. Man unterscheidet zwei Zugriffsberechtigungen:

– Privilegierter Kontext
  Nur privilegierte Benutzer dürfen auf den Inhalt des Kontextes zugreifen (z.B. Systemverwaltung).

– Nicht privilegierter Kontext
  Jeder Benutzer darf auf den Inhalt des Kontextes zugreifen. Der Zugriff auf den Kontext eines Memory Pools ist nur für die Benutzer möglich, die an den Memory Pool angeschlossen sind.

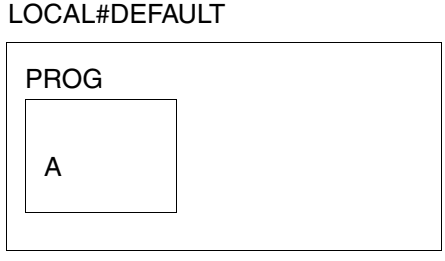
2.2.5 Beispiel zur Kontextverwendung

Das folgende Beispiel zeigt, wie Kontexte zur Vermeidung von Namenskonflikten und zur gesteuerten Befriedigung von Externverweisen verwendet werden können.

1. Ein Programm, dass in einer Programmbibliothek als Element mit dem Namen PROG abgespeichert ist, wird aufgerufen. Der DBL bildet zur Befriedigung von Externverweisen den Standard-Link-Kontext LOCAL#DEFAULT.

   ```
   /START-EXECUTABLE-PROGRAM FROM-FILE=(LIBRARY=...,ELEMENT-OR-SYMBOL=PROG)
   ```

   Ergebnis

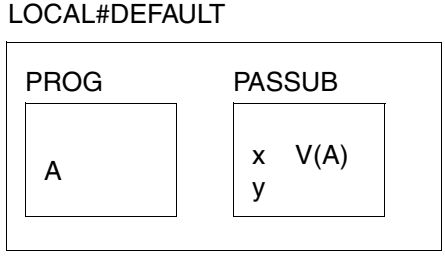
   

   A ist ein ENTRY in PROG

2. Während des Programmablaufes ruft PROG den Makro BIND auf und aktiviert damit ein PASCAL-Unterprogramm.

   ```
   BIND SYMBOL=PASSUB,LIBNAM=...
   ```

   Ergebnis

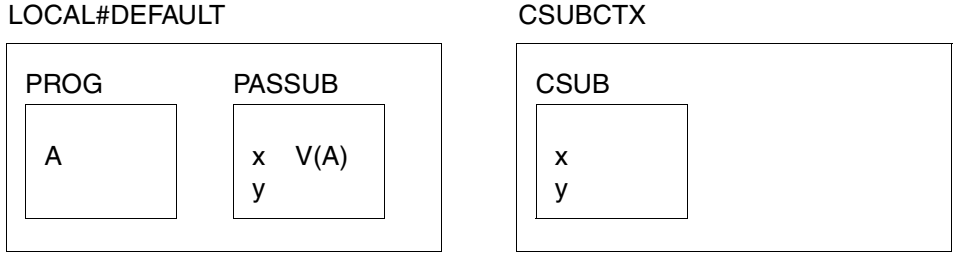
   V(A) in PASSUB wird durch den ENTRY A in PROG befriedigt.

   x und y sind Programmdefinitionen in einigen Laufzeitroutinen, die PASSUB verwendet.

3. Im weiteren Programmverlauf ruft PROG das Unterprogramm CSUB auf, das in einer anderen Programmiersprache (in C) als PASSUB geschrieben ist, aber ebenfalls Laufzeitroutinen x und y benutzt. Zur Vermeidung von Namenskonflikten müsste der BIND-Makro wie folgt aufgerufen werden:

   ```
   BIND SYMBOL=CSUB,LIBNAM=...,...,LNCTX=CSUBCTX
   ```

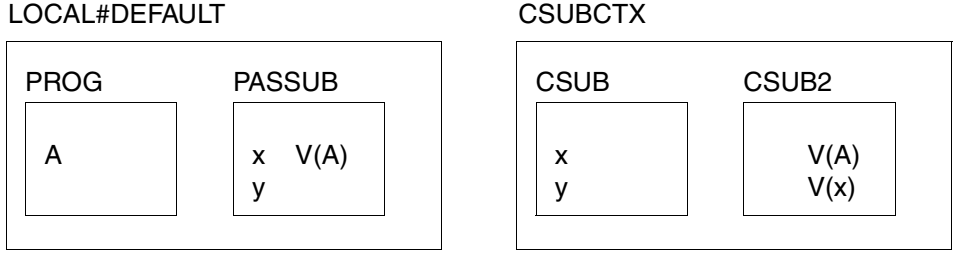
   Ergebnis



4. Ein weiteres Unterprogramm, das in C programmiert ist und in dem es unter anderem auch einen Externverweis auf den ENTRY A aus dem Hauptprogramm PROG gibt, wird aufgerufen. Externverweis V(A) soll mit ENTRY A aus PROG befriedigt werden; andere Externverweise sollen jedoch durch die C-Laufzeitroutinen befriedigt werden. Deshalb muss PROG das Unterprogramm CSUB2 wie folgt aufrufen:

   ```
   BIND SYMBOL=CSUB2,LIBNAM=...,...,LNKCTX=CSUBCTX,REFCTX=LOCAL#DEFAULT
   ```

   Ergebnis



   V(A) wird durch A in PROG befriedigt.
   V(x) wird durch x in CSUB befriedigt.

2.3 Zusätzliche Funktionen

Zusätzlich zum Binden und Laden bietet der DBL folgende Funktionen:

- Entladen und Entbinden von Objekten,
- Ausgeben von Binde- und Ladeinformationen,
- Aufbau einer eigenen Symboltabelle,
- Verarbeitung von REP-Dateien.

2.3.1 Entladen und Entbinden von Objekten

Nach dem Beenden des Programms mit dem Makroaufruf TERM, dem Kommando CANCEL-PROGRAM oder dem Kommando EXIT-JOB (LOGOFF), wird der gesamte Benutzerspeicherplatz freigegeben, der vom DBL und vom statischen Lader ELDE bisher belegt war.

Der Makroaufruf UNBIND gibt während des Programmlaufs den Speicherplatz frei, der von nicht mehr benötigten Objekten belegt ist. Das Objekt kann ein Kontext, eine Ladeeinheit, ein LLM oder ein Bindemodul (OM) sein. Bindemodule und LLMs sind nur komplett entladbar. Die Objekte können auch über einen Pfad angegeben werden, z.B.:

- ein Modul in einem bestimmten Kontext,
- ein Modul in einer bestimmten Ladeeinheit mit einer festgelegten Version,
- ein Modul in einer bestimmten Ladeeinheit mit einer festgelegten Version in einem bestimmten Kontext,
- eine Ladeeinheit mit einer festgelegten Version in einem bestimmten Kontext.

Module, die Bestandteil einer List-Name-Unit sind (siehe Abschnitt „Verwaltung von List-Name-Units“ auf Seite 38), können nicht einzeln, sondern nur zusammen mit der vollständigen List-Name-Unit entladen werden.

In jedem Fall wird nur das erste gefundene und zum Pfad passende Objekt entladen. Der belegte Speicherplatz wird nur seitenweise (in Einheiten von 4 KByte) freigegeben. Der Speicherplatz wird dem Memory Management nur zurückgegeben, falls keine anderen Module auf derselben Seite Speicherplatz beanspruchen. Ansonsten vermerkt sich der DBL die freien Bereiche und verwendet sie bei der nächsten Gelegenheit.

Zusätzlich zum Entladen von Objekten ist es möglich, Externverweise zu Programmabschnitten (CSECTs) und Einsprungstellen (ENTRYs) in diesen Objekten zu entbinden (Parameter UNLINK=YES).

Entbinden bedeutet, dass befriedigte Externverweise zu CSECTs und ENTRYs in den entladenen Objekten freigegeben werden. Ausgenommen sind befriedigte XDSECs. Die Externverweise in den Objekten entsprechen dann unbefriedigten Externverweisen, wie sie zum Zeitpunkt des Bindens und Ladens vorhanden waren. Sie erhalten den Wert, der im Ladeaufruf für unbefriedigte Externverweise angegeben wurde (Operand ERROR-EXIT) und werden, falls im Ladeaufruf gefordert, als Externverweise behandelt, die zu einem späteren Zeitpunkt befriedigt werden.

Entbinden wird nur innerhalb des Kontextes durchgeführt, in dem das Modul entladen wird. Wird in anderen Kontexten auf das entladene Objekt verwiesen, so werden diese Externverweise nicht entbunden. Entbinden wird nur durchgeführt, wenn die Externverweise zu einer Ladeeinheit gehören, für die im Ladeaufruf ein Externadressbuch mit Referenzen geladen wurde (Operand LOAD-INFORMATION=*REFERENCES).

2.3.2 Ausgeben von Binde- und Ladeinformationen

Binde- und Ladeinformationen über die gebundenen Ladeeinheiten und deren Kontexte kann der Benutzer mit dem Makroaufruf VSVI1 abfragen. Diese Informationen können für einen oder mehrere angegebene Kontexte oder für alle Kontexte angefordert werden. Benutzer ohne besondere Privilegien können nur auf nicht privilegierte Kontexte zugreifen. Informationen über Shared Code in Common Memory Pools sind nur für die Benutzer verfügbar, die sich vor dem Makroaufruf VSVI1 an den Common Memory Pool angeschlossen haben.

Folgende Informationen können abgefragt werden:

- Eine Liste mit Namen der Kontexte (SELECT=CTXLIST),
- Der Umfang des in einem Kontext geladenen Codes und der Umfang der dazugehörigen Binde- und Ladeinformationen (SELECT=CTXSIZE),
- Eine Liste mit Namen, Ladeadressen, Längen, Typen, Attributen und Kontexten von CSECTs, ENTRYs und COMMON-Bereichen (SELECT=ALLLIST),
- Eine Liste mit Namen, Ladeadressen, Längen, Typen, Attributen und Kontexten aller CSECTs und COMMON-Bereiche (SELECT=MODLIST),
- Ein Satz mit Name, Ladeadresse, Länge, Typ, Attributen und Kontext eines einzelnen Programmabschnitts (CSECT), ENTRY oder COMMON-Bereichs (SELECT= BYNAME),
- Ein Satz mit Name, Ladeadresse, Länge, Typ, Attributen und Kontext eines durch eine Adresse angegebenen Programmabschnitts (CSECT) oder COMMON-Bereichs (SELECT=BYADDR).

Die Einzelinformationen (Name, Ladeadresse, Länge, Attribut, Typ und Kontext) können unabhängig voneinander ausgewählt werden. Es ist auch möglich, nur die Länge der gewünschten Ausgabeinformationen anzufordern.

Der Makroaufruf PINF liefert globale Informationen über Programme, die mit den Kommandos LOAD- und START-EXECUTABLE-PROGRAM (bzw. LOAD- und START-PROGRAM) geladen wurden.

Folgende Informationen können angefordert werden:

- der interne Programmname (SELECT=INTNAME),
- die interne Programmversion (SELECT=INTVERS),
- das Erzeugungsdatum des Programmes (SELECT=INTDATE),
- der Copyright-Name des Programmes (SELECT=COPRIGHT),
- der Name der Bibliothek, in der sich das Programm befand (SELECT=FILENAME),
- der Elementname des Programmes in der Bibliothek (SELECT=ELEMNAME),
- die Elementversion (SELECT=ELEMVERS),
- den Elementtyp (SELECT=ELEMTYPE),
- der Name, der im Ladeaufruf (bei LOAD- oder START-EXECUTABLE-PROGRAM bzw. LOAD- oder START-PROGRAM) angegeben wurde (SELECT=SPECNAME),
- ein Indikator, der anzeigt, ob das geladene Programm vom statischen Lader ELDE geladen wurde oder das erste Modul einer Ladeeinheit war und vom DBL geladen wurde (SELECT=LOADTYPE).

2.3.3 Aufbau einer eigenen Symboltabelle

Ab BS2000/OSD-BC V3.0 ist es mit Hilfe der Makros ETABLE und ETABIT möglich, für Programmdefinitionen im geladenen Programm eine eigene, kontextbezogene Symboltabelle aufzubauen. Dem DBL können damit Symbolinformationen bekannt gegeben werden, womit ein Informationsaustausch zwischen verschiedenen Programmteilen möglich ist.

Der Makro ETABLE ersetzt den früheren Makroaufruf TABLE.
Jedoch behandelt der DBL Namenskonflikte bei BIND und ETABLE anders als früher bei BIND und TABLE. Die Behandlung von Namenskonflikten bei ETABLE ist im Handbuch „Makroaufrufe an den Ablaufteil“ [7] beschrieben.

2.3.4 Verarbeitung von REP-Dateien

REP-Dateien können in allen Ladeaufrufen mit den Kommandos LOAD- und START-EXECUTABLE-PROGRAM (bzw. LOAD- und START-PROGRAM) sowie mit den Makros ASHARE und BIND angegeben werden.

Eine REP-Datei gehört immer zu einer Ladeeinheit. Die in der REP-Datei enthaltenen REP-Sätze erlauben eine byteweise Korrektur der einzelnen Module einer Ladeeinheit in der Art und Weise, wie das auch beim Laden des BS2000-Organisationsprogrammes möglich ist (siehe Handbuch „Einführung in die Systembetreuung“ [9]).
Die REP-Sätze können entweder auf alle Module im Linkkontext oder nur auf die Module der Ladeeinheit angewendet werden (REPSCOP=CONTEXT oder UNIT). In die REP-Verarbeitung im Linkkontext können auch maskierte Symbole einbezogen werden.

Wenn der Ladevorgang beendet ist, öffnet der DBL die zur Ladeeinheit gehörende REP-Datei, liest die enthaltenen REP-Sätze und wendet die REP-Sätze auf die Module der Ladeeinheit oder auf alle Module im Linkkontext an. Der DBL greift dabei auch auf eine eventuell vorhandene NOREF-Datei zurück, die eine Liste von Symbolen (CSECTs und ENTRYs) enthält, die für die REP-Verarbeitung nicht von Bedeutung sind (siehe Seite 64).

Korrekturen, die den Code in einem Systemkontext betreffen, werden durch den BLS-Lock-Manager gesteuert. Während des gesamten Binde-/Ladeprozesses (einschließlich der REP-Verarbeitung) ist der betroffene Systemkontext gegen andere Schreibzugriffe geschützt.

Format der REP-Sätze und REP-Dateien

Die REP-Sätze müssen in dem Format bereitgestellt werden, das auch für REPs des BS2000-Organisationsprogrammes verwendet wird (einschließlich relativierbarer REPs). Das Format der REP-Sätze und der REP-Dateien sind ausführlich im Handbuch „Einführung in die Systembetreuung“ [9] beschrieben.

Jeder REP-Satz wird vor der Verarbeitung auf richtiges Format untersucht. Geprüft werden dabei u.a. die enthaltenen Daten, das Parity-Byte, die Modulversion und der Modulname. Das Klassenkennzeichen (Spalte 70) hat für Ladeeinheiten keine Bedeutung und wird deshalb nicht beachtet.

Fehlerhafte oder nicht erkannte REP-Sätze werden auf SYSOUT protokolliert und die Verarbeitung wird mit dem nächsten REP-Satz fortgesetzt.

Wenn die im Ladeaufruf angegebene REP-Datei nicht vorhanden ist, wird die REP-Verarbeitung mit der Meldung BLS0977/BLP0977 abgebrochen und die Ladeeinheit wird ohne REP-Korrekturen geladen. Bei anderen DVS-Fehlern beim Zugriff auf die REP-Datei bietet der DBL mit der Meldung BLS0998/BLP0998 einen erneuten Zugriffsversuch an. Wenn auch dieser Versuch fehlschlägt, wird die Ladeeinheit ohne REP-Korrekturen geladen.

REP-Dateien für Subsysteme

Eine REP-Datei, die zu einem Subsystem gehört, unterliegt folgender Namenskonvention:

```
SYSREP.<subsystem-name>.<subsystem-version>
```

Die REP-Datei sollte mehrbenutzbar sein - besonders dann, wenn das Subsystem nicht vorgeladen, sondern erst beim ersten Aufruf geladen wird.

Rückinformationen und Fehleranzeigen

Beim Verarbeiten der REP-Dateien werden folgende BLS-Meldungen ausgegeben:

BLS0990/BLP0990	nach dem erfolgreichen Öffnen der REP-Datei
BLS0980/BLP0980	für jeden Kommentarsatz in der REP-Datei
BLS0989/BLP0989, BLS0991/BLP0991, BLS0992/BLP0992, BLS0993/BLP0993, BLS0998/BLP0998	wenn Fehler bei der REP-Verarbeitung auftreten

NOREF-Dateien

Eine NOREF-Datei ist einer REP-Datei zugeordnet und enthält eine Liste von CSECTs und ENTRYs, die bei der REP-Verarbeitung nicht beachtet werden müssen. D.h. wenn der DBL eine CSECT oder einen ENTRY nicht findet, wofür ein REP-Satz vorhanden ist, dann durchsucht der DBL die NOREF-Datei. Steht dieses Symbol in der NOREF-Datei, so übergeht der DBL diesen REP-Satz ohne Fehlermeldung.

Damit ist es möglich, eine REP-Datei anzuwenden auf:

– Softwareprodukte, die aus mehreren Subsystemen oder Ladeeinheiten bestehen,

– Subsysteme oder Programme, die aus mehreren Ladeeinheiten bestehen.

i Ein „S“ (für Selectable Unit) in Spalte 69 des REP-Satzes hat die selbe Wirkung wie ein entsprechender Eintrag in der NOREF-Datei.

Format der NOREF-Dateien

Eine NOREF-Datei ist eine SAM-Datei mit variabler Länge. Sie besteht aus 8 Zeichen langen Sätzen (pro Symbolname ein Satz), die durch Leerzeichen voneinander getrennt sind. Der erste Satz der NOREF-Datei enthält in den ersten 4 Byte sedezimal die Anzahl der nachfolgenden Sätze.

Namenskonvention für NOREF-Dateien

Der DBL kann eine NOREF-Datei verarbeiten, wenn die Namenskonventionen für BS2000-REP-Dateien eingehalten werden:

– Wenn der im Ladeaufruf angegebenen REP-Dateiname den Namensteil „SYSREP“ enthält, ersetzt der DBL diesen Namensteil durch „SYSNRF“ und sucht eine NOREF-Datei mit diesem Namen.

– Andernfalls sucht der DBL eine NOREF-Datei mit den gleichen Namen wie die Hauptbibliothek, die zusätzlich das Suffix „.NOREF“ hat.

Der DBL gibt keine Meldung aus, wenn er keine NOREF-Datei findet.

Beim erfolgreichen Öffnen der NOREF-Datei gibt der DBL die Meldung BLS0995/BLP0995 aus.
Bei Fehlern während der NOREF-Verarbeitung werden die Meldungen BLS0996/BLP0996 oder BLS0997/BLP0997 ausgegeben.

2.4 Ablauf des dynamischen Bindeladers

2.4.1 Aufruf

Der Benutzer aktiviert den DBL mit Kommandos oder Makroaufrufen und steuert den Ablauf mit den Operanden dieser Kommandos oder Makroaufrufe.

Folgende **Kommandos** rufen den DBL auf:

- Die Kommandos START-EXECUTABLE-PROGRAM oder LOAD-EXECUTABLE-PROGRAM, falls keine Lademodule (Lademoduldatei oder Bibliothekselement des Typs C) geladen werden.
  Das Kommando START-EXECUTABLE-PROGRAM bindet Module zu einer Ladeeinheit zusammen, lädt diese in den Hauptspeicher und startet sie. Wenn der Benutzer die Ladeeinheit nur laden, aber nicht starten will, kann er das Kommando LOAD-EXECUTABLE-PROGRAM verwenden.
- Die Kommandos START-PROGRAM oder LOAD-PROGRAM, falls der Operand *MODULE angegeben wurde.
  Diese Kommandos werden nur noch aus Kompatibilitätsgründen unterstützt.
  Das Kommando START-PROGRAM bindet Module zu einer Ladeeinheit zusammen, lädt diese in den Hauptspeicher und startet sie. Wenn der Benutzer die Ladeeinheit nur laden, aber nicht starten will, kann er das Kommando LOAD-PROGRAM verwenden.
- Das Kommando CANCEL-PROGRAM
  Dieses Kommando beendet den Programmlauf und gibt den gesamten Benutzerspeicherplatz frei, der vom DBL und vom statischen Lader ELDE bisher belegt war.
- Die Kommandos MODIFY-DBL-DEFAULTS, RESET-DBL-DEFAULTS und SHOW-DBL-DEFAULTS setzen Voreinstellungen für den DBL-Ablauf oder zeigen die Voreinstellungen an.
- Das Kommando SELECT-PROGRAM-VERSION
  Dieses Kommando legt fest, welche Version eines Programmes verwendet wird, wenn der DBL auf mehrere Programmversionen zugreifen kann.

Folgende **Makroaufrufe** rufen den DBL auf:

– Der Makroaufruf BIND
Er bindet eine weitere Ladeeinheit in das ablaufende Programm ein.

– Der Makroaufruf UNBIND
Er gibt während des Programmlaufs Speicherplatz frei, der von nicht mehr benötigten Objekten belegt ist. Das Objekt kann ein Kontext, eine Ladeeinheit, ein LLM oder ein Bindemodul sein.

– Der Makroaufruf ASHARE
Er bindet und lädt Shared Code, den der Benutzer in Common Memory Pools als gemeinsam benutzbar zur Verfügung stellen will.

– Der Makroaufruf DSHARE
Er entlädt Shared Code aus Common Memory Pools, der mit dem ASHARE-Makro geladen wurde.

– Der Makroaufruf LDSLICE
Er lädt eine Slice, die in einem LLM vom Benutzer definiert wurde, in den Hauptspeicher.

– Der Makroaufruf VSVI1
Er liefert dem Benutzer Binde- und Ladeinformationen über die gebundenen Ladeeinheiten und deren Kontexte.

– Der Makroaufruf PINF
Er liefert dem Benutzer globale Informationen über geladene Programme.

– Die Makroaufrufe ILEMGT und ILEMIT
Sie ermöglichen den Aufbau und die Verwaltung von ILE-Listen.

– Die Makroaufrufe ETABLE und ETABIT
Sie ermöglichen den Aufbau einer eigenen Symboltabelle im Kontext.

– Der Makroaufruf GETPRGV
Er liefert dem Benutzer die aktuell ausgewählte Programmversion.

– Der Makroaufruf SELPRGV
Damit kann eine bestimmte Programmversion ausgewählt werden.

2.4.2 Meldungsbehandlung

Für die Meldungsbehandlung durch den DBL sind die Meldungen in bestimmte Meldungsklassen eingeteilt. Folgende Meldungsklassen sind möglich:

INFORMATION Informationsmeldung

WARNING Warnungsmeldung

ERROR Fehlermeldung

Jede Meldungsklasse hat eine bestimmte Gewichtung. Die Meldungsklasse INFORMATION hat die niedrigste, die Meldungsklasse ERROR die höchste Gewichtung.

Mit dem Operanden MESSAGE-CONTROL im Ladeaufruf kann man festlegen, für welche Meldungsklassen Meldungen ausgegeben werden sollen. Dieser Operand bestimmt die niedrigste Meldungsklasse, ab der Meldungen ausgegeben werden. Die folgende Tabelle zeigt, wie der Operand MESSAGE-CONTROL die Meldungsausgabe steuert.

	MESSAGE-CONTROL			
Meldungsklasse	*INFORMATION	*WARNING	*ERROR	*NONE
INFORMATION	ja	nein	nein	nein
WARNING	ja	ja	nein	nein
ERROR	ja	ja	ja	nein

Ist der Auftragsschalter 4 gesetzt, dann werden einige Informationsmeldungen unterdrückt. Erläuterungen zur Bedeutung der vom DBL ausgegebenen Meldungen erhalten Sie mit dem Kommando HELP-MSG-INFORMATION.

Abhängig vom Wert des Systemparameters EACTETYP (siehe Handbuch „Einführung in die Systembetreuung“ [9]) werden einige BLS-Meldungen auch auf SYSOUT und/oder die Operator-Konsole ausgegeben.

Meldungsschlüssel

Die Meldungschlüssel der BLS-Meldung sind wie die aller BS2000-Meldungen siebenstellig, wobei die ersten drei Buchstaben für die BS2000-Meldungsklasse und die letzten vier Zeichen für die Meldungsnummer stehen. Die BS2000-Meldungsklasse für BLS-Meldungen hängt davon ab, ob das Subsystem BLSSERV bereits geladen ist oder nicht. Vom Beginn einer Session bis zum Laden des Subsystems BLSSERV steht nur ein Teil der BLS-Funktionalität zur Verfügung und die BS2000-Meldungsklasse der Meldungen ist „BLP“. Danach ist die vollständige BLS-Funktionalität verfügbar und die BS2000-Meldungsklasse der Meldungen ist „BLS“. Beispielsweise wird bei der Zuweisung einer REP-Datei die Meldung BLP0990 oder BLS0990 ausgegeben, je nachdem, ob die Zuweisung vor oder nach dem Laden des Subsystems BLSSERV erfolgt. Die Bedeutung der Meldungen BLSxxxx und BLPxxxx mit gleicher Nummer ist in der Regel gleich. Es ist jedoch möglich, dass manche Inserts oder Erläuterungen voneinander abweichen.

2.5 Kommandos

Die nachfolgende Übersicht enthält alle Kommandos, die den DBL aufrufen. Die Kommandos sind im Handbuch „Kommandos“ [5] detailliert beschrieben.

Kommando	Funktion
CANCEL-PROGRAM	Programm entladen
LOAD-EXECUTABLE-PROGRAM	Programm laden
LOAD-PROGRAM	Programm laden
MODIFY-DBL-DEFAULTS	Voreinstellungen für DBL-Aufrufe ändern
RESET-DBL-DEFAULTS	Voreinstellungen für DBL-Aufrufe zurücksetzen
SELECT-PROGRAM-VERSION	Programmversion auswählen
SHOW-DBL-DEFAULTS	Voreinstellungen für DBL-Aufrufe anzeigen
START-EXECUTABLE-PROGRAM	Programm laden und starten
START-PROGRAM	Programm laden und starten

2.6 Makroaufrufe

Die nachfolgende Übersicht enthält alle Makroaufrufe des DBL. Die Makroaufrufe des DBL für frühere Versionen des BS2000 (bis BS2000 V9.5) werden nur noch auf Objektebene unterstützt. Die Makros sind im Handbuch „Makroaufrufe an den Ablaufteil“ [7] detailliert beschrieben.

Makro	**Funktion**
ASHARE	Shared Code in einen Common Memory Pool laden
BIND	Ladeeinheit binden und laden
DSHARE	Shared Code aus einem Common Memory Pool entladen
ETABIT *)	Ladeinformationstabelle aufbauen
ETABLE *)	Ladeinformation übergeben (Extended TABLE)
GETPRGV *)	Programmversion abfragen
ILEMGT *)	Verwaltung von ILEs (Indirect Linkage Entries)
ILEMIT *)	ILE-Tabelleneintrag erzeugen
LDSLICE	Slice laden
PINF	Globale Informationen über ein geladenes Programm ausgeben
SELPRGV *)	Programmversion auswählen
UNBIND	Entladen und Entbinden
VSVI1	Binde- und Ladeinformation ausgeben
*) Ab BS2000/OSD-BC V3.0. Diese Makros erfordern ASSEMBH.	

3 XS-Unterstützung des DBL

Dieses Kapitel wendet sich an Anwender, die den Adressraum oberhalb 16 Mbyte nutzen wollen. Zum Verständnis dieses Kapitels werden Grundbegriffe der XS-Programmierung vorausgesetzt.

3.1 Festlegen der vom Benutzer übergebenen Adressen

Die vom Benutzer in den Makroaufrufen ASHARE, BIND, DSHARE, LDSLICE, PINF, UNBIND und VSVI1 übergebenen Adressen bestimmt der DBL wie folgt:

– Beim Operanden INADDR des Makroaufrufs VSVI1 wird die Adresse immer als 31-Bit-Adresse festgelegt.

– Bei den übrigen Operanden in den Makroaufrufen richtet sich die Adresse nach dem Adressierungsmodus des aufrufenden Programms. Hat das aufrufende Programm den Adressierungsmodus AMODE 31, wird eine 31-Bit-Adresse, bei AMODE 24 eine 24-Bit-Adresse festgelegt.

3.2 Pseudo-RMODE eines Bindemoduls

Damit der dynamische Bindelader DBL ein Bindemodul als Einheit laden kann, ist es notwendig, die Lage des Moduls im Adressraum (oberhalb oder unterhalb 16 Mbyte) im gesamten festzulegen. Zu diesem Zweck wird für das Modul beim Binden ein Pseudo-RMODE festgelegt, der aus den Attributen RMODE der einzelnen CSECTs wie folgt bestimmt wird:

– Das Bindemodul erhält nur dann das Attribut (Pseudo-)RMODE ANY, wenn alle enthaltenen CSECTs das Attribut RMODE ANY besitzen.

– Enthält mindestens eine CSECT das Attribut RMODE 24, erhält auch das Modul das eingeschränkte Attribut (Pseudo-)RMODE 24.

Das Attribut AMODE wird durch den Programmabschnitt (CSECT) bestimmt, der die Einsprungstelle des Bindemoduls enthält.

3.3 Pseudo-RMODE eines LLM oder PAM-LLM

Folgende Erläuterungen gelten für PAM-LLMs ebenso wie für die bisherigen LLMs.

Damit der dynamische Bindelader DBL ein LLM als Einheit laden kann, ist es notwendig, die Lage des LLM im Adressraum (oberhalb oder unterhalb 16 Mbyte) im gesamten festzulegen. Zu diesem Zweck wird für das LLM beim Binden ein Pseudo-RMODE festgelegt. Man unterscheidet folgende zwei Möglichkeiten:

1. Für ein LLM, das aus einem Einzel-Slice (SINGLE) oder aus benutzerdefinierten Slices (BY-USER) besteht, bestimmt der BINDER den Pseudo-RMODE aus den Attributen RMODE der einzelnen CSECTs und eventuell auch aus den Attributen der COMMON-Bereiche wie folgt:

   – Das LLM erhält nur dann das Attribut (Pseudo-)RMODE ANY, wenn alle enthaltenen CSECTs das Attribut RMODE ANY besitzen.

   – Enthält mindestens eine CSECT das Attribut RMODE 24, erhält auch das LLM das eingeschränkte Attribut (Pseudo-)RMODE 24.

   Ein solches LLM kann nur als Ganzes unterhalb oder oberhalb 16 Mbyte geladen werden.

2. Für ein LLM, das aus Slices besteht, die aus Attributen gebildet wurden (BY-ATTRIBUTES), bestimmt der BINDER für jede Slice einen eigenen Pseudo-RMODE, auch wenn das Attribut RMODE nicht zum Bilden der Slices benutzt wurde. Der Pseudo-RMODE für jede Slice wird aus den Attributen RMODE der einzelnen CSECTs bestimmt wie unter Punkt 1 beschrieben.

   Ein solches LLM kann teilweise unterhalb und teilweise oberhalb 16 Mbyte geladen werden.

3.4 Festlegen der Ladeadresse

Die Ladeadresse und den Adressierungsmodus einer Ladeeinheit oberhalb 16 Mbyte legt in den Kommandos START/LOAD-EXECUTABLE-PROGRAM bzw. START/LOAD-PROGRAM und im Makroaufruf BIND der Operand PROGRAM-MODE=24/ANY fest.

Der DBL bestimmt die Ladeadresse und den Adressierungsmodus der Ladeeinheit

- aus dem Operanden PROGRAM-MODE und
- aus den Attributen AMODE und RMODE der CSECTs der Ladeeinheit.
- Wenn AMODE-CHECK=*ADVANCED angegeben ist, wird zusätzlich das AMODE-Attribut der Ladeeinheit zur Bestimmung von Ladeadresse und Adressierungsmodus der Ladeeinheit herangezogen.

3.4.1 Kommandos LOAD- und START-EXECUTABLE-PROGRAM (bzw. LOAD- und START-PROGRAM)

PROGRAM-MODE=24 (Standard)

Der DBL wertet diesen Operanden wie folgt aus:

- Die Ladeeinheit wird unterhalb 16 Mbyte geladen.
- Externverweise werden nur mit CSECTs oder ENTRYs befriedigt, die unterhalb 16 Mbyte liegen.
- Der 24-Bit-Adressierungsmodus wird eingestellt.
- Das Laden der Ladeeinheit wird mit einer Fehlermeldung abgebrochen, wenn eine CSECT mit dem Attribut AMODE 31 enthalten ist.

PROGRAM-MODE=ANY

Jedes Modul der Ladeeinheit kann oberhalb oder unterhalb 16 Mbyte geladen werden. Die Ladeadresse ist abhängig von den RMODE-Attributen der CSECTs des Moduls.

Enthält das Modul mehrere CSECTs, legt der DBL einen (Pseudo-)RMODE fest, der aus den Attributen RMODE der einzelnen CSECTs wie folgt bestimmt wird:

– Das Modul erhält nur dann das Attribut Pseudo-RMODE ANY, wenn alle enthaltenen CSECTs das Attribut RMODE ANY besitzen.

– Enthält mindestens eine CSECT das Attribut RMODE 24, erhält auch das Modul das Attribut Pseudo-RMODE 24.

– Wenn AMODE-CHECK=*ADVANCED angegeben ist, wird zusätzlich das AMODE-Attribut der Ladeeinheit zur Bestimmung von (Pseudo-)RMODE und Ladeadresse der Ladeeinheit herangezogen.

Bei AMODE-CHECK=*STD hängt Ladeadresse wie folgt vom (Pseudo-)RMODE ab:

(Pseudo-)RMODE	
24	Alle Module der Ladeeinheit werden unterhalb der 16-Mbyte-Grenze geladen
ANY	Alle Module der Ladeeinheit werden oberhalb der 16-Mbyte-Grenze geladen

Der Adressierungsmodus wird bei AMODE ANY durch die Lage der Einsprungstelle bestimmt. Liegt die Einsprungstelle unterhalb 16 Mbyte, wird der 24-Bit-Adressierungsmodus eingestellt, bei Lage oberhalb 16 Mbyte der 31-Bit-Adressierungsmodus.

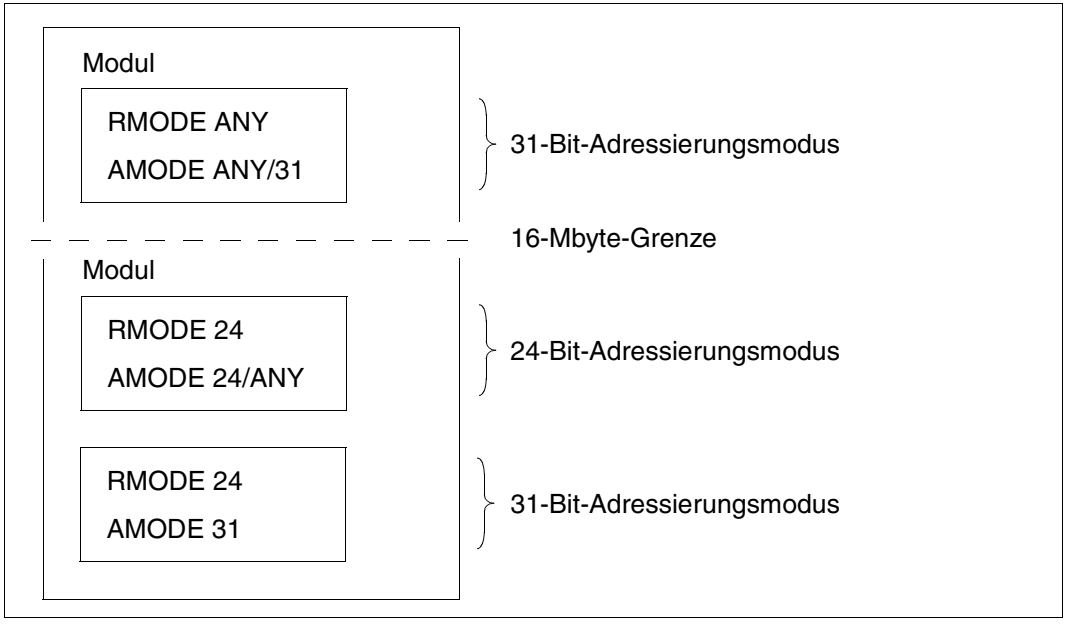


Bild 7: Adressierungsmodus bei START-/LOAD-EXECUTABLE-PROGRAM

Bei AMODE-CHECK=*ADVANCED gilt:

Wenn nach dem Laden des ersten Moduls der Ladeeinheit als AMODE-Attribut 24 ermittelt wurde, werden alle weiteren Module der Ladeeinheit unterhalb 16 Mbyte geladen.

3.4.2 BIND-Makroaufruf

Beim BIND-Makroaufruf bestimmt der dynamische Bindelader die Ladeadresse und den Adressierungsmodus aus den Operanden PROGMOD=ANY/24 und BRANCH=NO/YES.

PROGMOD=ANY (Standard)

Jedes Modul der Ladeeinheit kann oberhalb oder unterhalb 16 Mbyte geladen werden. Die Ladeadresse ist abhängig von den RMODE-Attributen der CSECTs des Moduls und vom Adressierungsmodus des Programms zum Zeitpunkt des BIND-Aufrufs.

Wenn AMODE-CHECK=*ADVANCED angegeben ist, wird zusätzlich das AMODE-Attribut der Ladeeinheit herangezogen um die Ladeadresse der Module der Ladeeinheit zu bestimmen.

Enthält das Modul mehrere CSECTs, legt der DBL einen (Pseudo-)RMODE fest, der aus den RMODE-Attributen der einzelnen CSECTs wie folgt bestimmt wird:

– Das Bindemodul erhält nur dann das Attribut Pseudo-RMODE ANY, wenn alle enthaltenen CSECTs das Attribut RMODE ANY besitzen.

– Enthält mindestens eine CSECT das Attribut RMODE 24, erhält auch das Modul das Attribut Pseudo-RMODE 24.

Bei AMODE-CHECK=*STD wird die Ladeadresse folgendermaßen bestimmt:

– Wenn der Addressierungsmodus des Programms beim BIND-Aufruf nicht 31 ist, werden alle Module der Ladeeinheit unterhalb der 16-Mbyte-Grenze geladen.

– Andernfalls werden die Module abhänging von ihren (Pseudo-)RMODE unter- oder oberhalb der 16-Mbyte-Grenze geladen

Der DBL lädt also jedes Modul nur oberhalb 16 Mbyte, wenn das Modul das Attribut RMODE ANY hat und beim BIND-Aufruf der 31-Bit-Adressierungsmodus eingestellt war.

In allen anderen Fällen wird das Modul unterhalb 16 Mbyte geladen.

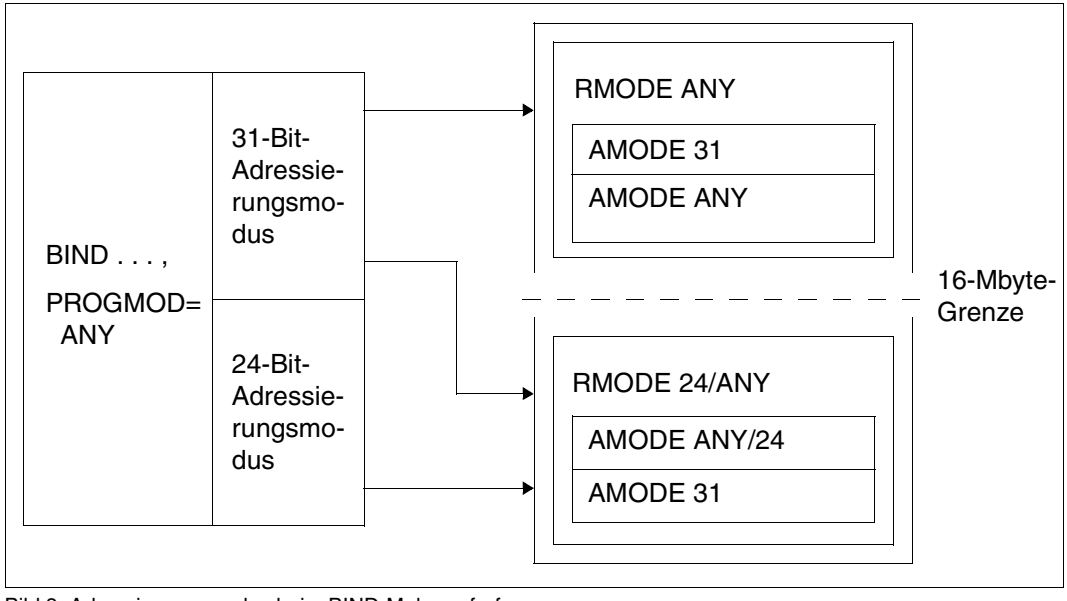


Bild 8: Adressierungsmodus beim BIND-Makroaufruf

Bei AMODE-CHECK=*ADVANCED gilt:

Wenn nach dem Laden des ersten Moduls der Ladeeinheit als AMODE-Attribut 24 ermittelt wurde, werden alle weiteren Module der Ladeeinheit unterhalb 16 Mbyte geladen.

PROGMOD=24

Der DBL wertet den Operanden PROGMOD=24 wie folgt aus:

– Alle Module der Ladeeinheit werden unterhalb 16 Mbyte geladen.

– Externverweise werden nur dann befriedigt, wenn sie eine Adresse unterhalb 16 Mbyte bezeichnen.

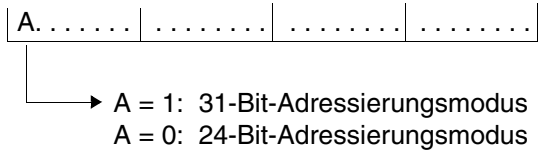
– Ist gleichzeitig der Operand BRANCH=YES angegeben, wird der 24-Bit-Adressierungsmodus eingestellt.

– Das Laden der Ladeeinheit wird mit Fehlermeldung abgebrochen, wenn die Ladeeinheit ein Modul mit dem Attribut AMODE 31 enthält.

BRANCH=NO (Standard)

Nach dem Laden des Moduls wird die Steuerung an das aufrufende Programm zurückgegeben. Der aktuelle Adressierungsmodus wird nicht verändert. Nach dem BIND-Aufruf zeigt das höchstwertige Bit des Feldes, das mit dem Operanden SYMBLAD definiert wurde an, welcher Adressierungsmodus bei Ausführung des Moduls eingestellt werden muss.

Feld SYMBLAD



Diese Information wird nach folgenden Kriterien bestimmt:

Lage der Einsprungstelle	AMODE (CSECT)		
	24	31	ANY
≤16 Mbyte	24-Bit-Modus	31-Bit-Modus	je nach AMODE bei BIND-Aufruf
> 16 Mbyte	31-Bit-Modus	31-Bit-Modus	31-Bit-Modus

BRANCH=YES

Es wird zuerst das nachgeladene Modul (ab Einsprungstelle) abgearbeitet. Der Adressierungsmodus wird bestimmt durch die Angabe für den Operanden PROGMOD in Verbindung mit dem Attribut AMODE des ersten Programmabschnitts (CSECT) des Moduls und unter Beachtung der Einsprungstelle und dem eingestellten Adressierungsmodus zum Zeitpunkt des BIND-Aufrufs. Der Adressierungsmodus wird intern nach folgenden Kriterien eingestellt:

PROGMOD	AMODE (CSECT)		
	24	31	ANY
24	24-Bit-Modus	unzulässig	24-Bit-Modus
ANY	24-Bit-Modus	31-Bit-Modus	31-Bit-Modus, wenn Einsprungstelle ≥ 16 Mbyte. Wie BIND-Aufruf, wenn Einsprungstelle < 16 Mbyte.

3.4.3 Behandlung von COMMON-Bereichen

Bei der Behandlung von COMMON-Bereichen ist Folgendes zu beachten:

– Wenn mindestens ein Modul der Ladeeinheit unterhalb 16 Mbyte geladen wird, werden sämtliche COMMON-Bereiche dieser Ladeeinheit unterhalb 16 Mbyte geladen.

– Die Ladeadresse des COMMON-Bereichs ist abhängig vom Operanden PROGMOD und von den Attributen RMODE der gleichnamigen COMMON-Definitionen der Ladeeinheit. Eventuell kann die Ladeadresse auch von den Attributen RMODE des gleichnamigen Programmabschnitts (CSECT) abhängen.

Der DBL bestimmt die Ladeadresse des COMMON-Bereichs in Abhängigkeit vom Operanden PROG-MOD wie folgt:

PROGMOD	Ladeadresse des COMMON-Bereichs
24	< 16 Mbyte
31	≥ 16 Mbyte wenn alle Attribute RMODE ANY sind < 16 Mbyte wenn mindestens ein Attribut RMODE 24 ist

3.5 Befriedigen von Externverweisen oberhalb 16 Mbyte

In BLSSERV bis einschließlich V2.4A gelten folgende Regeln beim Befriedigen von Externverweisen:

1. **PROGRAM-MODE=24** beim Lade-Kommando oder
   PROGMOD=24 beim BIND-Makro:

   Externverweise werden ausschließlich mit Symbolen befriedigt, die unterhalb 16 Mbyte geladen sind.

2. **PROGRAM-MODE=*ANY** beim Lade-Kommando oder
   PROGMOD=ANY beim BIND-Makro:

   Externverweise werden mit Symbolen befriedigt, die unterhalb oder oberhalb von 16 Mbyte geladen sind.

Wird eine Ladeeinheit mit PROGRAM-MODE=*ANY (bzw. PROGMOD=ANY) geladen, können Externverweise nach diesen Regeln auch mit Symbolen befriedigt werden, die oberhalb 16 Mbyte geladen sind. Muss diese Ladeeinheit jedoch (z.B. wegen des AMODE-Attributs des Aufrufers) im AMODE 24 ablaufen, führt dies zu Fehlern.

Um solche Inkonsistenzen zu vermeiden, können beim Laden mit BLSSERV ab V2.5A zusätzliche AMODE-Prüfungen durchgeführt werden. Mit dem Operanden AMODE-CHECK im Kommando START-/LOAD-EXECUTABLE-PROGRAM (oder AMODCHK beim BIND-Makro) wird gesteuert, ob diese zusätzlichen Prüfungen durchgeführt werden.

Bei AMODE-CHECK=*STD (Default-Wert) ist das Verhalten beim Laden und Befriedigen von Externverweisen kompatibel zu den Vorgängerversionen von BLSSERV. Ob Externverweise mit Modulen oberhalb 16 Mbyte befriedigt werden können oder nicht, entscheidet BLSSERV auf Grund des Operanden PROGRAM-MODE, des Pseudo-RMODE der geladenen Objekte und des AMODE-Attributs des rufenden Programms.
Damit können Inkosistenzen wie die oben beschriebene auftreten.

Bei AMODE-CHECK=*ADVANCED wird zusätzlich das AMODE-Attribut der Ladeeinheit berücksichtigt, um zu entscheiden, ob Externverweise auch mit Modulen oberhalb 16 Mbyte befriedigt werden können. Wenn also eine Ladeeinheit nach dem Laden ihres ersten Moduls ein AMODE-ATTRIBUT von 24 erhält, muss der Rest der Ladeeinheit unterhalb 16 Mbyte geladen werden und Externverweise dieser Ladeeinheit werden nur mit Symbolen befriedigt, die unterhalb von 16 MByte geladen werden können.

4 Der Lader ELDE

4.1 Aufgaben des Laders

Der statische Lader ELDE ist ebenfalls ein Teil des Subsystems BLSSERV. Er hat die Aufgabe, ein Programm (Lademodul) zu laden, das vom Binder TSOSLNK gebunden und in einer Programmdatei oder als Bibliothekselement (Typ C) in einer Programmbibliothek gespeichert wurde. Dabei hat der Lader Relativierungsinformationen auszuwerten, falls das Lademodul noch kein Speicherabbildformat besitzt (siehe COREIM=N in der PROGRAM-Anweisung des Binders TSOSLNK).

4.2 Aufruf des Laders

Der Lader ELDE kann vom Benutzer mit folgenden Kommandos und Makroaufrufen aufgerufen werden:

- Mit den Kommandos START-EXECUTABLE-PROGRAM oder LOAD-EXECUTABLE-PROGRAM, falls der Operand FROM-FILE mit einem Dateinamen angegeben wurde oder wenn der zu ladende Modul ein Lademodul ist.
- Mit den Kommandos START-PROGRAM oder LOAD-PROGRAM, falls der Operand FROM-FILE mit einem Dateinamen oder mit *PHASE angegeben wurde.
  Die Kommandos START-PROGRAM und LOAD-PROGRAM werden nur noch aus Kompatibilitätsgründen unterstützt. Für Neuanwendungen sollten die Kommandos START-EXECUTABLE-PROGRAM oder LOAD-EXECUTABLE-PROGRAM verwendet werden.
- Mit dem Makroaufruf LPOV, der Überlagerungssegmente nachlädt. Die Makroaufrufe CALL und SEGLD führen indirekt zur Ausführung des LPOV-Makros.

Die Kommandos sind im Handbuch „Kommandos“ [5] detailliert beschrieben.

5 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 Anwender eine Umstiegshilfe geben.

5.1 Bisherige und neue Begriffe

In diesem Abschnitt sind Begriffe des alten und des aktuellen Binder-Lader-Systems gegenübergestellt (siehe auch Kapitel „Fachwörter“ ab Seite 103).

5.1.1 Bisherige Begriffe

Bindemodul (OM)
Object Module (OM)
: Ladbare Einheit, die durch Übersetzen eines Quellprogramms mit Hilfe eines Sprachübersetzers erzeugt wird.

Vorgebundenes Bindemodul (Großmodul)
: Ladbare Einheit, die vom Binder TSOSLNK aus einzelnen Bindemodulen (OM) gebunden wird. Hat das gleiche Format wie ein Bindemodul (OM).

Bindemodulbibliothek (OML)
Object Module Library (OML)
: PAM-Datei, die Bindemodule als Bibliothekselemente enthält. Bindemodulbibliotheken werden ab LMS V2.0A (SDF-Format) nicht mehr unterstützt. Sie sind durch *Programmbibliotheken* zu ersetzen.

EAM-Bindemoduldatei
EAM Object Module File
: Temporäre System-Bindemodulbibliothek, in die von einem Compiler Bindemodule (OMs) oder vom Binder TSOSLNK Großmodule abgespeichert werden.

Lademodul (Phase)

Ablauffähige Einheit, die vom Binder TSOSLNK aus Bindemodulen (OMs) gebunden und in eine katalogisierte Programmdatei oder als Bibliothekselement vom Elementtyp C in einer Programmbibliothek gespeichert wird.

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.

TSOSLNK

Binder des bisherigen Binder-Lader-Systems. TSOSLNK bindet:

- entweder ein oder mehrere Bindemodule (OMs) zu einem ablauffähigen Programm (Lademodul) oder
- mehrere Bindemodule (OMs) zu einem einzigen vorgebundenen Bindemodul (Großmodul) zusammen.

DLL

Dynamischer Bindelader des bisherigen Binder-Lader-Systems. Bindet Bindemodule (OMs) zu einem ablauffähigen Programm und lädt dieses.

ELDE

Statischer Lader des bisherigen Binder-Lader-Systems. Lädt ein Programm (Lademodul), das vom Binder TSOSLNK gebunden wurde.

5.1.2 Neue Begriffe

Bindelademodul (LLM)
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 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 Binder-Lader-Systems.

BINDER

Binder des neuen Binder-Lader-Systems. Bindet Module zu einem Bindelademodul (LLM).

DBL

Dynamischer Bindelader des neuen Binder-Lader-Systems. Bindet Module zu einer Ladeeinheit und lädt diese.

5.2 Merkmale des aktuellen Binder-Lader-Systems

Das Konzept für das seit BS2000 Version 10 verfügbare System Binder-Lader-Starter (BLS) unterscheidet sich wesentlich vom Konzept für das alte Binder-Lader-System. Im Folgenden sind die Hauptmerkmale des aktuellen Konzepts zusammengestellt.

5.2.1 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.

5.2.2 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 Namenskollision, 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.

5.2.3 DBL und BINDER

- Eines der Hauptmerkmale des aktuellen Binder-Lader-Starter (BLS) ist die weitgehende Harmonisierung zwischen BINDER und DBL. Das bedeutet:
  - Unechte Aufrufe wie der Makroaufruf TABLE, die bisher zur Verbindung zwischen dem statischen Lader ELDE und dem bisherigen dynamischen Bindelader DLL verwendet wurden, werden durch Verwendung von LLMs ersetzt. 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. Der DBL lädt die PRIVATE-Slice in den Klasse-6-Speicher des Benutzers. Die PUBLIC-Slice erklärt der Benutzer als gemeinsam benutzbar, indem er sie mit dem DBL-Makro ASHARE in einen Common Memory Pool lädt.

5.3 Migration vom dynamischen Bindelader DLL zum DBL

5.3.1 Betriebsmodi RUN-MODE=*STD und RUN-MODE=*ADVANCED

Der dynamische Bindelader DBL arbeitet in zwei verschiedenen Betriebsmodi. Der Benutzer wählt den Betriebsmodus, in dem der DBL ablaufen soll, mit dem Operanden RUN-MODE in den Kommandos START-PROGRAM und LOAD-PROGRAM aus.

Bei Verwendung der Kommandos START/LOAD-EXECUTABLE-PROGRAM arbeitet der DBL immer im Betriebsmodus RUN-MODE=*ADVANCED.

*Betriebsmodus RUN-MODE=*STD*
In diesem Betriebsmodus arbeitet der DBL voll kompatibel zum DLL des alten Binder-Lader-Systems. Die neuen Funktionen des DBL können in diesem Betriebsmodus nicht angewendet werden. Der Betriebsmodus RUN-MODE=*STD ist standardmäßig voreingestellt.

*Betriebsmodus RUN-MODE=*ADVANCED*
In diesem Betriebsmodus unterstützt der DBL die seit BS2000 V10 eingeführten Funktionen. Diese Funktionen können zu Inkompatibilitäten gegenüber dem DLL des ehemaligen Binder-Lader-Systems führen.

5.3.2 Inkompatibilitäten bei RUN-MODE=*ADVANCED

Suchen von Modulen in Bibliotheken

Beim Suchen von Modulen in Programmbibliotheken steuert der Operand RUN-MODE die Auswahl der Module wie folgt:

– bei RUN-MODE=*STD werden nur Bindemodule (OMs) gesucht. LLMs werden ignoriert.

– Bei RUN-MODE=*ADVANCED werden LLMs und Bindemodule (OMs) miteinander verglichen. Dabei hat der Elementname eines LLM (Elementtyp L) oder ein Symbolname in einem LLM Vorrang vor dem Elementnamen eines Bindemoduls (OM) (Elementtyp R) oder vor einem Symbolnamen in einem OM.

– Bei den Kommandos START/LOAD-EXECUTABLE-PROGRAM kann die Suchreihenfolge über den Operanden TYPE gesteuert werden, wobei die Reihenfolge L,C,R voreingestellt ist.

Diese Suchstrategie begünstigt die Migration vom Bindemodul (OM) zum Bindelademodul (LLM). Kompatibilität besteht dann, wenn kein LLM gefunden wird, das den gleichen Elementnamen wie ein OM hat.

Suchen in alternativen Bibliotheken

Standardmäßig wird bei RUN-MODE=*ADVANCED die Tasklib nicht mehr berücksichtigt. Das wahlweise Zuschalten der Tasklib ist nur bei Verwendung der Kommandos START/LOAD-EXECUTABLE-PROGRAM möglich.

Behandlung von Namenskonflikten

Die Behandlung von Namenskonflikten ist vom gewählten Betriebsmodus (RUN-MODE=*STD oder RUN-MODE=*ADVANCED) wie folgt abhängig: (siehe auch Seite 32ff)

– bei RUN-MODE=*STD werden Namenskonflikte zwischen CSECTs immer entdeckt, unabhängig davon, ob die Symbole in einem Modul maskiert sind oder nicht. Namenskonflikte zwischen maskierten oder nicht maskierten ENTRYs werden immer akzeptiert. Nur der erste ENTRY wird zur Befriedigung von Externverweisen verwendet. Der Benutzer hat keine Möglichkeit, die Behandlung von Namenskonflikten zu steuern.

– bei RUN-MODE=*ADVANCED werden Namenskonflikte nur entdeckt, wenn die Symbole in einem Modul *nicht* maskiert sind. Der Benutzer wird über jeden Namenskonflikt, bei dem Fehler auftreten können, durch eine Meldung benachrichtigt. Mit dem Operanden NAME-COLLISION im Ladeaufruf hat der Benutzer die Möglichkeit, die Behandlung von Namenskonflikten zu steuern.

5.3.3 Laden von LLMs bei RUN-MODE=*STD

Viele Anwendungen, die nicht mehr weiterentwickelt werden, laden mit dem LINK-Makro externe Laufzeitkomponenten nach. Die Migration solcher Laufzeitkomponenten vom Bindemodul zum Bindelademodul wäre demnach nicht ohne den Verlust der Kompatibilität möglich, denn der LINK-Makro läuft im Standard-RUN-MODE, in dem LLMs bisher nicht verarbeitet werden konnten.

Damit die Migration ohne Kompatibilitätsverlust möglich ist, wurde der Standard-RUN-MODE des DBL ab BS2000/OSD-BC V3.0 so erweitert, dass LLMs unter bestimmten Bedingungen ebenfalls verarbeitet werden können. Im Prinzip muss ein solches LLM die Eigenschaften eines Bindemoduls haben, das heißt:

- es darf keine benutzerdefinierten Slices haben
- die Namen aller enthaltenen OMs dürfen maximal 8 Zeichen lang sein
- kein Symbolname darf länger als 8 Zeichen sein
- es darf keinen unbefriedigten Externverweis auf ein Symbol geben, dessen Name länger als 8 Zeichen ist
- vom privaten LLM-Teil darf es keinen Verweis auf ein PUBLIC-Symbol geben, dessen Name länger als 8 Zeichen ist
- das LLM muss die logische Strukturinformation enthalten

Auch LLMs, die mit der BINDER-Anweisung MERGE-MODULES bearbeitet wurden, können mit dem LINK-Makro geladen werden. Durch das Mischen eines LLMs können nicht benötigte Symbole verborgen werden. Das gemischte LLM bzw. Sub-LLM enthält nur noch ein Großmodul mit einer einzigen CSECT.

Durch die Unterschiede zwischen dem Standard-RUN-MODE und dem ADVANCED-RUN-MODE kann die Menge der mit LINK oder BIND nachgeladenen Module sehr unterschiedlich sein. Bei der Migration der Laufzeitkomponenten muss deshalb sehr sorgfältig darauf geachtet werden, dass sowohl mit dem BIND-Makro als auch mit dem LINK-Makro ein korrektes Ladeverhalten erreicht wird.

5.3.4 Laden eines LLM mit benutzerdefinierten Slices

Wird ein LLM geladen, das aus benutzerdefinierten Slices aufgebaut ist, ist Folgendes zu beachten:

- die Bibliothek und das Element bleiben eröffnet.
- wird im Ladeaufruf für die Bibliothek nicht der Operand LINK-NAME angegeben, bleibt die Bibliothek für alle weiteren Ladeaufrufe gesperrt.

5.3.5 Migration der bisherigen Makros zu den neuen Makros

Im Folgenden ist beschrieben, durch welche Makros die bis BS2000 V10.0 verfügbaren Makros ersetzt wurden.

Makro LINK

Die Funktionen des Makros LINK werden durch den gleichwertigen Makro BIND realisiert. Sie sind eine Untermenge der Funktionen des Makros BIND. Zu beachten ist, dass PROGMOD=ANY beim BIND-Makro Default-Wert ist.
Um die Migration zu erleichtern, kann der LINK-Makro ab BS2000/OSD-BC V3.0 unter bestimmten Bedingungen auch LLMs laden (siehe Seite 93).

Makro LPOV

Der Makro LPOV wird für LLMs nicht unterstützt. Wenn vom BINDER ein LLM mit benutzerdefinierten Slices erzeugt wird, darf in keinem der eingefügten Bindemodule (OMs) der Makro LPOV enthalten sein. Bindemodule (OMs), die den Makro LPOV enthalten, können nur vom Binder TSOSLNK zu einem Programm (Lademodul) gebunden werden.

Makros PBUNLD und UNLOD

Die Funktionen der Makros PBUNLD und UNLOD werden durch den gleichwertigen Makro UNBIND realisiert. Der Makro UNBIND ist voll kompatibel zu den Makroaufrufen PBUNLD und UNLOD, wenn im Makroaufruf UNBIND der Operand UNLINK=NO angegeben wird (Standardwert). In dem entladenen Objekt werden dann CSECTs und ENTRYS *nicht* entbunden, d.h. befriedigte Externverweise zu diesen Symbolen bleiben befriedigt. Wird im Makroaufruf UNBIND der Operand UNLINK=YES angegeben, werden befriedigte Externverweise entbunden, d.h sie werden als nicht befriedigte Externverweise behandelt. Bei den Makros PBUNLD und UNLOD ist diese Funktion nicht enthalten.

Makro VSVI

Die Funktionen des Makros VSVI werden durch den gleichwertigen Makro VSVI1 realisiert. Die Ausgabeinformation ist abhängig vom gewählten Betriebsmodus (Operand RUNMOD).

Im Betriebsmodus RUNMOD=STD ist die Ausgabeinformation des Makros VSVI1 voll kompatibel zu der Information, die vom Makro VSVI ausgegeben wurde. Der Makro VSVI1 verarbeitet in diesem Betriebsmodus nur maximal 8 Zeichen lange Symbolnamen und maximal 16 Zeichen lange Kontextnamen. Längere Namen werden auf 8 Zeichen oder 16 Zeichen gekürzt.
Die Bezugsgröße für eine ausgegebene Information ist ein Eintrag in den DBL-Tabellen mit einer *festen* Länge von 36 byte.

Im Betriebsmodus RUNMOD=ADV verarbeitet der Makro VSVI1 maximal 32 Zeichen lange Namen.
Die Bezugsgröße für eine ausgegebene Information ist ein Eintrag in den DBL-Tabellen mit einer *variablen* Länge. Die Länge ist abhängig von der Länge der Namen und der Art der gewünschten Information.

Makro ITABL

Die Funktionen des Makros ITABL sind eine Untermenge der Funktionen des Makros VSVI und können daher durch die Funktionen des Makros VSVI1 ersetzt werden.

Makro TABLE

Die Funktionen des Makros TABLE sind in erweiterter Form mit den Makros ETABLE und ETABIT realisiert, die ab BS2000/OSD-BC V3.0 verfügbar sind. Zu beachten ist, dass Namenskonflikte bei ETABLE anders behandelt werden als bei TABLE.

5.4 Migration vom statischen Lader ELDE zum DBL

Zum Aufruf des Laders ELDE sollten grundsätzlich nur noch die neuen Kommandos START/LOAD-EXECUTABLE-PROGRAM verwendet werden. Falls das Lademodul als C-Element in einer Bibliothek vorliegt, sollte dies ohne explizite Typ-Angebe erfolgen:

```
START-EXE FROM-FILE=*LIB-ELEM(LIB=bibliothek,ELEM=element)
    oder
LOAD-EXE FROM-FILE=*LIB-ELEM(LIB=bibliothek,ELEM=element)
```

Die Standardeinstellung TYPE=(L,C,R) begünstigt eine eventuelle spätere Migration zum Bindelademodul (LLM). Sofern keine explizite Vereinbarung für DBL-Parameter erforderlich ist, muss das Aufrufkommando bei der Migration gar nicht mehr geändert werden; es genügt, ein entsprechendes L-Element in die Bibliothek einzutragen.

5.5 Migration von Programmen zu PAM-LLMs

PAM-LLMs unterscheiden sich von (herkömmlichen) LLMs durch den „Behälter“ in dem sie abgespeichert werden:

– ein PAM-LLM ist in einer PAM-Datei abgespeichert

– ein LLM ist in einer Programmbibliothek als Element mit dem Typ L abgespeichert.

In ihrer internen Struktur sind LLMs und PAM-LLMs identisch. Daher gelten die in diesem Kapitel enthaltenen Erläuterungen zur Migration von Programmen zu LLMs genauso für die Migration zu PAM-LLMs.

Der Vorteil einer Migration von Programmen zu PAM-LLMs statt LLMs besteht darin, dass die Kommandos für den Aufruf nicht geändert werden müssen: Die Kommandos `/EXEC HUGO`, `/START-PROGRAM HUGO` oder `/START-EXECUTABLE-PROGRAM HUGO` starten das Objekt, das sich in der Datei HUGO befindet, unabhängig davon ob es sich um ein Programm oder ein PAM-LLM handelt.

6 Performantes Laden von Programmen/Produkten

In diesem Kapitel werden Maßnahmen beschrieben, die der Anwender/Produktersteller berücksichtigen sollte, damit die Ladezeit eines Programmes/Produktes optimiert werden kann.

Es wird dabei i. d. R. davon ausgegangen, dass die Programme bzw. Module in PLAM-Bibliotheken abgelegt sind. Das Binden der Programme/Module erfolgt deshalb mit BINDER, das Laden mit dem Kommando START-EXECUTABLE-PROGRAM. Die Ausführungen beschränken sich auf das Arbeiten mit LLMs.

6.1 Allgemeine Hinweise zur Beschleunigung des Ladevorganges

- Alle beteiligten PLAM-Bibliotheken sollten so gut wie möglich organisiert sein, d. h.:
  - sie sollten keine überflüssigen LLMs enthalten
  - sie sollten vollständig reorganisiert sein, sodass die enthaltenen LLMs nicht in unnötig viele Extents aufgeteilt sind
  - die Bibliothek selbst sollte aus so wenig Extents wie möglich bestehen
  - die Bibliothek sollte eventuell mit (STD,2) eingerichtet sein

  Mit all diesen Maßnahmen wird die Anzahl der Ein-/Ausgaben reduziert und damit die Laufzeit verbessert; der Effekt ist natürlich abhängig von der Ausgangsstruktur. Eine mit (STD,2) eingerichtete Bibliothek wird um etwa 1 KByte pro Element größer sein als die Original-Bibliothek; der Ein-/Ausgabe-Einsparungseffekt wird um so größer sein, je mehr (nicht zu kleine) Elemente die Bibliothek hat. Die Laufzeitverbesserung kann über 10% betragen.
- Falls Module mit dem Attribut READ-ONLY versehen sind, sollte ein Slice für Read-Only Module und ein Slice für Read-Write-Module erzeugt werden. Dies beschleunigt den Ladevorgang sowohl durch geringeren CPU-Bedarf als auch durch Einsparung von Ein-/Ausgaben.

- Die Anzahl der Bibliothekszugriffe wird reduziert, wenn alle Symbole, die für spätere BINDER-Läufe nicht mehr benötigt werden, invisible gesetzt werden; BINDER-Anweisung:

```
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=...,SYMBOL-TYPE=...,
                           SCOPE=...,VISIBLE=*NO,...
```

- Falls PUBLIC/PRIVATE-Slices verwendet werden, sollte Format 2 zur CPU-Reduzierung von BLS und DSSM verwendet werden (siehe BINDER-Anweisung //SAVE-LLM ... FROM-BS2000-VERSIONS=*FROM-V11).

  Mit dem Operanden SUBSYSTEM-ENTRIES bei der BINDER-Anweisung //START-LLM-CREATION kann ein symbolischer Name für die Verknüpfungen zwischen dem PRIVATE- und dem PUBLIC-Slice angegeben werden. Dieser Name muss auch im DSSM-Katalog definiert sein. (Mit dem alten Format wird für jede Verbindung DSSM aufgerufen.)

- Es ist darauf zu achten, dass in der Parameterdatei von DSSM der Parameter LOGGING *nicht* auf ON gesetzt ist (Default: OFF). ON würde dazu führen, dass beim Laden von Subsystemen DSSM-spezifische Daten protokolliert werden.

6.2 Strukturelle Maßnahmen zur Reduzierung des Betriebsmittelbedarfs

Es ist leider nur selten möglich, den Effekt jeder einzelnen der folgenden Maßnahmen genauer zu beziffern. Er hängt zu stark von der Struktur der Ladeobjekte, Subsysteme und Bibliotheken ab. Man kann jedoch in der Regel davon ausgehen, dass pro Maßnahme ein Mindestgewinn von 10% eintritt.

Bibliotheken reduzieren bzw. deren Inhalt optimieren

- Mischen der alternativen Bibliotheken (hier Altlibs genannt)

  Durch Mischen der Altlibs z.B. in eine einzige Bibliothek (dies könnte sogar die spezifizierte Lade-Bibliothek sein) werden die Suchvorgänge in BLS/PLAM stark reduziert, wodurch z. T. deutlich mehr als 10% an Ein-/Ausgaben und CPU-Zeit eingespart werden.

  Das Mischen ist im Allgemeinen nur dann anzuraten, wenn keine namensgleichen CSECTs oder ENTRIES existieren.

- Module aus Bibliotheken fest einbinden

  Falls sich die Inhalte der Altlibs und der spezifizierten Bibliothek nie bzw. nur selten ändern, empfiehlt es sich eventuell, die Module in das Ladeobjekt fest einzubinden. Dadurch wird der CPU-Bedarf beim Laden verringert. Allerdings wird durch das statische Binden das Ladeobjekt größer und damit werden in der Regel mehr Ein-/Ausgaben durchgeführt, was wiederum zur Verlängerung der Laufzeit führt.
  Diese Maßnahme ist demnach günstig, wenn nur kleine Module betroffen sind und wenn die Altlib(s) besonders viele Entries enthalten.

  Wird diese Variante der Performance-Steigerung gewählt, so ist natürlich darauf zu achten, dass der Bindevorgang bei jeder Altlib-Änderung erneut durchzuführen ist.

- Mischform

  Falls sich z.B. der Inhalt von nur einer der beteiligten Altlibs häufig ändert, so empfiehlt es sich eventuell, die Mischform der beiden Methoden zu wählen.

Nachladen aus Shared-Code einschränken

- Kein Nachladen/Binden aus Shared-Code

  Viele Programme benötigen kein Nachladen/Linken aus einem Subsystem/Shared-Programm oder User-Shared-Memory. Diese Programme sollten wie folgt gestartet werden:
  `/START-EXECUTABLE-PROGRAM ...,RESOLUTION=(SHARE-SCOPE=*NONE),...`
  Dadurch wird verhindert, dass BLS die Subsysteme nach dem passenden Entry durchsucht, und es ergibt sich eine z. T. erhebliche Einsparung an CPU-Zeit.

- Nur aus System-Memory nachladen

  Dieser Fall ist der Default-Fall; es gibt keine Möglichkeit, die Suche nach dem Entry auf bestimmte Subsysteme oder nur auf Shared-Programme zu beschränken.

- Nur aus User-Shared-Memory nachladen

  Beim Kommando START-EXECUTABLE-PROGRAM-kann man mit dem Parameter
  `RESOLUTION(...,SHARE-SCOPE=*MEMORY-POOL(...)...)`
  die Beschränkung auf User-Shared-Memory definieren. Es ist dabei zusätzlich möglich, sich auf ausgewählte Memory-Pools zu beschränken.

- Vorladen des User-Shared-Memory

  Sofern möglich, sollten alle shareable Teile der benützten Anwendungen vorgeladen werden (Beispiel: FOR1-RTS). Dadurch werden sie nur einmal geladen; bei jedem weiteren Ladevorgang muss nur noch die (aufwandsarme) Verknüpfung durchgeführt werden. Damit können meist deutlich mehr als 10% der Ein-/Ausgaben und des CPU-Bedarfes eingespart werden.

ESD-Information eliminieren

- Vollkommene Eliminierung

  Diese Maßnahme ist nur für "standalone"-Programme durchführbar. Dies sind LLMs, die von anderen Programmen nicht aufgerufen werden und die vollständig vorgebunden sind. Für sie sollte zur Beschleunigung des Ladevorgangs die ESD- und sonstigen BLS-Informationstabellen durch einen entsprechenden BINDER-Aufruf eliminiert werden. Dazu ist der BINDER wie folgt aufzurufen:

```
/START-BINDER
//START-LLM-UPDATE LIB=<orig-biblio>,ELEMENT=<elem-name>
//SAVE-LLM LIB=<neue lade-biblio>,TEST-SUPPORT=*NO,MAP=*NO,
           SYMBOL-DICTIONARY=*NO,LOGICAL-STRUCTURE=*NO,
           RELOCATION-DATA=*NO
//END
```

Durch diese Parameterwahl werden nur die zum Laden notwendigen BLS-Strukturen eines einfachen Programmes erzeugt. Damit wird das Objekt kleiner. Beim Laden des Programmes werden somit CPU und weitere Ein-/Ausgaben eingespart. Alle für die Bearbeitung notwendigen Informationen sind danach nicht mehr vorhanden.
Das bedeutet:

- Diese Parameter können nicht für Ladeobjekte, die in mehrere Slices aufgeteilt sind, verwendet werden.
- Mit solch einem Objekt können keine LLM-Updates durchgeführt werden.
- Relocatable Objekte (z.B. als Teil eines Runtime-Systems) können nicht erzeugt werden.

Die (zusätzliche) Parameter-Einstellung REQUIRED-COMPRESSION=*YES sollte nicht gewählt werden, da dadurch zwar max. 10% der Ein-/Ausgaben eingespart werden, aber etwa 40% mehr CPU benötigt werden.

- Teilweise Eliminierung

  Beim Mischen von LLMs oder Sub-LLMs in einen Großmodul mit einer einzigen CSECT sollte bei der BINDER-Anweisung MERGE-MODULES angegeben werden, welche ESD-Informationen im Extern-Adressbuch bleiben sollen. Damit kann der BLS-Aufwand beim Laden stark reduziert werden.

6.3 Beschleunigung des Ladevorganges von C-Programmen

Das Laden von C-Programmen wird beschleunigt, wenn die wenigen nicht vorladbaren Module aus dem C-RTS zum übersetzten Programm dazugebunden werden. Dies wird durch folgende zusätzliche BINDER-Anweisung erreicht:

```
//RESOLVE-BY-AUTOLINK LIB=<Partial-Bind-Lib>
```

Die <Partial-Bind-Lib> wird unter dem Namen SYSLNK.CRTE.PARTIAL-BIND installiert. Außerdem ist das Ausblenden von Extern-Namen zur Unterdrückung von DUPLICATE SYMBOLS notwendig:

```
//MODIFY-SYMBOL-VISIBILITY SYMBOL-NAME=*ALL,VISIBLE=*NO
```

Durch beide Anweisungen werden CPU-Zeit und vor allem Laufzeit beim Laden von C-Programmen stark reduziert, wobei das Ladeobjekt nur wenige Seiten (ca. 18) größer als das übersetzte Programm ist.

Hinweis

Damit die CRTE in den Shared-Code geladen wird, müssen die Subsysteme CRTEC und CRTECOM geladen sein.

6.4 Nutzung von DAB

Falls das gebundene Objekt und/oder die benutzten Bibliotheken sehr groß sind und deshalb eine hohe Anzahl von Ein-/Ausgaben beim Laden ausgeführt werden muss, sollte der Einsatz von DAB (Disk Access Buffer) geprüft werden.

Mit DAB können bei häufig referenzierten Dateien die Ein-/Ausgaben so stark reduziert werden, dass im Extremfall die Laufzeit nur noch vom CPU-Bedarf abhängt.

Für das Ladeobjekt ist dabei ein Cache-Bereich zu wählen, der genauso groß wie das Objekt ist. Dies ist besonders einfach, wenn das Objekt das einzige Element der Bibliothek ist. Dann wird mit dem Kommando

```
/START-DAB-CACHING AREA=*FILE(FILE-AREA=<biblio>),CACHE-SIZE=*BY-FILE
```

ein Hauptspeicherbereich von der Größe der Bibliothek reserviert. Dieser wird beim ersten Laden des Objektes gefüllt, sodass bei allen folgenden Ladevorgängen keine zeitintensiven Ein/Ausgaben mehr notwendig sind.

Die Cache-Bereiche für die Bibliotheken können in der Regel kleiner gewählt werden als die Gesamtgröße der Bibliotheken, da meist nicht alle enthaltenen Module nachgeladen werden. Dazu ist beim Kommando /START-DAB-CACHING die Cache-Größe (in KByte oder MByte) explizit anzugeben. Mit SM2 oder dem Kommando /SHOW-DAB-CACHING kann die Cache-Trefferrate jedes Cache-Bereiches überwacht werden. Die Hit-Rate sollte mindestens 80% betragen; ist sie niedriger, so muss der Bereich vergrößert werden.

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 *Ladeeinheit* beim Ablauf erwartet. Er kann eingestellt werden auf:

- 24-Bit-Adressierung (AMODE=24)
- 31-Bit-Adressierung (AMODE=31)
- 32-Bit-Adressierung (AMODE=32) RISC Code
- 24-, 31- oder 32-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*.

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 (LLM)
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 (OM)
Object Module (OM)

Ladbare Einheit, die durch Übersetzen eines Quellprogramms mit Hilfe eines Sprachübersetzers erzeugt wird.

Bindemodulbibliothek (OML)
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)* oder vom Binder TSOSLNK *Großmodule* 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.

Externadressbuch (ESD oder ESV)

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*.

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 *vorgebundenes 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* angesprungen 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*.

Ladeeinheit
Load Unit

Enthält alle *Module*, die mit *einem* Ladeaufruf geladen werden. Jede Ladeeinheit liegt in einem *Kontext*.

Lademodul
Synonym für *Programm*

List-Name-Unit
Wenn mehrere oder alle Objekte einer Bibliothek mit einem einzigen Aufruf des BIND-Makros geladen werden, entsteht eine so genannte List-Name-Unit. Dadurch werden mehrfache DBL-Aufrufe beim Laden einer Symbol-Liste derselben Ladeeinheit vermieden.

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.

LSD
Test- und Diagnoseinformation.

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.

Programm
Ablauffähige Einheit, die vom Binder TSOSLNK aus *Bindemodulen (OMs)* gebunden und in eine katalogisierte Programmdatei oder als Bibliothekselement vom *Elementtyp C* in einer *Programmbibliothek* gespeichert wird.

Programmabschnitt (CSECT)
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 (RLD oder LRLD)

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.

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 (LSD)

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*.

Vorgebundenes Bindemodul

Ladbare Einheit, die vom Binder TSOSLNK aus einzelnen *Bindemodulen (OM)* gebunden wird. Hat das gleiche Format wie ein Bindemodul (OM).

Zugriffsberechtigung eines Kontext

Legt fest, welche Benutzer auf einen *Kontext* zugreifen dürfen. Der Kontext kann privilegiert oder nicht privilegiert sein.

Literatur

Die Handbücher sind online unter *http://manuals.ts.fujitsu.com* zu finden oder in gedruckter Form gegen gesondertes Entgelt unter *http://manualshop.ts.fujitsu.com* zu bestellen.

[1] **BINDER**
Binder in BS2000/OSD
Benutzerhandbuch

[2] **ASSEMBH** (BS2000)
Beschreibung

[3] **LMS** (BS2000/OSD)
SDF-Format
Benutzerhandbuch

[4] **BS2000/OSD-BC**
Systeminstallation
Benutzerhandbuch

[5] **BS2000/OSD-BC**
Kommandos
Benutzerhandbuch

[6] **JV** (BS2000/OSD)
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] **DSSM/SSCM**
Verwaltung von Subsystemen in BS2000/OSD
Benutzerhandbuch

[11] **BS2000/OSD-BC**
Dienstprogramme
Benutzerhandbuch

[12] **BS2000/OSD-BC**
Einführung in das DVS
Benutzerhandbuch

[13] **SDF** (BS2000/OSD)
Einführung in die Dialogschnittstelle SDF
Benutzerhandbuch

[14] **SDF-P** (BS2000/OSD)
Programmieren in der Kommandosprache
Benutzerhandbuch

[15] **SDF-A** (BS2000/OSD)
Benutzerhandbuch

[16] **LMS** (BS2000)
ISP-Format
Benutzerhandbuch

[17] **XHCS** (BS2000/OSD)
8-bit-Code-Verarbeitung im BS2000/OSD
Benutzerhandbuch

[18] **POSIX** (BS2000/OSD)
Grundlagen für Anwender und Systemverwalter
Benutzerhandbuch

Stichwörter