Deutsch

FUJITSU Software BS2000

CRTE V10.1A

Common Runtime Environment

Benutzerhandbuch

Ausgabe Dezember 2019

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
bs2000services@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2015

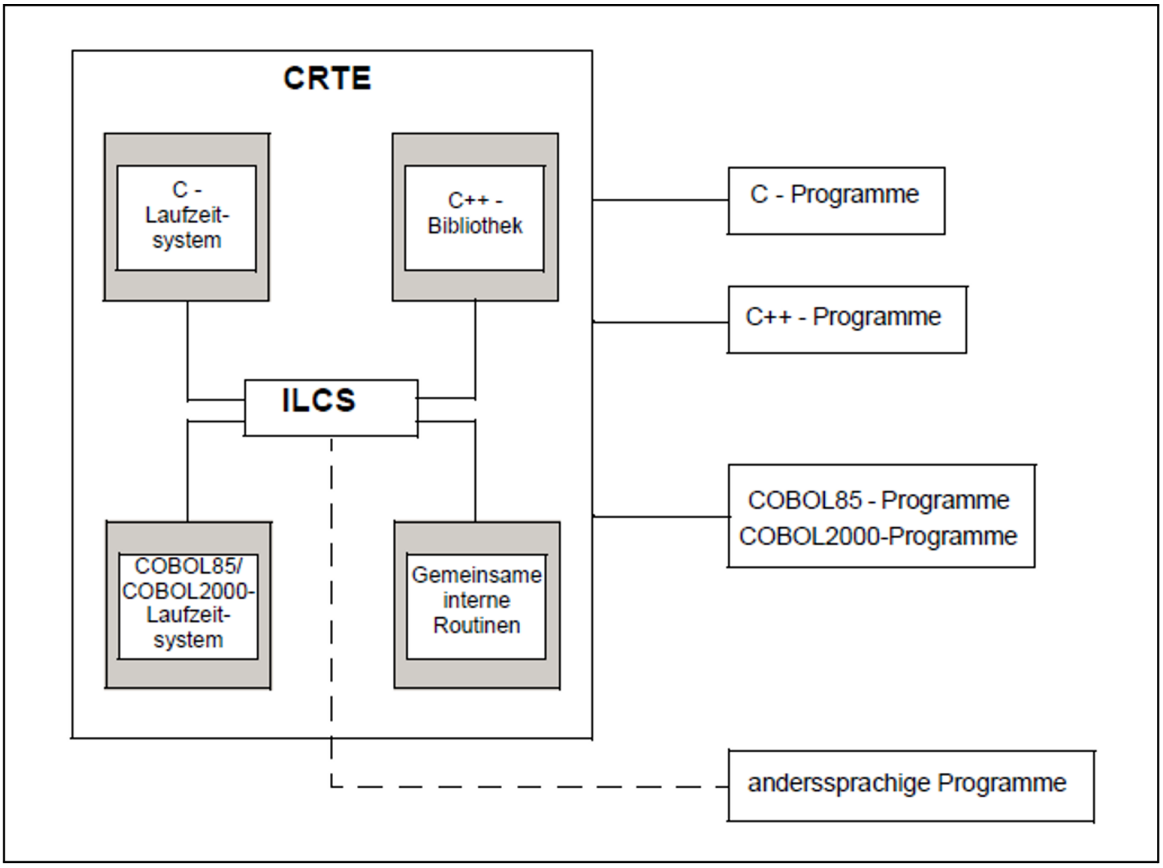
Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:20 erfüllt.

Copyright und Handelsmarken

Inhaltsverzeichnis

1 Einleitung

Das **C**ommon **R**un**T**ime **E**nvironment (**CRTE**) ist die gemeinsame Laufzeitumgebung für C-, C++- und COBOL85- bzw. COBOL2000-Programme im BS2000.
Das CRTE enthält neben den Laufzeitsystemen für C, C++ und COBOL85/COBOL2000 auch alle Routinen der Programm-Kommunikationsschnittstelle ILCS (Inter-Language Communication Services). Aus diesem Grund bietet nur die Verwendung des CRTE die Gewähr, dass Programmsysteme, die aus verschiedensprachigen Programmen bestehen, erfolgreich ablaufen können.



1.1 Zielsetzung und Zielgruppen des Handbuchs

Dieses Handbuch wendet sich an Programmierer und Systemverwalter im BS2000, die die gemeinsame Laufzeitumgebung für COBOL85-, COBOL2000-, C-, und C++-Objekte sowie für Fremdsprachenmix verwenden wollen.

1.2 Konzept des Handbuchs

Dieses Handbuch beschreibt die Komponenten, die Installation und den Einsatz von CRTE.

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

Ergänzende Produkt-Informationen

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

1.3 Änderungen gegenüber dem Vorgänger-Handbuch

Dieses Handbuch enthält gegenüber dem Vorgänger-Handbuch zur Version V10.0B folgende Änderungen:

- neue Bibliotheken zur Unterstützung des Standards ISO/IEC C++ 2017.
- Anpassung der Lieferbestandteile an die aktuelle CRTE-Version.

1.4 Darstellungsmittel

In diesem Handbuch werden folgende Mittel zur Darstellung von funktional wichtigen Textteilen verwendet:

> ¡ für Hinweistexte

> ! **ACHTUNG!** für Warnhinweise

kursive Schrift
kennzeichnet eine Variable, wenn Sie dafür einen Wert einsetzen müssen.

`dicktengleiche Schrift`

für die Darstellung von Eingaben für das System, Systemausgaben und für Dateinamen in Beispielen.

kommando
In der Syntaxbeschreibung für Kommandos werden diejenigen Bestandteile (Bezeichnungen von Kommandos und Parametern) fett dargestellt, die unverändert eingegeben werden müssen.

2 Liefereinheit, Installation und gemeinsame Benutzbarkeit des CRTE

Dieses Kapitel informiert über folgende Themen:

2.1 Liefereinheit CRTE V10.1A

Die Liefereinheit CRTE V10.1A besteht aus folgenden PLAM-Bibliotheken bzw. katalogisierten Dateien:

Bibliotheks- / Dateiname	**Inhalt**
SYSLNK.CRTE.101	Einzelmodule zum kompletten Einbinden des C-Laufzeitsystems Module des COBOL85/COBOL2000-Laufzeitsystems Module und Makros der Programm-Kommunikationsschnittstelle ILCS Module der gemeinsam genutzten internen Laufzeitroutinen
SYSLNK.CRTE.101.PARTIAL-BIND	Verbindungsmodule auf das dynamisch nachladbare C- und COBOL-Laufzeitsystem Nicht vorladbare Module des COBOL85/COBOL2000 und C/C++-Laufzeitsystems Module der Programm-Kommunikationsschnittstelle ILCS Verbindungsmodul auf die gemeinsam genutzten, dynamisch nachladbaren internen Laufzeitroutinen
SYSLNK.CRTE.101.COMPL	Verbindungsmodule auf das dynamisch nachladbare C- und COBOL-Laufzeitsystem
SYSLNK.CRTE.101.CPP	Module und Include-Header der C++-Bibliotheksfunktionen für C++ ab V2.1 (Cfront)
SYSLNK.CRTE.101.CFCPP	Laufzeitsystem für C/C++ ab V3.0 (Cfront)
SYSLNK.CRTE.101.COMPV2	Module für den Ablauf von gebundenen C-V2.0-Programmen
SYSLNK.CRTE.101.COMPV1	Module zum Neubinden von C-V1.0-Objekten
SYSLNK.CRTE.101.POSIX	Bindeschalter für die POSIX-Bibliotheksfunktionen (inklusive Zeitfunktionen)
SYSLNK.CRTE.101.TIME	Bindeschalter für die POSIX-Zeitfunktionen
SYSLNK.CRTE.101.TIMESHIFT	Bindeschalter zur Verlegung des Stichtags (Epoche) für Zeitfunktionen (in der aktuellen Version ohne Auswirkung, wird nur aus Kompatibilitätsgründen unterstützt)
SYSLNK.CRTE.101.TIME38	Zeitfunktionen für spezielle Anwendungen
SYSLNK.CRTE.101.TIME50	Bindeschalter zur Verarbeitung von Zeitstempeln, die mit dem Stichtag (Epoche) 1.1.1950 ermittelt wurden
SYSLNK.CRTE.101.STDCPP	Standard-C++-Bibliothek für C/C++ ab V3.0 (ANSI-C++)
SYSLNK.CRTE.101.SHARE	gemeinsam benutzbares COBOL85/COBOL2000-LZS (Großmodul ITC...) gemeinsam benutzbares C-LZS (Großmodul IC@...)
SYSLNK.CRTE.101.RTSCPP	Laufzeitsystem für C/C++ ab V3.0 (ANSI-C++)

SYSLNK.CRTE.101.CPP-COMPL	Nachladbare Großmodule des C++-Laufzeitsystems, der Standard-C++-Bibliothek und der C++-Tool-Bibliothek
SYSLNK.CRTE.101.TOOLS	C++-Bibliothek Tools.h++ für C/C++ ab V3.0 (ANSI-C++)
SYSLNK.CRTE.101.CXX01	Standard-C++-Bibliothek und Laufzeitsystem für C/C++ ab V4.0 (Standard C++ 2017)
SYSLIB.CRTE.101	Include-Header und Makros der C/C++-Bibliotheksfunktionen ILCS-Makros
SYSLIB.CRTE.101.CPP	Include-Header der C++-Bibliotheksfunktionen (Cfront)
SYSSSC.CRTE.101.C SYSSSC.CRTE.101.C.LOW SYSSSC.CRTE.101.PARTIAL SYSSSC.CRTE.101.PARTIAL.LOW SYSSSC.CRTE.101.COB-PART SYSSSC.CRTE.101.COB-PART.LOW SYSSSC.CRTE.101.COBOL SYSSSC.CRTE.101.COBOL.LOW SYSSSC.CRTE.101.SIS SYSSSC.CRTE.101.SIS.LOW	Subsystem-Deklarationsdateien für C, COBOL85 sowie COBOL2000 und die gemeinsam genutzten Bestandteile (PROSOS)
SYSSII.CRTE.101	IMON-Informationsdatei
SINPRC.CRTE.101	Prozeduren (z.B. ICXPART) für Korrekturen in den C-Verbindungsmodulen, falls CRTE auf einer Nicht-Standardkennung installiert wird
SINLIB.CRTE.101	POSIX- und TIME-Bindeschalter und Include-Header der C-Bibliothek zum Installieren in das POSIX-Dateisystem
SINLIB.CRTE.101.CXX01	Include-Header der C++ 2017-Bibliotheksfunktionen
SKULNK.CRTE.101	Modulbibliothek (X86-Code) ohne COBOL-Laufzeitsystem
SKULNK.CRTE.101.PARTIAL-BIND	Module für Standard Partial-Bind (X86-Code) ohne COBOL-Laufzeitsystem
SKULNK.CRTE.101.COMPL	Module für Complete Partial-Bind (X86-Code) ohne COBOL-Laufzeitsystem
SKULNK.CRTE.101.POSIX	Bindeschalter für POSIX (inkl. Zeitfunktionen)
SKULNK.CRTE.101.TIME	Bindeschalter für POSIX-Zeitfunktionen
SKULNK.CRTE.101.TIMESHIFT	Bindeschalter zur Verlegung des Stichtags (Epoche) für Zeitfunktionen (in der aktuellen Version ohne Auswirkung, wird nur aus Kompatibilitätsgründen unterstützt)
SKULNK.CRTE.101.TIME38	Zeitfunktionen für spezielle Anwendungen

SKULNK.CRTE.101.TIME50	Bindeschalter zur Verarbeitung von Zeitstempeln, die mit dem Stichtag (Epoche) 1.1.1950 ermittelt wurden
SKULNK.CRTE.101.RTSCPP	X86-Variante des C++-Laufzeitsystems
SKULNK.CRTE.101.CPP-COMPL	Nachladbare Großmodule des C++-Laufzeitsystems, der Standard-C++-Bibliothek und der C++-Tool-Bibliothek
SKULNK.CRTE.101.STDCPP	X86-Variante der Standard-C++-Bibliothek
SKULNK.CRTE.101.TOOLS	X86-Variante der Bibliothek Tools.h++
SKULNK.CRTE.101.CXX01	X86-Variante der Standard-C++-Bibliothek und der C++ 2017 Laufzeitbibliothek
SKUSSC.CRTE.101.PARTIAL SKUSSC.CRTE.101.SIS	Subsystem-Deklarationsdateien für X86-Variante
SYSDOC.CRTE.101.CXX01.OSS	Readme und Lizenzen der im C++ 2017 Laufzeitsystem enthaltenen Open Source Software
SYSFGM.CRTE.101.D SYSFGM.CRTE.101.E	Freigabemitteilung deutsch bzw. englisch

C-V2.0- und ILCS-Module (SYSLNK.CRTE-BASYS.101.CLIB und SYSLNK.CRTE-BASYS.101.ILCS) werden mit CRTE-BASYS ausgeliefert.

Die *.PTH-Dateien werden ebenfalls mit CRTE-BASYS ausgeliefert.

Die Meldungsdateien des CRTE sind Bestandteil des Grundausbaus von BS2000.

2.2 CRTE installieren

Dieser Abschnitt informiert über folgende Themen:

- CRTE-Bibliotheken, die ohne Versionsangaben zu installieren sind
- Standardinstallation auf die Kennung „$.“
- Installation mit IMON auf eine Nicht-Standardkennung
- Installation der Header-Dateien und POSIX-Bindeschalter im Default-POSIX-Dateiverzeichnis
- Installation der Header-Dateien und POSIX-Bindeschalter in beliebigem POSIX-Dateiverzeichnis
- Private Installation
- Wechsel von einer CRTE-Vorgängerversion auf CRTE V10.1A

2.2.1 CRTE-Bibliotheken, die ohne Versionsangaben zu installieren sind

Folgende CRTE-Bibliotheken müssen ohne die Versionsangabe im Namen („101“) installiert sein:

SKULNK.CRTE.101
SKULNK.CRTE.101.CXX01
SKULNK.CRTE.101.POSIX
SKULNK.CRTE.101.RTSCPP
SKULNK.CRTE.101.STDCPP
SKULNK.CRTE.101.TOOLS
SKULNK.CRTE.101.TIME
SKULNK.CRTE.101.TIMESHIFT
SKULNK.CRTE.101.TIME38
SKULNK.CRTE.101.TIME50
SKULNK.CRTE.101.PARTIAL-BIND
SKULNK.CRTE.101.COMPL
SKULNK.CRTE.101.CPP-COMPL
SYSDOC.CRTE.101.CXX01.OSS
SYSLIB.CRTE.101
SYSLIB.CRTE.101.CPP
SYSLIB.CRTE.101.CXX01
SYSLNK.CRTE.101
SYSLNK.CRTE.101.CXX01
SYSLNK.CRTE.101.COMPV1
SYSLNK.CRTE.101.COMPV2
SYSLNK.CRTE.101.CPP
SYSLNK.CRTE.101.CFCPP
SYSLNK.CRTE.101.PARTIAL-BIND
SYSLNK.CRTE.101.COMPL
SYSLNK.CRTE.101.CPP-COMPL
SYSLNK.CRTE.101.POSIX
SYSLNK.CRTE.101.RTSCPP
SYSLNK.CRTE.101.STDCPP
SYSLNK.CRTE.101.TIME
SYSLNK.CRTE.101.TIMESHIFT
SYSLNK.CRTE.101.TIME38
SYSLNK.CRTE.101.TIME50
SYSLNK.CRTE.101.TOOLS

Bei der Installation mit dem Produkt IMON (**I**nstallations **Mon**itor) werden die genannten Bibliotheken automatisch in Bibliotheken **ohne** Versionsangabe kopiert.

Wenn Sie nicht mit IMON installieren, müssen Sie die oben aufgelisteten Bibliotheken und die Bibliotheken des Produkts POSIX-HEADER selbst in Dateien ohne Versionsangabe umbenennen. Hierfür können Sie die Prozedur ICXINST aus der Bibliothek SINPRC.CRTE.101 verwenden.

2.2.2 Standardinstallation auf die Kennung „$.“

Die Standardinstallation des CRTE und der POSIX-HEADER erfolgt in der Regel über das Produkt IMON (**I** nstallations **Mon**itor).

Es empfiehlt sich, das CRTE im Standard-Installationsmodus zu installieren, d.h. auf die Standardkennung des Systems ($.). Bei der Installation mit IMON werden die im Abschnitt "CRTE-Bibliotheken, die ohne Versionsangaben zu installieren sind" genannten Bibliotheken automatisch in Bibliotheken ohne Versionsangabe umbenannt. Wenn Sie ohne IMON installieren, müssen Sie die Bibliotheken „von Hand“ bzw. mit der Prozedur ICXINST umbenennen (siehe Abschnitt "CRTE-Bibliotheken, die ohne Versionsangaben zu installieren sind" unten).

2.2.3 Installation mit IMON auf eine Nicht-Standardkennung

Wenn Sie CRTE mit IMON auf eine Nicht-Standardkennung installieren, führt IMON automatisch alle notwendigen Schritte aus.

2.2.4 Installation der Header-Dateien und POSIX-Bindeschalter im Default-POSIX-Dateiverzeichnis

Die mit CRTE angebotenen POSIX-Bibliotheksfunktionen können Sie nutzen, sofern auch ein POSIX-Subsystem verfügbar ist.

Die zur Nutzung der CRTE-POSIX-Funktionalität benötigte Software-Konfiguration richten Sie wie folgt ein:

1. Installieren Sie CRTE in POSIX. Hierfür verwenden Sie den Aufruf:

   ```
   /START-POSIX-INSTALLATION
   ```

2. Installieren Sie die POSIX-Header in POSIX.
3. Kopieren Sie die Bibliothek SYSLIB.POSIX-HEADER.101 (Release Unit von BS2000 OSD/BC V10.0) in eine Bibliothek namens SYSLIB.POSIX-HEADER (Name ohne Versionsangabe). Benutzen Sie dazu die Prozedur ICXINST aus der Bibliothek SINPRC.POSIX-HEADER.101. Bei der Installation mit IMON wird dies automatisch durchgeführt.

2.2.5 Installation der Header-Dateien und POSIX-Bindeschalter in beliebigem POSIX-Dateiverzeichnis

Wenn Sie unter POSIX arbeiten, können Sie die Header-Dateien auch in einem frei wählbaren Dateiverzeichnis unter POSIX „privat“ installieren. Damit haben Sie die Möglichkeit, CRTE V10.1A zusammen mit einem entsprechend privat installierten C/C++-Compiler ab V3.0 parallel zu einer älteren CRTE-Version im POSIX zu betreiben.

Hierfür installieren Sie die Header von CRTE und die POSIX-HEADER mit den Prozeduren ICXINSTALL gesondert im `ufs` unter POSIX:

```
/CALL-PROC *LIB(SINPRC.CRTE.101,ICXINSTALL), PROC-PAR=(directory='directory')
```

```
/CALL-PROC *LIB(SINPRC.POSIX-HEADER.101,ICXINSTALL), -
/             PROC-PAR=(directory='directory')
```

Die Header werden unter *directory*`/usr/include` abgelegt, die Bindeschalter unter *directory*`/opt/CRTE/lib`.

Für die Deinstallation stehen die Prozeduren ICXDELETE zur Verfügung:

```
/CALL-PROC *LIB(SINPRC.CRTE.101,ICXDELETE), PROC-PAR=(directory='directory')
```

```
/CALL-PROC *LIB(SINPRC.POSIX-HEADER.101,ICXDELETE), -
/             PROC-PAR=(directory='directory')
```

2.2.6 Private Installation

CRTE bietet Ihnen die Möglichkeit der Privat-Installation ohne Einsatz von IMON. Auf diese Weise können Sie die neue CRTE-Version auf einer „privaten“ Kennung parallel zu einer im System bereits vorhandenen „offiziellen“ Version betreiben.

Hierbei müssen Sie die Dateien mit der Prozedur ICXINST übertragen und umbenennen und anschließend die Prozedur ICXPPART aufrufen:

`/CALL-PROC *LIB-ELEM(SINPRC.CRTE.101,ICXPPART),PROC-PAR='`*privateUserid*`'`

Die Prozedur ICXPPART ändert das C-Verbindungsmodul, das COBOL-Verbindungsmodul und das Verbindungsmodul für die gemeinsam genutzten Laufzeitroutinen in den Bibliotheken SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CRTE.COMPL und SYSLNK.CRTE dahingehend, dass das dynamisch nachladbare C-Laufzeitsystem, das COBOL-Laufzeitsystem und die gemeinsam benutzten Bestandteile aus der Bibliothek $*privateUserid*.SYSLNK.CRTE nachgeladen werden, unabhängig von vorgeladenen Subsystemen und IMON-Zugriffen.

Außerdem müssen Sie die Voreinstellung der USER-INCLUDE-LIBRARY-Option und der STD-INCLUDE-LIBRARY-Option des C- und C++-Compilers entsprechend umstellen. Der C/C++-Compiler muss dann auch ohne IMON installiert sein.

Nachfolgend sind die für die Privatinstallation des CRTE erforderlichen Schritte zusammengefasst::

1. Spielen Sie die Dateien unter der gewünschten Kennung ein.
2. Starten Sie die Prozedur ICXINST für CRTE und POSIX-HEADER.

   Die Prozedur entfernt jeweils die Versionsangaben aus den Bibliotheksnamen:

   `/CALL-PROC *LIB(SINPRC.CRTE.101,ICXINST)`

   `/CALL-PROC *LIB(SINPRC.POSIX-HEADER.101,ICXINST)`
3. Rufen Sie die Prozedur ICXPPART auf:

   `/CALL-PROC *LIB(SINPRC.CRTE.101,ICXPPART), PROC-PAR='`*privateUserid*`'`

   Diese Prozedur ersetzt in den Adaptermodulen für die Bindetechniken
   Standard Partial-Bind und Complete Partial-Bind den `$.SYSLNK.CRTE` durch `$`*privateU-serid*`.SYSLNK.CRTE`.

   Die genannten Änderungen sind notwendig, wenn beim Binden die PARTIAL-BIND-Bibliothek oder die COMPL-Bibliothek verwendet wird.
4. Wenn Sie Ihre Programme unter POSIX übersetzen, müssen Sie zusätzlich die Prozedur ICXINSTALL aufrufen, um auch die Header-Dateien und Bindeschalter unter einem privaten Pfad zu installieren (siehe nachfolgender Abschnitt "Installation der Header-Dateien und POSIX-Bindeschalter in beliebigem POSIX-Dateiverzeichnis“).

2.2.7 Wechsel von einer CRTE-Vorgängerversion auf CRTE V10.1A

Beim Umstieg von einer Vorgängerversion auf CRTE V10.1A verfahren Sie wie folgt:

1. Vor der Installation der mit CRTE V10.1A und mit POSIX-HEADER V10.1A ausgelieferten Header-Dateien im POSIX (siehe "Installation der Header-Dateien und POSIX-Bindeschalter im Default-POSIX-Dateiverzeichnis“) müssen Sie zunächst die Header-Dateien der CRTE- und POSIX-HEADER-Vorgängerversion entfernen. Hierzu benötigen Sie die Deinstallationsprozeduren aus den alten Produktdateien von CRTE und POSIX-HEADER:
   - Bei einer Standardinstallation verwenden Sie zum Löschen das POSIX-Installationstool (/START-POSIX-INSTALLATION) und wählen die Funktion „Delete packages from POSIX“.
   - Bei einer Privatinstallation verwenden Sie zum Löschen die mit CRTE und POSIX-HEADER ausgelieferten Prozeduren der Vorgängerversionen:

     `/CALL-PROC *LIB(SINPRC.CRTE.`*`oldversion`*`, ICXDELETE)`
     `/CALL-PROC *LIB(SINPRC.POSIX-HEADER.`*`oldversion`*`, ICXDELETE)`
2. Alle Dateien der Vorgängerversionen dürfen Sie nun löschen.
3. Wenn die nachfolgend aufgelisteten Subsysteme vorgeladen sind, müssen Sie diese vor der Installation von CRTE V10.1A stoppen und aus dem Subsystemkatalog entfernen bzw. durch die entsprechenden Subsysteme der Version V10.1A ersetzen:

   CRTEC, CRTECOB, CRTESIS, CRTESIK, CRTEPART, COBPART, CRTECOM, CRTEPARK

   Bei einer Standard-Installation von CRTE V10.1A mit IMON werden die neuen Subsystemeinträge automatisch generiert.

> ILCS (CRTECOM) wird seit CRTE V2.1 nicht mehr als vorladbares Subsystem ausgeliefert. Falls Sie bisher mit einer CRTE-Version kleiner V2.1 gearbeitet haben, müssen Sie den Eintrag für CRTECOM im Subsystemkatalog löschen, da es dafür keine Entsprechung in CRTE V10.1A gibt.

2.3 Gemeinsam benutzbares CRTE

Bis auf wenige Routinen ist das gesamte CRTE gemeinsam benutzbar.

Das gemeinsam benutzbare CRTE ist in folgende Subsysteme aufgeteilt:

- CRTECOB für COBOL85 bzw. COBOL2000,
- CRTEC für C/C++,
- CRTEPART für Bindetechnik Partial-Bind auf /390 (C/C++)
- COBPART für Bindetechnik Partial-Bind auf /390 (COBOL)
- CRTESIS für die gemeinsam benutzbaren Bestandteile

Der Systemverwalter kann diese Subsysteme in den Klasse-4-Speicher laden.

> **i** Wenn die Subsysteme in den oberen Adressraum des Klasse-4-Speichers geladen sind, ist beim Aufruf von Programmen, die das entsprechende Laufzeitsystem dynamisch binden, im START-PROGRAM-Kommando die Angabe PROG-MO-DE=ANY erforderlich.

Subsystemkatalog generieren

Als Eingabedateien für die Generierung des Subsystemkatalogs werden folgende Subsystem-Deklarationsdateien bereitgestellt:

SYSSSC.CRTE.101.C

CRTEC für C/C++ im Klasse-4-Speicher (oberer Adressraum)

SYSSSC.CRTE.101.C.LOW

CRTEC für C/C++ im Klasse-4-Speicher (unterer Adressraum)

SYSSSC.CRTE.101.PARTIAL

CRTEPART für C/C++ PARTIAL im Klasse-4-Speicher (oberer Adressraum)

SYSSSC.CRTE.101.PARTIAL.LOW

CRTEPART für C/C++ PARTIAL im Klasse-4-Speicher (unterer Adressraum)

SYSSSC.CRTE.101.COB-PART

COBPART für COBOL PARTIAL im Klasse-4-Speicher (oberer Adressraum)

SYSSSC.CRTE.101.COB-PART.LOW

COBPART für COBOL PARTIAL im Klasse-4-Speicher (unterer Adressraum)

SYSSSC.CRTE.101.COBOL

CRTECOB für COBOL im Klasse-4-Speicher (oberer Adressraum)

SYSSSC.CRTE.101.COBOL.LOW

CRTECOB für COBOL im Klasse-4-Speicher (unterer Adressraum)

SYSSSC.CRTE.101.SIS

CRTESIS für COBOL im Klasse-4-Speicher (oberer Adressraum)

SYSSSC.CRTE.101.SIS.LOW

CRTESIS für COBOL im Klasse-4-Speicher (unterer Adressraum)

Für die Generierung des Subsystemkatalogs steht im BS2000 das Programm SSCM zur Verfügung (siehe Handbuch „Verwaltung von Subsystemen (DSSM/SSCM)“).

Innerhalb des Klasse-4-Speichers können die Subsysteme entweder in den oberen oder unteren Adressraum (d.h. oberhalb oder unterhalb von 16 MB) geladen werden.

Alle Subsysteme werden standardmäßig in die oberen 16 MB des Klasse-4-Speichers geladen.
Die Dateien mit der Endung LOW ermöglichen das Vorladen unterhalb von 16 MB.

Subsysteme aktivieren und deaktivieren

Die Subsystemnamen und -versionen zum Aktivieren bzw. Deaktivieren der Subsysteme sind folgender Tabelle zu entnehmen:

Subsystem	Name	Version
C	CRTEC	10.1
C-PARTIAL[1)] (/390)	CRTEPART	10.1
C-PARTIAL[1)] (X86)	CRTEPARK	10.1
COBOL85/COBOL2000 PARTIAL[1)] (/390)	COBPART	10.1
COBOL85/COBOL2000	CRTECOB	10.1
gemeinsam benutzbare interne Bestandteile	CRTESIS	10.1
gemeinsam benutzbare interne Bestandteile (X86)	CRTESIK	10.1

[1)]Die Subsysteme ... PARTIAL gelten gleichermaßen für Standard und Complete Partial-Bind.

2.4 Benötigte Laufzeitprodukte bei Programmverknüpfung

Da die CRTE-Versionen aufwärts kompatibel sind, muss bei Programmverknüpfungen mindestens die jeweils höchste der benötigten CRTE-Versionen verwendet werden, wenn die einzelnen Programme unterschiedliche CRTE-Versionen voraussetzen.

Einzelheiten zur benötigten CRTE-Version entnehmen Sie bitte den Freigabemitteilungen der jeweiligen Compiler.

Bei Verknüpfung mit Programmen anderer Sprachen (ASSEMBH ab V1.2, FOR1 V2.2, Pascal-XT ab V2.2, PLI1 ab V4.2, RPG3 ab V4.0) muss zusätzlich das Laufzeitsystem dieser Sprachen eingebunden werden.

3 Sprachspezifische Komponenten des CRTE

Neben den gemeinsam benutzbaren Komponenten des CRTE gibt es die folgenden sprachspezifischen Komponenten:

- COBOL85- bzw. COBOL2000-Laufzeitsystem
- C-Laufzeitsystem
- C++-Laufzeitsystem
- C++-Laufzeitsystem (Cfront)

3.1 CRTE-Bibliotheken für die Bindetechnik Partial-Bind

Die Bindetechnik Partial-Bind gibt es in zwei Varianten:

- Standard Partial-Bind
- Complete Partial-Bind

Je nachdem, welche Partial-Bind-Variante Sie auf /390-Systemen einsetzen, benötigen Sie unterschiedliche Bibliotheken:

- Partial-Bind auf /390-Systemen:
  - SYSLNK.CRTE.PARTIAL-BIND für Standard Partial-Bind
  - SYSLNK.CRTE.COMPL für Complete Partial-Bind

> **i** Die CRTE-Bibliotheken für die Bindetechnik Partial-Bind dürfen **nur zum Binden**, **nicht** aber als BLSLIB **für das Laden** von Programmen mit offenen Externverweisen beim dynamischen Binden mit DBL (siehe "Dynamisches Binden mit dem DBL") verwendet werden, insbesondere dann nicht, wenn das Subsystem CRTEC vorgeladen ist. Als BLSLIB ist in diesem Fall ausschließlich die Bibliothek SYSLNK.CRTE zu verwenden.
>
> Bei COBOL- und gemischten C-/COBOL-Programmen darf die Bibliothek für Complete Partial-Bind nur verwendet werden, wenn die Objekte unter POSIX ablaufen und Shared Objects geladen werden.

Nähere Informationen zur Bindetechnik Partial-Bind im Allgemeinen sowie zu den Besonderheiten der beiden Partial-Bind-Varianten finden Sie im Abschnitt "Dynamisches Nachladen des C-/COBOL-Laufzeitsystems und der internen Routinen (Bindetechnik Partial-Bind)".

3.1.1 Bibliotheken für die Bindetechnik Partial-Bind auf /390-Systemen

Die Bibliotheken SYSLNK.CRTE.PARTIAL-BIND und SYSLNK.CRTE.COMPL ermöglichen es, C-Programme, C-/COBOL-Mischprogramme oder COBOL-Programme so zu binden, dass die folgenden Bedingungen erfüllt sind:

- Die C-Programme, C-/COBOL-Mischprogramme oder COBOL-Programme enthalten keine offenen Externverweise mehr.
- Das C-/COBOL-Laufzeitsystem bzw. die gemeinsamen internen Routinen werden erst zum Ablaufzeitpunkt dynamisch nachgeladen.

Dazu enthalten die Bibliotheken SYSLNK.CRTE.PARTIAL-BIND und SYSLNK.CR-TE.COMPL Verbindungsmodule, die an Stelle der C-Laufzeitmodule, der COBOL-Laufzeitmodule bzw. der gemeinsamen internen Laufzeitroutinen eingebunden werden und die alle offenen Externbezüge eines Programms auf die vorladbaren Teile des C-/COBOL-Laufzeitsystems bzw. der gemeinsamen internen Laufzeitroutinen befriedigen.

Die Bibliotheken enthalten Module, die für das Binden eines C- bzw. COBOL-Programms ohne offene Externbezüge auf das CRTE benötigt werden. Dies sind u.a.

- nicht vorladbare Module des C-Laufzeitsystems, Namens-Adaptermodule, ILCS-Module sowie
- nicht vorladbare COBOL85/COBOL2000-Laufzeitmodule.

Das eigentliche C-/COBOL-Laufzeitsystem bzw. die eigentlichen internen Laufzeitroutinen stehen als dynamisch nachladbare Großmodule in der Bibliothek SYSLNK.CRTE und als Bestandteil der Subsysteme CRTEPART /COBPART bzw. CRTESIS zur Verfügung.

Wenn Sie CRTE privat installiert haben und die Verbindungsmodule mit der Prozedur ICX-PPART geändert haben, werden die benötigten Module unabhängig von vorgeladenen Subsystemen und IMON-Zugriffen aus der Bibliothek von der privaten Kennung nachgeladen. Das gilt auch dann, wenn die mit diesen Verbindungsmodulen gebundenen Programme an einer anderen Anlage eingesetzt werden.

Wenn Sie die offizielle CRTE-Installation verwenden, wird zum Ablaufzeitpunkt zunächst versucht, die Großmodule aus dem Klasse 4-Speicher nachzuladen. Ist das Subsystem CRTEPART/COBPART bzw. CRTESIS nicht gestartet, so wird über die `get-path-name`-Funktionalität von IMON ermittelt, wo die Bibliothek SYSLNK.CRTE abgelegt wurde und anschließend aus dieser Bibliothek dynamisch nachgeladen. Ist weder das Subsystem noch IMON verfügbar, dann wird die Bibliothek unter der Standardkennung $ gesucht bzw. unter der beim Aufruf von ICXPART angegebenen Kennung.

Da weder das gesamte C-/COBOL-Laufzeitsystem noch die gemeinsamen internen Laufzeitroutinen eingebunden werden, sondern lediglich die Verbindungsmodule (und noch einige Module, die nicht zum vorladbaren LZS gehören), benötigt das fertig gebundene Programm bzw. Modul deutlich weniger Plattenspeicherplatz als beim statischen Einbinden der C- / COBOL- Laufzeitmodule und der gemeinsamen internen Laufzeitroutinen aus der Bibliothek SYSLNK.CRTE. Außerdem wird die Bindezeit und, sofern das bzw. die benötigten Subsysteme vorgeladen sind, die Ladezeit beschleunigt.

3.2 COBOL85- bzw. COBOL2000-Laufzeitsystem

Das COBOL85- bzw. COBOL2000-Laufzeitsystem besteht aus Laufzeitmodulen, die beim Binden eines COBOL85 /COBOL2000-Objekts zum ablauffähigen Programm verwendet werden.

Die COBOL-Laufzeitmodule sind an dem Präfix „ITC“ zu erkennen. Sie stehen, soweit sie nicht mit dem Subsystem vorgeladen werden, als Elemente in den folgenden Bibliotheken zur Verfügung:

- Bibliothek für das statische Binden:
  - SYSLNK.CRTE (/390-Systeme)

    Die COBOL-Laufzeitmodule liegen im Bindemodul-Format (Typ R) vor.

  Beim statischen Binden mit CRTE werden alle unbefriedigten externen Referenzen befriedigt.
- Bibliothek für das Binden mit der Partial-Bind-Bindetechnik
  - SYSLNK.CRTE.PARTIAL-BIND und SYSLNK.CRTE.COMPL (/390-Systeme)

  Diese Bibliotheken unterstützen die Partial-Bind-Bindetechnik, die eine signifikante Reduzierung der Größe von fertig gebundenen COBOL-Programmen ermöglicht. Hierbei werden alle unbefriedigten Referenzen durch Verbindungsmodule befriedigt. Das CO-BOL85/COBOL2000-Laufzeitsystem wird erst zum Ablaufzeitpunkt dynamisch nachgeladen (siehe Abschnitt "Bibliotheken für die Bindetechnik Partial-Bind auf /390-Systemen“).

  Die mit der Partial-Bind-Bindetechnik gebundenen Programme verwenden das Subsystem COBPART (Binden mit SYSLNK.CRTE.PARTIAL-BIND oder SYSLNK.CRTE.COMPL).

  > **i** Die Bindetechnik Complete Partial-Bind (Bibliothek SYSLNK.CRTE.COMPL) darf bei COBOL- und gemischten C-/COBOL-Programmen nur eingesetzt werden, wenn die Objekte unter POSIX ablaufen und Shared Objects geladen werden.

Nähere Informationen sowie Beispiele zu den verschiedenen Bindetechniken finden Sie im Kapitel "Binden mit CRTE“.

Die meisten gemeinsam benutzbar gestalteten Laufzeitmodule sind in einem Großmodul namens ITCSR85 in der Bibliothek SYSLNK.CRTE.SHARE vorgebunden. Dieses Großmodul kann mit dem Subsystem CRTECOB vom Systemverwalter in den Klasse-4-Speicher geladen werden (siehe "Gemeinsam benutzbares CRTE").

Die Laufzeitmodule stellen dem COBOL85- bzw. COBOL2000-Compiler bekannte Unterprogramme dar, die im Wesentlichen in zwei Gruppen unterteilt werden können:

1. Unterprogramme für komplexe COBOL-Anweisungen
2. Unterprogramme zum Anschluss des generierten Moduls an Betriebssystemfunktionen

   Diese Unterprogramme dienen hauptsächlich dazu, die Codegenerierung des Compilers völlig unabhängig vom Betriebssystem zu halten.

   Wesentliche Funktionen unter diesem Titel sind:

   - Anschluss der COBOL-Programme an das Ein-/Ausgabesystem
   - Anschluss der COBOL-Programme an SORT
   - Anschluss der COBOL-Programme an UDS
   - Anschluss der COBOL-Programme an Ablaufteil-Funktionen

Eine Liste mit den Namen und Funktionen aller COBOL85- bzw. COBOL2000-Laufzeitmodule befindet sich in den COBOL-Benutzerhandbüchern.

Unterstützung großer Dateien (> 2 GB) in COBOL-Programmen

COBOL-Programme, die mit COBOL85 V2.3A oder COBOL2000 kompiliert und mit CRTE V10.1A gebunden wurden, können große BS2000-Dateien (> 2 GB) verarbeiten.

Die maximale Größe einer BS2000-Datei beträgt 4096 Gigabyte (= 4 Terabyte), eine POSIX-Datei kann maximal 1024 Gigabyte (= 1 Terabyte) groß sein.

3.3 C-Laufzeitsystem

Das C-Laufzeitsystem wird für das Übersetzen und Binden von C- und C++-Programmen benötigt. C++-Programme, die auch die C++-Bibliotheksfunktionen für komplexe Mathematik und/oder Standard-Ein-/Ausgabe nutzen wollen, benötigen zusätzliche Include-Elemente und Module.

Im Wesentlichen besteht das C-Laufzeitsystem aus folgenden Komponenten:

- Standard-Include-Elemente für die C-Bibliotheksfunktionen
- Quellprogramme für die Generierung der benutzerspezifischen Lokalitäten
- C-Laufzeitmodule

  Die Laufzeitmodule enthalten u.a. den Code für sämtliche C-Bibliotheksfunktionen, zentrale Routinen zur Realisierung auch der C++-Bibliotheksfunktionen und weitere Routinen zur Realisierung der Betriebssystemschnittstellen von C- und C++-Programmen..

  Die Modul- und Entry-Namen des C-Laufzeitsystems beginnen mit „IC@“, „ICS“, „ICX“ oder „ICP“.

  Die C-Laufzeitmodule werden in verschiedenen Ausführungen bereitgestellt, so dass sie dynamisch oder statisch eingebunden, dynamisch nachgeladen und gemeinsam benutzbar in den Klasse-4-Speicher vorgeladen werden können. Es sind nicht alle Module des C-Laufzeitsystems gemeinsam benutzbar gestaltet und vorladbar. Dazu zählen z.B. die Namens-Adaptermodule (siehe "Bibliothek SYSLNK.CRTE" im Abschnitt "Bibliotheken des C-Laufzeitsystems").

Diese und weitere Komponenten des C-Laufzeitsystems sind als Bibliothekselemente in verschiedenen CRTE-Bibliotheken gespeichert. Im Folgenden werden die CRTE-Bibliotheken mit ihren jeweiligen Inhalten aufgeführt.

3.3.1 Bibliotheken des C-Laufzeitsystems

In diesem Abschnitt werden folgende Bibliotheken beschrieben:

Bibliothek SYSLNK.CRTE

Diese Bibliothek enthält:

- Quellprogramme für die Generierung der benutzerspezifischen Lokalitäten (Typ S)

USLOCA	Assembler-Quellprogramm
USLOCC	C-Quellprogramm

- Einzelmodule des C-Laufzeitsystems
  (Bindemodule, Typ R CRTESIS; subsystem:CRTESIS)

  Diese Module können statisch mit dem BINDER oder dynamisch mit dem DBL gebunden werden. Das C-Laufzeitsystem kann auch dynamisch nachgeladen werden (Partial-Bind-Bindetechnik). In diesem Fall muss beim Binden statt der Bibliothek SYS-LNK.CRTE eine der beiden folgenden Bibliotheken angegeben werden:
  - SYSLNK.CRTE.PARTIAL-BIND (Standard Partial-Bind)
  - SYSLNK.CRTE.COMPL (Complete Partial-Bind)
- Namens-Adaptermodule für neue C-Bibliotheksfunktionen
  - Bindemodule (OMs, Typ R)
  - Bindelademodule (LLMs, Typ L)

  Die Adaptermodule gehören zu den nicht vorladbaren Bestandteilen des C-Laufzeitsystems und müssen deshalb in das Anwendungsprogramm eingebunden werden.
- C-Laufzeitsystem als dynamisch nachladbares Großmodul (Bindemodul, Typ R) und Bindelademodul (LLM, Typ L)

  Dieses Großmodul wird zum Ablaufzeitpunkt dynamisch nachgeladen, wenn beim Binden an Stelle der Bibliothek SYSLNK.CRTE die Bibliothek SYSLNK.CRTE.PARTI-AL-BIND bzw. SYSLNK.CRTE.COMPL eingebunden wird und das nachladbare C-Laufzeitsystem als Subsystem CRTEPART nicht in den Klasse-4-Speicher vorgeladen ist.

Bibliothek SYSLNK.CRTE.SHARE

Diese Bibliothek enthält das C-Laufzeitsystem als gemeinsam benutzbares Großmodul IC@RTSXS. Dieses Großmodul kann vom Systemverwalter als Subsystem CRTEC im Klasse-4-Speicher geladen werden.

Bibliothek SYSLNK.CRTE.POSIX

Diese „Bindeschalter“-Bibliothek muss immer eingebunden werden, wenn die POSIX-Funktionen des C-Laufzeitsystems verwendet werden. Sie enthält die Module ICSS44Y@ und ICSTISP@ (Bindemodul, Typ R), das **vorrangig** vor den Modulen in der Bibliothek SYSLNK.CRTE bzw. SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK. CRTE.COMPL eingebunden werden muss. Es ist erforderlich, beim Binden mit BINDER die Bibliothek mit einer INCLUDE-MODULES-Anweisung (ohne Angabe eines Modulnamens) einzubinden, da sonst beim Einbinden über den Autolink-Mechanismus (RESOLVEs) unbedingt die Bindereihenfolge zu beachten ist und sich der Inhalt der Bibliotheken auch ändern kann. Beim dynamischen Binden mit dem DBL muss der Bindeschalter-Bibliothek ein niedrigerer Linkname BLSLIB*nn* zugewiesen werden als den nachrangig einzubindenden CRTE-Bibliotheken. Bei Programmentwicklung in der POSIX-Shell mit den Kommandos cc, c89 oder CC wird der POSIX-Bindeschalter automatisch eingebunden.

Bibliotheken SYSLNK.CRTE.COMPV1 und SYSLNK.CRTE.COMPV2

Vom C-Compiler V1.0 erzeugte Objekte sind mit CRTE ablauffähig, wenn sie sowohl mit der Bibliothek SYSLNK. CRTE bzw. SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CR-TE.COMPL als auch mit der Bibliothek SYSLNK. CRTE.COMPV1 neu gebunden werden.

C-Module bzw. C-Programme mit eingebundenem Verbindungsmodul aus der CLIB V2.0 benötigen für den Ablauf das zu C V2.0 kompatible, dynamisch nachladbare C-Laufzeitsystem. Dieses Laufzeitsystem ist nach der Installation von CRTE in der Bibliothek $.CLIB enthalten. Dadurch entfällt die Notwendigkeit (vgl. CRTE bis einschließlich V1.0B), vor Aufruf eines C-V2.0-Programmes die Bibliothek SYSLNK.CRTE.COMPV2 mit dem Linknamen IC@CLIB zuzuweisen. Prozeduren, die diese Zuweisung enthalten, müssen nicht geändert werden.

Bibliothek SYSLNK.CRTE.TIMESHIFT

Diese „Bindeschalter“-Bibliothek wird nur noch aus Kompatibiltätsgründen unterstützt.

Sie diente in früheren Versionen dazu, den Stichtag (Epoche) für die Zeitfunktionen `ctime`, `difftime`, `ftime`, `gmtime`, `localtime`, `mktime` und `time` vom 1.1.1950 auf den 1.1.1970 00:00:00 zu verlegen. Da in der aktuellen CRTE-Version der Stichtag standardmäßig der 1.1.1970 00:00:00 ist, ist der Einsatz dieses Bindeschalters nicht mehr erforderlich.

Bibliothek SYSLNK.CRTE.TIME50

Diese „Bindeschalter“-Bibliothek verlegt den Stichtag (Epoche) für die Zeitfunktionen `ctime`, `difftime`, `ftime`, `gmtime`, `localtime`, `mktime` und `time` auf den 1.1.1950 00:00:00. Damit kann für die genannten Zeitfunktionen das Verhalten eingestellt werden, wie es in den CRTE-Versionen bis V2.9 sowie in V10.0A und V11.0A standardmäßig eingestellt war.

Dieser Bindeschalter ist dann erforderlich, wenn Zeitstempel verarbeitet werden sollen, die mit einem Stichtag (Epoche) 01.01.1950 erzeugt wurden.

Allerdings liefern die oben genannten Zeitfunktionen in diesem Fall für Zeitstempel nach dem 19.01.2018 03:14:07 keine korrekten Ergebnisse mehr. Insbesondere führt der Aufruf der Funktionen `time` und `ftime` zur Beendigung des Programms, da das aktuelle Datum größer ist als der 19.01.2018.

Geben Sie beim Binden die folgende Anweisung an, um den Bindeschalter zu verwenden:

```
INCLUDE-MODULE LIB=<bibliothek>,ELEMENT=*ALL
```

Bibliothek SYSLIB.CRTE

Die Source-Bestandteile des C-Laufzeitsystems (Standard-Include-Elemente und Quellprogramme für benutzereigene Lokalitäten) werden in der Bibliothek SYSLIB.CRTE bereitgestellt. Diese Bibliothek enthält außerdem die ILCS-Makros.

3.3.2 Unterstützung von großen Dateien > 2 GB durch 64-Bit-Funktionen

C/C++-Programme können große BS2000-Dateien (> 2 GB) verarbeiten, wenn sie mit einem C/C++-Compiler übersetzt wurden, der den Datentyp long long unterstützt. Dies ist für C/C++-Compiler >= V3.1 der Fall.

Die maximale Größe einer BS2000-Datei beträgt 4096 Gigabyte (= 4 Terabyte), eine POSIX-Datei kann maximal 1024 Gigabyte (= 1 Terabyte) groß sein.

3.4 Cfront-C++-Bibliotheksfunktionen und -Laufzeitsystem

Die Cfront-C++-Bibliotheksfunktionen für komplexe Mathematik und Standard-Ein-/Ausgabe werden mit den Bibliotheken SYSLNK.CRTE.CFCPP bzw. SYSLNK.CRTE.CPP zur Verfügung gestellt:

- SYSLNK.CRTE.CFCPP enthält das Laufzeitsystem für Cfront-Programme ab C++ V3.0B.
- Das Laufzeitsystem für C++-Programme < V3.0 ist in SYSLNK.CRTE.CPP enthalten.

3.5 ANSI-C++-Bibliotheken und -Laufzeitsysteme

Die folgenden Bibliotheken werden für den C/C++-Compiler ab Version 3.0 zur Verfügung gestellt:

- Für die Nutzung im ANSI-C++-Modus des Compilers die Standard-C++-Bibliothek (SYSLNK.CRTE.STDCPP), das C++-Laufzeitsystem (SYSLNK.CRTE.RTSCPP) und die Bibliothek Tools.h++ (SYSLNK.CRTE.TOOLS).
- Für die Bindetechnik Complete Partial-Bind gibt es die Bibliothek SYSLNK.CRTE.CPP-COMPL. Diese enthält alle Module der oben genannten Bibliotheken. Dabei sind jeweils alle Module einer Bibliothek als ein LLM-Großmodul zusammengebunden.
- Für die Nutzung im Cfront-C++-Modus des Compilers zusätzlich zu den C++-Bibliotheksfunktionen für komplexe Mathematik und streamorientierte I/O (siehe vorhergehender Abschnitt) das Cfront-C++-Laufzeitsystem (SYSLNK.CRTE.CFCPP).

Die folgenden Bibliotheken werden für den C/C++-Compiler ab Version 4.0 zur Verfügung gestellt:

- Für die Nutzung im C++ 2017-Modus des Compilers die Standard-C++-Bibliothek und das C++-Laufzeitsystem (SYSLNK.CRTE.CXX01), die Header-Bibliothek SINLIB.CRTE.101.CXX01 und die Lizenzbibliothek SYSDOC.CRTE.101.CXX01.OSS.

Die Verwendung dieser Bibliotheken ist ausführlich beschrieben im „C/C++ -Benutzerhandbuch“ [9].

3.6 Gemeinsame interne Laufzeitroutinen

Das CRTE enthält gemeinsame Routinen, die sowohl vom COBOL85/COBOL2000- als auch vom C/C++-Laufzeitsystem intern verwendet werden. Die Module dieser internen Routinen sind mit Ausnahme der Speicherverwaltung und der mathematischen Routinen im Subsystem CRTESIS zusammengefasst. Die Speicherverwaltung ist Teil der Subsysteme CRTEC und CRTEPART (Subsysteme siehe "Gemeinsam benutzbares CRTE").

Die Module sind an folgenden Präfixen erkennbar:

IML...	Mathematische Routinen
IT0...	ILCS-Routinen (siehe Kapitel "Programm-Kommunikationsschnittstelle ILCS“)
IT1...	Service-Routinen, z.B. zur Daten- und Speicherverwaltung, zum Meldungswesen sowie für die Garbage Collection für COBOL2000-Programme

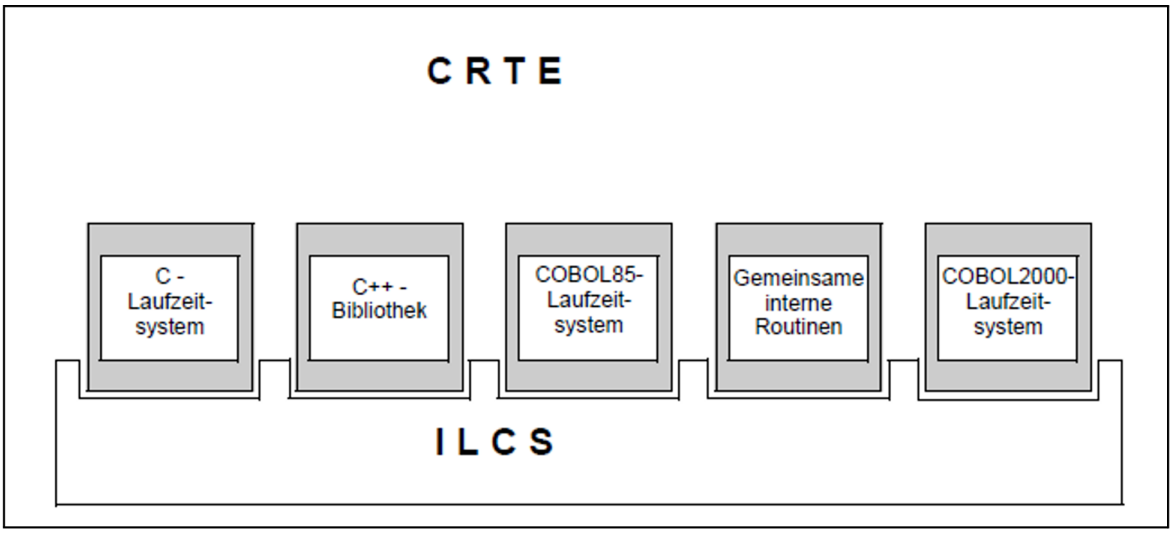
Durch die Verwendung von SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CRTE.COMPL können Programme, die die gemeinsamen internen Laufzeitroutinen benutzen, ohne offene Externbezüge gebunden werden. Die Laufzeitroutinen selbst werden dann erst während des Programmablaufs dynamisch nachgeladen.

4 Programm-Kommunikationsschnittstelle ILCS

Die Programm-Kommunikationsschnittstelle ILCS (**I**nter **L**anguage **C**ommunication **S**ervices) bildet die Basis für die Realisierung des gemeinsamen Laufzeitsystems CRTE. Die Schnittstelle ILCS vereinheitlicht und vereinfacht sprachübergreifend wesentliche Funktionen der Kommunikation zwischen den Programmen einer Ablaufeinheit und zwischen Ablaufeinheit und Betriebssystem.

4.1 Architektur und Funktionalität von ILCS

Eine Reihe von ILCS-Basisfunktionen können vom Sprach-Laufzeitsystem und den Anwendungen auch direkt genutzt werden.



ILCS ist eine Kombination aus Software und Schnittstellen-Konventionen:

- ILCS enthält Laufzeitroutinen, die in der CRTE-Bibliothek zusammengefasst sind
- In ILCS ist auch die den „Standard-Linkage-Konventionen im BS2000“ entsprechende Kommunikationsschnittstelle definiert; d.h. jedes von einem ILCS-fähigen Compiler erzeugte Modul ist entsprechend den Standard-Linkage-Konventionen für die Verknüpfung mit gleich- und verschiedensprachigen Programmen vorbereitet.

Die ILCS-Routinen sind Bestandteil der Liefereinheit CRTE und werden mit dieser immer in der aktuellsten Version ausgeliefert. Sie sind an dem Präfix „IT0“ zu erkennen.

Im Einzelnen bietet ILCS folgende Funktionen:

- Programminitialisierung
- Programmbeendigung
- Konvention zur Verknüpfung verschiedensprachiger Programme
- einheitliche Richtlinien zur Ereignisbehandlung
- Speicherverwaltung (Stack- und Heap-Speicher)
- einheitliche Behandlung der Programmmaske
- Verarbeitung nichtlokaler Sprünge
- ab C/C++ V3.0 Unterstützung der C++-Ausnahmebehandlung

Alle neueren Versionen der BS2000-Compiler sind ILCS-fähig:

Compiler	ab Version
Ada	2.1
ASSEMBH	1.1

C	2.0
C++	2.1A
COBOL85	1.1
COBOL2000	1.1
FOR1	2.2
Pascal-XT	2.2
PLI1	4.1
RPG3	3.0
UX-Basic	3.0B

Ferner ist auch openUTM ab Version 3.3 an ILCS angepasst.

Programme, die mit ILCS-fähigen Compilern übersetzt wurden, lassen sich auf einfache Weise mittels ILCS zu einem Programmsystem verknüpfen. Enthält ein Programmsystem Programme, die sich nicht gemäß den ILCS-Konventionen verhalten, müssen sie ggf. so

umgestaltet werden, dass sie den ILCS-Konventionen entsprechen. Andernfalls besteht- zumindest bei der Verknüpfung verschiedensprachiger Programme - die Gefahr der Inkompatibilität.

ILCS besteht aus folgenden Modulen:

- dem Codemodul IT0SL@ und
- dem Datenmodul IT0SL#.

ILCS (CRTECOM) wird seit CRTE V2.1 nicht mehr als vorladbares Subsystem ausgeliefert. Beim Binden von Anwendungen, die CRTE voraussetzen, muss sichergestellt werden, dass immer die ILCS aus dem CRTE eingebunden wird. Insbesondere dürfen weder ein alter Modul IT0INITS aus einer der Sprachlaufzeitsystem-Bibliotheken noch Module aus der $.SYSLNK.ILCS nachgebunden werden. Deshalb muss zum Auflösen von Externreferenzen die Bibliothek $.SYSLNK.CRTE vor allen anderen Sprachlaufzeitsystem-Bibliotheken durchsucht werden. Desweiteren ist darauf zu achten, dass ILCS und Laufzeitsystem-Module in einer Anwendung nicht mehrfach eingebunden sein dürfen. Ausgenommen hiervon sind abgeschottete Subsysteme.

4.2 ILCS-Konventionen

Dieser Abschnitt informiert über folgende Themen:

- Registerkonventionen
- Datenstrukturen
- Programmmasken-Behandlung durch ILCS

4.2.1 Registerkonventionen

Registerversorgung bei Aufruf

Die nachfolgende Tabelle gibt eine Übersicht über die Registerversorgung, die das aufrufende Programm vor dem Ansprung des aufgerufenen Programms durchführt.

Registernummer	Inhalt
0	Anzahl der Parameter
1	Anfangsadresse der Parameter-Adressliste
2 - 12	undefiniert
13	Anfangsadresse des Sicherstellungsbereiches (Save Area) des aufrufenden Programms
14	Adresse des Rückkehrpunktes ins aufrufende Programm
15	Adresse des Einsprungpunktes im aufgerufenen Programm
PM	Programmmaske: Wert aus PCD-Feld „Programmmaske“ (vorbelegt mit X'0C')

Registerversorgung bei Rückkehr ins aufrufende Programm

Die nachfolgende Tabelle gibt eine Übersicht über die Registerversorgung, die das aufgerufene Programm beim Rücksprung ins aufrufende Programm durchführt.

Registernummer	Inhalt
0 und 1	Returnwerte von Ganzzahl-Funktionen oder undefiniert
Gleitpunktregister 0	Returnwerte von Gleitpunkt-Funktionen oder undefiniert
2 - 14	wie bei Aufruf-Versorgung
15	undefiniert
PM	Programmmaske: Wert aus PCD-Feld „Programmmaske“

4.2.2 Datenstrukturen

Sicherstellungsbereich (Save Area)

Das aufrufende Programm liefert in Register 13 die Adresse eines Sicherstellungsbereiches, in dem das aufgerufene Programm die aktuellen Registerstände ablegen kann. Das aufgerufene Programm legt einen neuen Sicherstellungsbereich an und verkettet die beiden Sicherstellungsbereiche.
Der Sicherstellungsbereich ist folgendermaßen aufgebaut:

Byte	Inhalt
1-4	1. Byte: 1. Bit: Aktivitätsbit (1: Programm aktiv, 0: Programm inaktiv) 2.-8. Bit: reserviert 2. Byte: Version = X' 01' 3. und 4. Byte: X' FEFF'
5-8	enthält die Anfangsadresse des Sicherstellungsbereiches des aufrufenden Programms. Im **ersten** aufrufenden Programm ist der Inhalt dieses Feldes -1.
9-12	enthält ggf. die Anfangsadresse des nächsten Sicherstellungsbereiches.
13-16	Inhalt von Register 14
17-20	Inhalt von Register 15
21-24	Inhalt von Register 0
25-28	Inhalt von Register 1
29-32	Inhalt von Register 2
. .	
69-72	Inhalt von Register 12
73-76	reserviert für FOR1
77-80	Adresse der PCD
81-84	Adresse der EHL (Event Handler List): Ist keine EHL definiert, enthält das Feld den Wert -1.
85-128	reserviert

Prosys Common Data Area (PCD)

Die PCD ist ein allgemeiner Datenbereich, der von allen Programmiersprachen verwendet wird. Der erste Teil enthält die von ILCS verwendeten Datenbereiche, u.a. auch das Feld „Programmmaske“, das mit dem Wert X’0C’ vorbelegt ist. Der zweite Teil der PCD enthält die jeweils 128 Byte großen Programmiersprachen-Bereiche, die den Laufzeitsystemen der verschiedenen Sprachen zur Verfügung stehen.

4.2.3 Programmmasken-Behandlung durch ILCS

Die Programmmaske für den Programmablauf wird bei der Initialisierung auf den Wert des PCD-Feldes „Programmmaske“ gesetzt (vorbelegt mit X’0C’: Fixpunkt- und Dezimal-Überlauf führen zum Abbruch, Exponenten-Unterlauf und Mantisse Null führen nicht zum Abbruch). Wird sie während des Programmablaufs verändert, muss sie vor dem nächsten Programmaufruf bzw. Rücksprung auf den Wert des PCD-Feldes „Programmmaske“ zurückgesetzt werden.
In C/C++ kann die Programmmaske mit der RUNTIME-OPTIONS-Option verändert werden (siehe C- und C++-Benutzerhandbücher), in Assembler mit dem ASSEMBH-Strukturmakro @SETPM (siehe ASSEMBH-Beschreibung [12]).

4.3 Initialisierung

Dieser Abschnitt informiert über folgende Themen:

- Initialisierung von ILCS
- Initialisierung bei dynamischem Nachladen von Programmen

4.3.1 Initialisierung von ILCS

Die Initialisierung von ILCS erfolgt in der Regel automatisch.
Falls ILCS ausnahmsweise nicht initialisiert sein sollte, kann es vor dem Ansprung des ersten ILCS-Moduls durch den Aufruf der Initialisierungsroutine **IT0INITS** in einem AssemblerProgramm nachinitialisiert werden (siehe auch Beispiel im Anhang).
IT0INITS wird ohne Parameter aufgerufen. Außerdem muss Register 0 mit dem Wert 0 belegt werden (Anzahl der Parameter=0), da die Routine für interne Zwecke bis zu 9 Parameter besitzen kann. IT0INITS liefert in Register 0 die Adresse der PCD.
Register 15 enthält folgende Returncodes:

Returncode	Bedeutung
0	Initialisierung von ILCS erfolgreich durchgeführt
1	IT0INITS wurde bereits einmal aufgerufen
2	BS2000-Version wird nicht unterstützt
3	Versionsunverträglichkeit zwischen Code und Daten
4	Speichermangel bei der Initialisierung der Stack-Verwaltung
5	Speichermangel bei der Initialisierung der Heap-Verwaltung
6	Standard Event Handler konnte nicht initialisiert werden
7	Eine Sprach-Initialisierungsroutine liefert einen Fehler

4.3.2 Initialisierung bei dynamischem Nachladen von Programmen

ILCS führt bei Programmstart die notwendigen Initialisierungen für alle am Programmsystem beteiligten Sprachen durch.Die Beteiligung einer Sprache wird jedoch nur erkannt, wenn die Module einer Sprache statisch eingebunden sind oder wenn sie vom dynamischen Bindelader erkannt werden.

Wenn in Assemblerprogrammen Programmteile mit dem BIND-Makro dynamisch nachgeladen werden, die Objekte in bisher nicht beteiligten Sprachen enthalten, so sind für diese Sprachen die notwendigen Initialisierungen noch nicht durchgeführt.
Für diesen Fall muss nach dem dynamischen Nachladen in einer Assemblerroutine die ILCS-Routine IT0ININ aufgerufen werden. Für den Aufruf von IT0ININ steht das ASSEMBH-Strukturmakro @ININ zur Verfügung (siehe Beispiel im Anhang und Handbuch „ASSEMBH-Beschreibung“[12]). Bei Aufruf von IT0ININ muss ILCS bereits initialisiert sein.

Beim dynamischen Nachladen von Unterprogrammen in COBOL85/COBOL2000-Programmen mit „CALL bezeichner“ wird der Aufruf von IT0ININ intern automatisch durch das COBOL85/COBOL2000-Laufzeitsystem durchgeführt.

Besonderheit in C++

Auch wenn C++-Programmteile bereits vorhanden sind, muss nach jedem dynamischen Nachladen eines C++-Programmteils die ILCS-Routine IT0ININ aufgerufen werden. Nur so ist gewährleistet, dass Aufrufe für globale Konstruktoren durchgeführt werden.

4.4 Abgeschottete Subsysteme

Ein Subsystem, das auf eigenen globalen Daten arbeitet, kann sich abschotten.

Dies bietet folgende Vorteile:

- Das Subsystem ist unabhängig von unterschiedlichen Versionen einer Komponente, die sowohl in der Umgebung als auch im Subsystem enthalten ist.
- Die internen Zustände von Sprachlaufzeitsystemen der Anwendung und des Subsystems beeinflussen sich nicht gegenseitig.

Andererseits bedingt die Abschottung folgende Einschränkungen:

- Globale Sprünge (z.B. longjump in C) aus dem Subsystem in die Anwendung sind nicht möglich.
- Ereignisse aus dem Subsystem können nicht mehr in die Anwendung propagiert werden.

4.4.1 Anforderungen an ein abgeschottetes Subsystem

Die oben genannten Besonderheiten führen zu folgenden Anforderungen an ein abgeschottetes Subsystem:

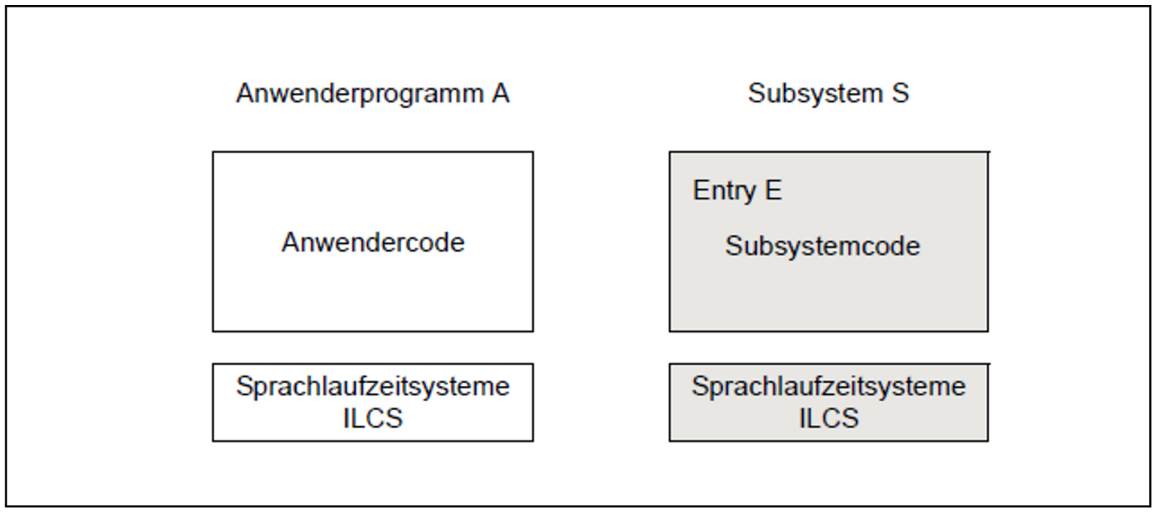
- Das Subsystem benötigt „private“ Kopien der verwendeten Laufzeitsysteme und der ILCS. Die Entries dieser Komponenten dürfen nach außen nicht mehr sichtbar sein.
- Das Subsystem muss einen Schalter enthalten, der die Initialisierung des Subsystems beim ersten Aufruf steuert.
- Das Subsystem benötigt Variablen für die Adressen der Save Area der Anwendung sowie für die erste Save Area des Subsystems. In der ersten Save Area des Subsystems müssen die Felder PSA und EHL mit dem Wert F‘-1‘ belegt sein.

Darüber hinaus sind beim Binden von Subsystemen, beim Umschalten auf die Subsystemumgebung sowie bei der Ereignis- und Contingency-Behandlung einige Gesichtpunkte zu beachten, die in den nachfolgenden Abschnitten beschrieben sind.

4.4.2 Architektur eines abgeschotteten Subsystems

Ein Subsystem wird mit allen benötigten Komponenten vorgebunden. Soll das Subsystem vorladbar sein, dann müssen Code- und Datenteile getrennt gebunden werden.

Die folgende Abbildung skizziert ein abgeschottetes Subsystem.



4.4.3 Für die Abschottung erforderliche Aktionen

Eine aufgerufene Subsystem-Funktion muss folgende Aktionen ausführen:

1. Register des Aufrufers in dessen Save Area sichern und die Adresse dieser Save Area in der entsprechenden Variablen sichern.
2. Beim ersten Aufruf (Schalter gesetzt) IT0INITS zur Initialisierung der ILCS und der Sprachlaufzeitsysteme aufrufen und die von IT0INITS zurückgelieferte PCD-Adresse in der Save Area des Subsystems eintragen (danach den Schalter ausschalten).
3. SEH-Ereignisbehandlung des Subsystems mit IT0SEHEN abschalten, wenn das Subsystem keine explizite Ereignisbehandlung durchführt.
   Alternativ dazu kann die Ereignisbehandlung des rufenden Programms mit IT0CRDEA deaktiviert und die ILCS des Subsystems mit IT0CRACT aktiviert werden, wenn der Schalter nicht gesetzt ist.
4. Adresse der ersten Save Area des Subsystems in Register 13 laden. Damit wird auf die ILCS des Subsystems umgeschaltet und der Code der aufgerufenen Funktion kann ausgeführt werden.

Vor dem Rücksprung zur Anwendung müssen folgende Aktionen ausgeführt werden:

1. Falls die Ereignisbehandlung des Subsystem mit IT0CACT aktiviert wurde, sollte sie mit IT0CDEA bis zum nächsten Aufruf des Subsystems deaktiviert werden, sofern dem Subsystem keine asynchronen instanzspezifischen Ereignisse zugestellt werden sollen, solange es nicht aktiv ist.
   Die ILCS des rufenden Programms sollte wieder aktiviert werden (IT0CACT).
2. Die Register der Anwendung müssen geladen werden und in Register 13 muss die Adresse der Save Area der Anwendung geladen werden.
3. Das Funktionsergebnis der aufgerufenen Subsystem-Funktion muss an die Anwendung weitergegeben werden.

4.4.4 POSIX-Nutzung in Subsystemen

POSIX-Schnittstellen dürfen nur von *einem* Subsystem oder von der Anwendung genutzt werden, weil es sonst zu einer gegenseitigen Beeinflussung der Standard-Ströme kommt.

4.4.5 STXIT-Ereignisse und Contingencies

Die Zustellung von Ereignissen an die verschiedenen ILCS-Instanzen wird durch die Aktivierung bzw. Deaktivierung der entsprechenden PCD‘s gesteuert.
Dabei wird zwischen instanz-spezifischen und task-spezifischen Ereignissen und Contingencies unterschieden.

Werden keine speziellen Vorkehrungen getroffen, bekommt das Subsystem Programmfehler (STXIT-Klasse PROCHK und ERROR) auch dann mitgeteilt, wenn der Fehler gar nicht im Subsystem aufgetreten ist.
Enthält die Anwendung mehrere Komponenten mit privaten ILCS-Instanzen, dann wird das Verhalten im Fehlerfall durch die Reihenfolge der SEH-Anmeldungen beim Betriebssystem beeinflusst, was in der Regel unerwünscht ist.
Wenn das Subsystem alle Ereignisse ignorieren kann, sollte der SEH mittels IT0SEHEN deaktiviert werden. Dieses Verfahren hat allerdings den Nachteil, dass auch die Fehlerbehandlung der Sprachlaufzeitsysteme deaktiviert wird.

4.4.6 Entladen eines Subsystems

Ein Subsystem kann beendet und anschließend entladen werden, ohne dass die gesamte Anwendung beendet werden muss.
Dies geschieht durch den Aufruf von IT0CRTRM.

4.5 Verknüpfung verschiedensprachiger ILCS-Programme

Dieser Abschnitt informiert über folgende Themen:

- Parameterübergabe
- Übergabe von Funktions-Returnwerten
- Ereignisbehandlung durch ILCS

4.5.1 Parameterübergabe

Die Darstellung der Datentypen weist bei den durch ILCS verknüpfbaren Programmiersprachen starke Unterschiede auf. Im Folgenden werden jene Datentypen aufgeführt, die in den einzelnen Programmiersprachen eine gleiche Datendarstellung besitzen und daher problemlos als Parameter übergeben werden können. Bei der Verwendung anderer Datentypen als Parameter ist die genaue Kenntnis der jeweiligen Datenablage erforderlich, um den korrekten Programmablauf sicherzustellen.

Die Übergabe erfolgt „by reference“, d.h. es wird die Adresse des Datums übergeben.

Compiler	**Datentypen**			
	Binär Wort	Gleitpunkt Wort	Gleitpunkt Doppelwort	String
C / C++	long	float	double	char <var> [<size>]
COBOL2000 COBOL85	PIC S9(i) COMP-5 SYNC 5 <= i <= 9	COMP-1	COMP-2	USAGE DISPLAY
ASSEMBH (@-Makros)	F	E	D	C
FOR1	INTEGER*4	REAL*4	REAL*8	CHARACTER*i
Pascal-XT	long_integer	short_real	long_real	packed array [<range>] of char
PLI1	BIN FIXED(31) ALIGNED	BIN FLOAT(21) DEC FLOAT(6)	BIN FLOAT(53) DEC FLOAT(16)	CHAR(i)
RPG3	binäres Feld, 0 Dezimalstel.			alphanum. Feld, feste Länge
UX-Basic	%%,?	!	!!	$

Kompatible Datentypen bei Parameterübergabe

Das aufrufende Programm legt eine Liste der übergebenen Adressen an. Die Anzahl der Parameter wird in Register 0, die Adresse der Liste in Register 1 übergeben (siehe Abschnitt "Registerkonventionen“).

Parameterübergabe in Assembler

Die Daten müssen immer ausgerichtet abgelegt werden; d.h. 32-Bit-Ganzzahlen in binärer Darstellung liegen auf Wortgrenze, Gleitpunktzahlen auf Wort- bzw. Doppelwortgrenze, long long auf Doppelwortgrenze, Strings auf Bytegrenze. Die Länge von Strings ist konstant und dem gerufenen Programm bekannt.

Parameterübergabe in C und C++

In C/C++ werden Parameter laut Sprachdefinition standardmäßig „by value“ übergeben. Diese Art der Parameterübergabe ist uneingeschränkt nur bei Programmsystemen möglich, in denen ausschließlich die Sprachen C und C++ beteiligt sind.

Bei der Verknüpfung mit Objekten in anderen Sprachen müssen folgende Maßnahmen getroffen werden:

- Bei Aufruf der anderssprachigen Routine muss in C/C++ die Adresse des zu übergebenden Datenelements angegeben werden, z.B. &par. Technisch gesehen wird dann die Adresse des Datenelements „by value“ übergeben.
- Wird eine C/C++-Routine von einer anderssprachigen Routine aufgerufen, müssen die Formalparameter in C /C++ als Zeiger auf die zu übergebenden Datenelemente deklariert werden, z.B. f (T *par).

Parameterübergabe in COBOL

In COBOL85 und COBOL2000 werden Parameter standardmäßig „by reference“ übergeben (siehe Tabelle).

Parameterübergabe „by value“

COBOL2000 kann beliebige numerische Parameter „by value“ übergeben, wenn für das aufgerufene Programm ein Prototyp existiert. Existiert kein Prototyp, ist nur eine eingeschränkte Parameterübergabe „by value“ möglich (siehe nächste Seite).

Lage und Ausrichtung der Daten in der Parameterliste hängen von der Datenbeschreibung im Prototyp ab. Die folgenden Tabellen zeigen im einzelnen, wie sich die Datenbeschreibungen auswirken:

Binärdaten PIC [S]9(i) USAGE {COMP | COMP-5 | BINARY}	**Darstellung in Parameterliste**
Halbwort (1 <= i <= 4)	rechtsbündig in ausgerichtetem Ganzwort (Byte 1 u. 2 undefiniert)
Ganzwort (5 <= i <= 9)	ausgerichtetes Ganzwort
Doppelwort (10 <= i <= 18)	ausgerichtetes Doppelwort (zur Ausrichtung eingefügte Bytes undefiniert)

Gleitpunktdaten	**Darstellung in Parameterliste**
Wort (USAGE COMP-1)	ausgerichtetes Ganzwort
Doppelwort (USAGE COMP-2)	ausgerichtetes Doppelwort (zur Ausrichtung eingefügte Bytes undefiniert)

Gepackte (USAGE COMP-3/PACKED DECIMAL) oder abdruckbare (USAGE DISPLAY) numerische Daten sollten nur zwischen COBOL2000-Programmen „by value“ übergeben werden. In der Parameterliste werden sie linksbündig auf Wortgrenze (Länge <8 Byte) bzw. Doppelwortgrenze (Länge >= 8 Byte) abgelegt.

Existiert kein Prototyp, können mit COBOL2000 nur Datenfelder mit den folgenden Definitionen „by value“ übergeben werden:

- COMP-5 PIC [S]9(i) (1 <= i <= 9) oder
- 1 Byte langes Datenfeld beliebigen Typs

Der Parameter wird gegebenenfalls auf 4 Byte erweitert, indem linksbündig mit binären Nullen aufgefüllt wird.

Diese eingeschränkte Übergabe „by value“ ist auch mit COBOL85 ab Version 2.1B möglich.

Übergabe von OMITTED (nur bei COBOL2000 möglich)

Parameter, die bei einem Unterprogrammaufruf weggelassen werden (Angabe OMITTED beim Aufruf, OPTIONAL im Prototyp), werden als ausgerichtetes Ganzwort mit dem Wert X'FFFFFFFF' in der Parameterliste abgelegt.

Beim OMITTED-Test in einem gerufenen COBOL2000-Programm wird jeder Parameter, der mit der Adresse X' FFFFFFFF' in der Parameterliste steht, als nicht übergeben (OMITTED) gewertet.

Parameterübergabe in Java

Datenaustausch zwischen C- und Java-Programmen ist möglich, allerdings nicht über die ILCS-Schnittstelle. Informationen zum Datenaustausch zwischen C/C++ und Java entnehmen Sie bitte der Dokumentation zu Java.

4.5.2 Übergabe von Funktions-Returnwerten

Returnwerte von Ganzzahl-Funktionen werden in den Registern 0 und 1, Returnwerte von Gleitpunkt-Funktionen werden im Gleitpunktregister 0 übergeben.
Die Übergabe von Returnwerten mit anderen Datentypen in Register 0 und 1 ist möglich, aber nicht durch ILCS festgelegt. Ihre Darstellung ist den einzelnen Programmiersprachen freigestellt.
COBOL-Unterprogramme ab COBOL85 V2.0 bzw. COBOL2000 V1.0 verhalten sich immer wie Ganzzahlfunktionen und liefern dem aufrufenden Programm in Register 0 und 1 den Inhalt des COBOL-Sonderregisters RETURN-CODE zurück.

Ab der COBOL85-Version 2.1B bzw. COBOL2000 V1.0 kann mittels einer Steueranweisung über das COBOL-Sonderregister RETURN-CODE auf den Returnwert eines aufgerufenen C-Unterprogramms zugegriffen werden (siehe „COBOL85- bzw. COBOL2000 Benutzerhandbuch“ [2, 4]).

4.5.3 Ereignisbehandlung durch ILCS

ILCS unterscheidet zwischen einer impliziten sprachspezifischen Ereignisbehandlung und einer explizit an- und abmeldbaren Ereignisbehandlung. Für die implizite Ereignisbehandlung steht die ILCS-Schnittstelle „Standard-Event-Handler“ (SEH) zur Verfügung, für die explizite Ereignisbehandlung der „Standard-STXIT-Handler“ (SSH).

Standard-Event-Handler SEH

Über die SEH-Schnittstelle ist eine Ereignisbehandlung nur für die STXIT-Ereignisse ERROR und PROCHK möglich. Diese Schnittstelle wird intern von Laufzeitsystemen genutzt, um die in der jeweiligen Programmiersprache vorgesehenen Sprachmittel für Ausnahmebehandlung zu realisieren (z.B. Pascal, Pl/1, Ada), oder um bei Auftritt eines ERROR- oder PROCHK-Ereignisses vor dem Programmabbruch noch bestimmte Maßnahmen, z.B. die Ausgabe von Fehlermeldungen, zu treffen (z.B. COBOL85).

Standard-STXIT-Handler SSH

Über die SSH-Schnittstelle können benutzereigene Routinen für sämtliche BS2000-STXIT-Ereignisse explizit an- und abgemeldet werden. Diese Schnittstelle wird derzeit vom C/C++- und vom ASSEMBH-Laufzeitsystem genutzt: C/C++-Programme können benutzereigene Routinen zur Ereignisbehandlung mit den C-Bibliotheksfunktionen signal und cstxit anmelden, Assemblerprogramme mit den ASSEMBH-Strukturmakros @STXEN und @STXDI.

Die explizite Anmeldung von benutzereigenen Routinen für die STXIT-Ereignisklassen ER-ROR und PROCHK (SSH) setzt die implizite Behandlung dieser Ereignisse durch andere Sprachen (SEH) außer Kraft. Um die implizite Ereignisbehandlung anderer Sprachen (COBOL85, Pascal-XT usw.) wieder wirksam werden zu lassen, müssen bei Verlassen der C- bzw. der ASSEMBH-Sprachumgebung die Ereignisbehandlungsroutinen für ERROR und PROCHK explizit abgemeldet werden. In C/C++ erfolgt die Abmeldung durch Zuordnung von SIG_DFL bei der C-Bibliotheksfunktion signal, in Assembler mit dem ASSEMBH-Strukturmakro @STXDI.

Direkt beim BS2000 angemeldete STXIT-Routinen

STXIT-Routinen können durch Verwendung des BS2000-Makros STXIT in Assemblerprogrammen auch direkt beim BS2000 angemeldet werden. Diese Routinen entziehen sich jedoch der Koordinierung durch ILCS; u.a. bleibt bei der direkten Anmeldung von STXIT-Routinen für PROCHK- und ERROR-Ereignisse die implizite Behandlung dieser Ereignisse über die SEH-Schnittstelle aktiv.

4.6 Verknüpfung von ILCS- mit Nicht-ILCS-Programmen

Dieser Abschnitt informiert über folgende Themen:

- Verknüpfung von ILCS-Programmen mit Nicht-ILCS-Programmen gleicher Sprache
- Verknüpfung von ILCS-Programmen mit Nicht-ILCS-Programmen anderer Sprache

4.6.1 Verknüpfung von ILCS-Programmen mit Nicht-ILCS-Programmen gleicher Sprache

Die Verknüpfung von ILCS- mit gleichsprachigen Nicht-ILCS-Programmen (= Programme, die nicht von einem ILCS-fähigen Compiler erzeugt wurden) sind je nach Sprache (Compiler und zugehöriges Laufzeitsystem) unterschiedlich geregelt. Im folgenden werden nur für die CRTE-Sprachen COBOL und C sowie für Assembler die Regelungen aufgeführt, die bei der Verknüpfung von Nicht-ILCS- mit ILCS-Programmen zu beachten sind. Die Regelungen für die anderen Sprachen (z.B. Fortran, Pascal) sind in den entsprechenden Benutzerhandbüchern beschrieben.

COBOL85

Die Verknüpfung von Nicht-ILCS-Programmen (d.h. mit COBOL85 V1.0 erzeugte Programme) mit ILCS-Programmen ist grundsätzlich möglich. Die Programmkommunikation über ILCS ist aber nur gewährleistet, wenn das Hauptprogramm ein ILCS-Programm ist. Wenn das Hauptprogramm ein Nicht-ILCS-Programm ist, gelten auch alle aufgerufenen ILCS-Unterprogramme als Nicht-ILCS-Programme.

COBOL2000

Die Verknüpfung von Nicht-ILCS-Programmen (d.h. mit COBOL85 V1.0 erzeugte Programme) mit ILCS-Programmen ist grundsätzlich möglich.
Da COBOL2000 intern in jedem Fall die ILCS-Infrastruktur nutzt, wird beim Aufruf von COBOL2000-Modulen ILCS initialisiert, auch wenn das Hauptprogramm ein Nicht-ILCS-Programm ist.

> ¡ Bei Verwendung von COBOL-Programmen mit USE-Prozeduren ist eine gültige Rückwärtsverkettung in der Save-Area erforderlich. X'FFFFFFFF' bzw. X'00000000' werden als Ende der Rückwärtsverkettung interpretiert.

C und C++

Nicht-ILCS-Programme (d.h. mit C V1.0 erzeugte Programme) und ILCS-Programme in C können problemlos miteinander verknüpft werden. Falls das aufrufende Hauptprogramm ein C-V1.0-Programm ist, wird vom C-Laufzeitsystem die ILCS-Umgebung automatisch nachinitialsiert. Die Verknüpfung ist bis zur C-Version 2.0 ohne jegliche Vorkehrung möglich, ab der C- bzw C++-Version 2.1A nur unter folgenden Voraussetzungen: Bei der Übersetzung muss in der COMPILER-ACTION-Option die Voreinstellung INLINE-OPTIMAL des LINKAGE-Parameters auf OUT-OF-LINE oder V1-COMPATIBLE-INLINE geändert werden.
Die Übersetzung eines C-Programms mit dem C++-Compiler muss im KR- oder ANSI-Modus erfolgen. Programme, die im CPLUSPLUS-Modus übersetzt werden, sind generell nicht mit C-V1.0-Objekten verknüpfbar.

Assembler

Der Aufruf eines ILCS-Programms durch ein Nicht-ILCS-Programm verlangt die Anpassung des Sicherstellungsbereichs (Save Area) des Nicht-ILCS-Programms an die ILCS-Konvention. Diese Anpassung kann der Benutzer durch folgende Maßnahmen vollziehen:

- Der Sicherstellungsbereich des Nicht-ILCS-Objekts muss mit dem Makro IT0VSA erzeugt werden.
- Vor dem Aufruf des ILCS-Objekts muss das Feld &P.VPCD mit der Adresse des aktuellen IT0PCD-Moduls versorgt werden. Diese Adresse lässt sich mit der Adresskonstanten A(IT0PCD) bestimmen.

Ein Beispiel hierzu finden Sie im Anhang.

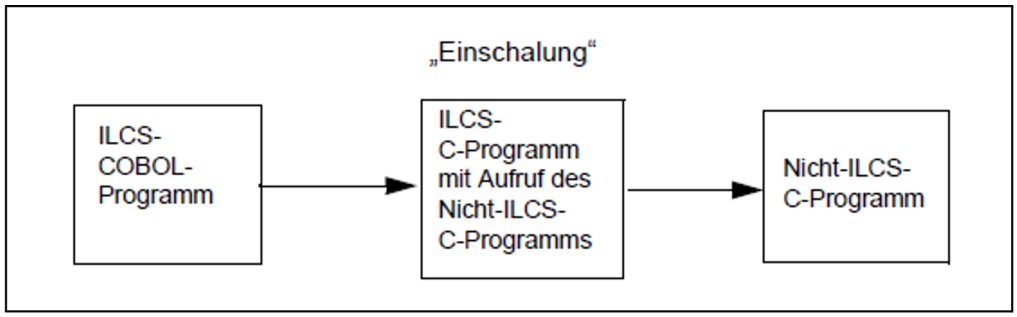
Java

Für die Verknüpfung von C/C++-Programmen und Java-Programmen gibt es verschiedene Techniken, z.B. die Verknüpfung über das Java Native Interface (JNI). Nähere Informationen hierzu finden Sie in der Dokumentation zu Java.

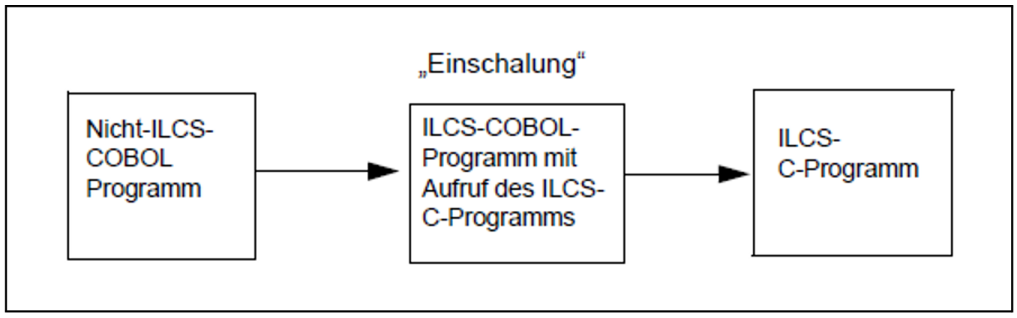
4.6.2 Verknüpfung von ILCS-Programmen mit Nicht-ILCS-Programmen anderer Sprache

Die Verknüpfung von Nicht-ILCS-Programmen mit ILCS-Programmen anderer Sprachen ist nur möglich, wenn die Nicht-ILCS-Programme durch ILCS-Programme gleicher Sprache „eingeschalt“ werden. D.h. der Benutzer muss ein Programm schreiben, welches das Nicht-ILCS-Programm aufruft, und dieses Programm mit einem ILCS-fähigen Compiler übersetzen..

Beispiel: ILCS-COBOL-Programm ruft Nicht-ILCS-C-Programm



Beispiel: Nicht-ILCS-COBOL-Programm ruft ILCS-C-Programm



4.7 Fehlerbehandlung

ILCS meldet normalerweise Fehlersituationen über einen Returncode an das rufende Programm. In den folgenden Fehlersituationen ist das jedoch nicht möglich, da es keinen Rufer gibt:

- Es tritt eine Ausnahmesituation in einer Unterbrechungsroutine auf (z.B. Ressourcenmangel)..
- Es wird aus IT0CREND in den Koroutinenadapter zurückgesprungen.

Diese Situationen werden in ILCS als *interne* Fehler behandelt.

Ebenso werden die folgenden Fehlersituationen beim Standard-Event-Handler (SEH) als interne Fehler behandelt, bei denen die normale Fehlerbehandlung des SEH - Wiederauslösen des Fehlers - nicht möglich ist:

- Datenfehler. Dabei handelt es sich um SEH-Ausnahmen, die die Daten so verändert haben, dass die Ausnahmen nicht wieder ausgelöst werden können.
- Eine SEH-Behandlungsroutine meldet, dass ein STXIT-Ereignis wie ein Datenfehler zu behandeln ist.

Bei internen Fehlern wird

- eine Fehlermeldung ausgegeben (Voraussetzung für die Ausgabe einer Fehlermeldung ist CRTE-MSG V1.3)
- Debug-Information in der Prosys Common Data Area (PCD) hinterlegt und
- die Anwendung mit dem Makro TERM abnormal beendet.

Die Felder in der PCD werden beim ersten Fehler, der einen Abbruch verursacht, eingetragen. Folgefehler bleiben unberücksichtigt.
Die PCD-Felder werden auch dann versorgt, wenn die abnormale Beendigung durch den TERM-Makro durch Angabe eines Benutzerexits abgefangen worden ist.

Die möglichen Werte der PCD-Felder M7TYP, M7DIA und M7ADR sind im Folgenden aufgelistet.

Der Anwender kann auch den TERM-Makro über den Inhalt bestimmter PCD-Felder steuern. Die Initialisierung der entsprechenden Felder ist im Abschnitt "Steuerung des TERM-Makros durch den Anwender“ beschrieben.

4.7.1 Debug-Informationen in der PCD

In den Feldern der PCD werden folgende Informationen hinterlegt (die Einträge beziehen sich auf den ersten Fehler, der einen Abbruch verursacht. Folgefehler bleiben unberücksichtigt):

1. Im Feld M7TYP der Fehlertyp:

Wert	Meldungsnummer	Bedeutung
0	CCM0200	Datenfehler
1	CCM0201	SEH signalisiert einen Datenfehler
2	CCM0202	Rückkehr aus dem Koroutinenadapter
3	CCM0203	Speichermangel SSH (Standard STXIT Handler)
4	CCM0204	Speichermangel SCH (Standard Contingency Handler)
5	CCM0205	systeminterner Fehler (Fehler beim Heap- oder Stack-Umschalten o.ä.)
6	CCM0205	LEVCO meldet Returncode beim SSH
7	CCM0205	LEVCO meldet Returncode beim SCH

2. Im Feld M7DIA: Das STXIT-Unterbrechungsgewicht oder der Returncode eines SVC.
3. Im Feld M7ADR: Adresse in der ILCS, an der der Fehler aufgetreten ist.

4.7.2 Steuerung des TERM-Makros durch den Anwender

Beim Aufruf des TERM-Makros durch ILCS werden die Parameter UNIT, DUMP, MODE und URETCD mit den Werten versorgt, die in den PCD-Feldern AUNT, ADMP, AMDE und AMJV stehen.

Die folgende Tabelle zeigt, wie die PCD-Felder AUNT, ADMP, AMDE und AMJV initialisiert sind. Als Anwender haben Sie die Möglichkeit, diese PCD-Felder zu modifizieren (indem Sie den gewünschten Wert per Assemblerbefehl in das PCD-Feld schreiben) und damit das Verhalten des TERM-Makros zu beeinflussen.

Inhalt des PCD-Felds	**entsprechender Parameterwert beim TERM-Makro**
AUNT = X‘02‘	UNIT = STEP
ADMP = X‘01‘	DUMP = Y
AMDE = X‘04‘	MODE = ABNORMAL
AMJV = CL4'3'	URETCD = CL4‘3‘

4.7.3 Meldungstexte

CCM0200 DATENFEHLER: UG=‘(&00)‘

CCM0200 DATA EXCEPTION: EW=‘(&00)‘

Bedeutung

Es ist ein nicht behandelter Datenfehler aufgetreten wie zum Beispiel eine Division durch Null, OVERFLOW o.ä. UG gibt das Unterbrechungsgewicht an.

Maßnahme

Korrektur des Fehlers im Programm.

CCM0201 SEH-Routine MELDET DATENFEHLER: UG=‘(&00)‘

CCM0201 DATA EXCEPTION REPORTED BY SEH-Routine: EW=‘(&00)‘

Bedeutung

Eine SEH-Routine meldet per Returncode, dass das SEH-Ereignis wie ein unbehandelter Datenfehler zu behandeln ist.

Maßnahme

Korrektur des Fehlers im Programm.

CCM0202 ILLEGALER RUECKSPRUNG AUS IT0CREND

CCM0202 ILLEGAL RETURN FROM IT0CREND

Bedeutung

IT0CREND kehrt in den Koroutinenadapter zurück.

Maßnahme

Bitte wenden Sie sich an Ihre Systembetreuung.

CCM0203 SPEICHERMANGEL BEI START EINES STXIT-PROZESSES

CCM0203 NOT ENOUGH MEMORY SPACE AVAILABLE ON STARTING A STXIT PROCESS

Bedeutung

Resourcenengpass.

Maßnahme

Prüfen Sie, ob eine Schachtelung von Unterbrechungsaufrufen vorliegt oder eine Aufrufschleife.Falls dies nicht der Fall ist, wenden Sie sich bitte an Ihren Systemverwalter wegen der Zuweisung von mehr Speicherplatz.

CCM0204 SPEICHERMANGEL BEI START EINES CONTINGENCY-PROZESSES

CCM0204 NOT ENOUGH MEMORY SPACE AVAILABLE ON STARTING A CONTINGENCY PROCESS

Bedeutung

Resourcenengpass.

Maßnahme

Prüfen Sie, ob eine Schachtelung von Unterbrechungsaufrufen vorliegt oder eine Aufrufschleife.Falls dies nicht der Fall ist, wenden Sie sich bitte an Ihren Systemverwalter wegen der Zuweisung von mehr Speicherplatz.

`CCM0205 INTERNER FEHLER IN ILCS`

`CCM0205 INTERNAL ILCS ERROR`

Bedeutung

In ILCS ist ein interner Fehler aufgetreten.

Maßnahme

Bitte wenden Sie sich an Ihre Systembetreuung.

5 Binden mit CRTE

Ein Quellprogramm, das von einem Compiler übersetzt wurde, liegt entweder im Bindemodul-Format oder im Bindelademodul-Format vor. Bindemodule (Object Module, OM) und Bindelademodule (Link and Load Module, LLM) sind Eingabeobjekte für das System Binder-Lader-Starter (BLS), das aus diesen Objekten ablauffähige Programme erzeugt.

Der Binder fügt ein übersetztes Quellprogramm mit allen zum Programmlauf erforderlichen Bindemodulen oder Bindelademodulen zu einer lauffähigen Einheit zusammen. Dabei modifiziert er jedes Modul um Adressen, die sich auf Bereiche außerhalb des betreffenden Moduls beziehen und deshalb vom Compiler noch nicht eingetragen werden konnten. Diese Adressen werden als Externbezüge oder Externe Referenzen bezeichnet.

In den nachfolgenden Abschnitten wird das Binden auf /390-Systemen beschrieben. Im Abschnitt "Eingabebeispiele " finden Sie entsprechende Eingabebeispiele.

> ¡ Bindelademodule (LLM) mit eingebundenem Laufzeitsystem **dürfen nicht** in Bibliotheken abgelegt werden, aus denen auch nicht vorgebundene Bindelademodule direkt geladen werden sollen.

5.1 Binden mit BINDER und DBL

Die Module des CRTE sollten grundsätzlich unter Verwendung des Autolink-Mechanismus, d.h. RESOLVE-BY-AUTOLINK-Anweisungen (BINDER) bzw. AUTOLNK-Angabe im BIND-Makro des DBL (dynamischer Bindelader) eingebunden werden, da die Namen der CRTE-Module keine garantierten externen Schnittstellen sind und sich in Folgeversionen ändern können.

BINDER und DBL gehören zum System **Binder-Lader-Starter (BLS)**.

5.1.1 Binder BINDER

Der Binder BINDER bindet Module zu einer ladbaren Einheit zusammen. Diese Einheit bezeichnet man als Bindelademodul (Link and Load Module, LLM). Ein Bindelademodul wird vom BINDER als Bibliothekselement vom Typ L in einer Bibliothek abgespeichert.

Aus folgenden Modulen kann der BINDER ein Bindelademodul binden:

- Bindemodule (OMs) und vorgebundene Bindemodule (Großmodule) aus einer Bibliothek oder aus der temporären EAM-Bindemoduldatei.
- Bindelademodule (LLMs), die vom Compiler oder BINDER erzeugt wurden und aus einer Bibliothek stammen.

5.1.2 Dynamischer Bindelader DBL

Der dynamische Bindelader DBL hat die Aufgabe, Module zu einer Ladeeinheit zu binden und diese zu laden. Der DBL bindet jedoch nur dann temporär, wenn noch kein ablauffähiges Programm vorliegt.

Der DBL kann aus folgenden Modulen eine Ladeeinheit binden:

- Bindelademodule (LLMs), die vom BINDER gebunden oder von Compilern erzeugt und in einer Bibliothek (Typ L) gespeichert wurden.
- Bindemodule (OMs), die von Compilern erzeugt und in einer Bibliothek (Typ R) oder in der temporären EAM-Bindemoduldatei gespeichert wurden.

5.1.3 Binden über den Autolink-Mechanismus

Beim Autolink-Mechanismus versuchen BINDER und DBL, alle nicht aufgelösten Externbezüge in den Modulen einer Ladeeinheit zu befriedigen.

Wichtig ist die Reihenfolge, in der Sie die zu durchsuchenden Bibliotheken angeben. Um sicherzustellen, dass die gesuchten Externbezüge aus den richtigen Modulen befriedigt werden, müssen Sie deshalb nachstehende **logische** Reihenfolge bei der Angabe der Bibliotheken einhalten:

1. benutzereigene Bibliotheken
2. die CRTE-Bibliotheken SYSLNK.CRTE (oder SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CRTE.COMPL) und SYSLNK.CRTE.CPP plus SYSLNK.CRTE.CFCPP
3. sprachspezifische Laufzeitbibliotheken von Nicht-CRTE-Sprachen (bei „Fremdsprachenmix“))

- Beachten Sie, dass der BINDER die Bibliotheken in genau der Reihenfolge durchsucht, in der sie angegeben werden.
- Der Cfront-Modus wird von SYSLNK.CRTE.COMPL nicht unterstützt.

Ausführliche Informationen zum Binden via Autolink-Mechanismus finden Sie in den Handbüchern „BINDER“ und „Bindelader-Starter“.

5.1.4 Binden bei geschachtelten Externbezügen

Die C++-Laufzeitbibliothek und die Fortran90-Laufzeitbibliothek enthalten Externbezüge auf die C-Laufzeitbibliothek. Dies bedeutet, dass nicht alle Externbezüge mit einmaligem chronologischem Durchsuchen der angegebenen Bibliotheken aufgelöst werden können, sondern bereits durchsuchte Bibliotheken nocheinmal durchsucht werden müssen.

> ¡ Beachten Sie, dass der BINDER den Suchvorgang nur dann automatisch wiederholt, wenn alle zu durchsuchenden Laufzeitbibliotheken als Liste in einer einzigen RESOLVE-BY-AUTOLINK-Anweisung angegeben werden.

5.1.5 Zeitfunktionen verwenden

Standardmäßig verwenden die Zeitfunktionen `ctime`, `difftime`, `ftime`, `gmtime`, `localtime`, `mktime` und `time` den 01.01.1970 00:00:00 als Stichtag (Epoche). Somit liefern die genannten Zeitfunktionen vom 13.12.1901 20:45:52 bis zum 19.01.2038 03:14:07 korrekte Ergebnisse. Dies entspricht dem Verhalten bei Einsatz des Bindeschalters TIMESHIFT in CRTE-Versionen seit V2.9A (siehe auch "Bibliothek SYSLNK.CRTE.TIMESHIFT“ im Abschnitt "Bibliotheken des C-Laufzeitsystems").

Falls Sie für die genannten Zeitfunktionen das standardmäßige Verhalten in CRTE-Versionen bis V2.9 bzw. V10.0A oder V11.0A wünschen, müssen Sie den Bindeschalter TIME50 verwenden (siehe auch "Bibliothek SYSLNK.CRTE.TIME50“ im Abschnitt "Bibliotheken des C-Laufzeitsystems").

5.1.6 Symbole maskieren

Beim Binden mit dem BINDER werden Symbole (CSECTs, ENTRYs) standardmäßig nicht maskiert und bleiben für spätere Bindeläufe mit dem BINDER oder DBL sichtbar.Beim Binden mit dem DBL hat dies folgende Auswirkungen: Wenn in einer Bibliothek sowohl vom Compiler generierte Einzelmodule als auch LLMs mit eingebundenem Laufzeitsystem stehen, werden beim dynamischen Binden der Einzelmodule u.U. die Externbezüge auf das Laufzeitsystem aus irgendeinem vorgebundenen Modul befriedigt und nicht aus der Laufzeitbibliothek. In diesem Fall erhält man diverse „DUPLICATES“-Warnungen vom DBL. Auf Grund des Autolink-Mechanismus wird nämlich zuerst die Bibliothek durchsucht, in der das Einzelmodul steht und erst anschließend die mit den Linknamen BLSLIB*nn* zugewiesenen Laufzeitbibliotheken.

Um sicherzustellen, dass beim dynamischen Binden die Externbezüge immer aus der aktuellen Laufzeitbibliothek und nicht aus einem beliebigen Modul befriedigt werden, empfiehlt es sich deshalb,

- entweder Einzelmodule und vorgebundene Module in unterschiedlichen Bibliotheken zu halten oder
- beim Binden mit dem BINDER die Symbole z.B. mit der Anweisung MODIFY-SYMBOL-VISIBILITY VISIBLE=NO zu maskieren.

5.1.7 Sprachspezifische Besonderheiten

Besonderheiten in C und C++

Module, die auf eine der unten aufgeführten Arten erzeugt werden, liegen ausschließlich im LLM-Format vor. Programmsysteme, die solche Programmteile enthalten, können mit dem DBL nur im RUN-MODE=ADVANCED oder mit dem BINDER gebunden werden:

- C-Module, erzeugt mit dem C/C++-Compiler ab V3.0
- C++-Module

Für C++-Module, die mit einem C/C++-Compiler ab V3.0 im ANSI-C++-Modus erzeugt werden, gilt die weitere Besonderheit, dass sie (u.a. wegen der Template-Instanziierung) mit einer eigens hierfür bereitgestellten Compiler-Anweisung (BIND) gebunden werden müssen. Auf das Binden von ANSI-C++-Modulen wird im Rahmen dieses Benutzerhandbuchs nicht näher eingegangen. Einzelheiten finden Sie im „C/C++ -Benutzerhandbuch“ [9].

Besonderheiten in COBOL

Siehe dazu das Kapitel **„Binden, Laden, Starten“** im „COBOL85- bzw. COBOL2000 Benutzerhandbuch“ [2, 4].

5.2 Bindetechniken

Es sind drei grundlegende Bindetechniken zu unterscheiden:

- statisches Binden
- dynamisches Nachladen des C-/COBOL-Laufzeitsystems und der internen Routinen (Bindetechnik Partial-Bind)
- dynamisches Binden mit dem DBL

Eingabebeispiele zu den hier vorgestellten Bindetechniken finden Sie im Abschnitt "Eingabebeispiele".

5.2.1 Statisches Binden mit dem BINDER

Mit dem BINDER können Sie Bindemodule und Bindelademodule (LLM) zu einem Bindelademodul (LLM) binden und als Element vom Typ „L“ in einer Bibliothek abspeichern.

Bei der Technik des statischen Bindens mit CRTE bleiben keine Referenzen offen. Beim Binden verwenden Sie die Bibliothek SYSLNK.CRTE.

5.2.2 Dynamisches Nachladen des C-/COBOL-Laufzeitsystems und der internen Routinen (Bindetechnik Partial-Bind)

Die Bindetechnik Partial-Bind bietet die Möglichkeit, beim Binden mit dem BINDER an Stelle des kompletten C-, C++- oder COBOL-Laufzeitsystems und der internen Routinen nur Verbindungsmodule einzubinden, die alle offenen Externbezüge auf die vorladbaren Bestandteile des CRTE befriedigen. Die benötigten Module selbst werden dann, sofern sie nicht vorgeladen sind, erst zum Ablaufzeitpunkt dynamisch nachgeladen.

Es gibt zwei Varianten der Bindetechnik Partial-Bind:

- Standard Partial-Bind
- Complete Partial-Bind

Wegen der erheblichen Einsparung von Plattenspeicherplatz empfiehlt es sich, für das Binden folgender Programme generell die Bindetechnik Partial-Bind zu verwenden:

- C-Programme
- COBOL-Programme
- Programme in anderen Sprachen, die C- oder COBOL-Programmteile enthalten bzw. die gemeinsamen internen Routinen verwenden.

In jedem Fall muss das CRTE im Ablaufsystem installiert sein.

i
- Bei Programmaufruf muss das passende CRTE verfügbar sein.
- Wenn Sie unter POSIX Shared Libraries einsetzen, ist erfolgreiches Binden nur mit Complete Partial-Bind gewährleistet.

Standard Partial-Bind

Beim Binden mit Standard Partial-Bind verwenden Sie anstelle der Bibliothek SYSLNK.CR-TE die Bibliothek SYSLNK.CRTE.PARTIAL-BIND (siehe Abschnitt "Bibliotheken für die Bindetechnik Partial-Bind auf /390-Systemen“).

Beim Standard Partial-Bind befriedigen die in SYSLNK.CRTE.PARTIAL-BIND bereitgestellten, nachladenden Verbindungsmodule alle offenen Externbezüge des zu bindenden Bindemoduls. Andere Externbezüge werden nicht berücksichtigt.

i Benötigt ein nachgeladener Modul Entries des Laufzeitsystems, die vom nachladenden Modul nicht benötigt werden und somit nicht eingebunden worden sind, so kommt es zu unbefriedigten Externbezügen. Verwenden Sie in solchen Fällen bitte die Bindetechnik Complete Partial-Bind (siehe "Complete Partial-Bind“).

Für das Binden von C++-Programmen werden je nach Modus (C-Front oder ANSI) weitere Bibliotheken benötigt (siehe Abschnitt "Cfront-C++-Bibliotheksfunktionen und -Laufzeitsystem“ und Abschnitt "ANSI-C++-Bibliotheken und -Laufzeitsysteme“).

Complete Partial-Bind

Beim Binden mit Complete Partial-Bind verwenden Sie anstelle der Bibliothek SYSLNK.CRTE die Bibliothek SYSLNK.CRTE.COMPL (siehe Abschnitt "Bibliotheken für die Bindetechnik Partial-Bind auf /390-Systemen“).

Beim Complete Partial-Bind enthalten die in SYSLNK.CRTE.COMPL bereitgestellten, nachladenden Verbindungsmodule alle Entries und externen Daten des kompletten C- oder COBOL-Laufzeitsystems. Damit sind unbefriedigte Externbezüge, wie sie beim Standard Partial-Bind auftreten können, beim Complete Partial-Bind ausgeschlossen.

Complete Partial-Bind bei C-, COBOL- und gemischten C-/COBOL-Programmen

Die Laufzeitsysteme von C und COBOL werden jeweils entweder komplett oder gar nicht eingebunden. Dies bedeutet:

- Wenn Ihr Anwenderprogramm mindestens einen C-Entry benötigt, werden beim Complete Partial-Bind alle Externbezüge auf das gesamte C-Laufzeitsystem befriedigt.
- Wenn Ihr COBOL-Programm mindestens einen COBOL-Entry benötigt, werden beim Complete Partial-Bind alle Externbezüge auf das gesamte COBOL-Laufzeitsystem befriedigt.
- Bei reinen C-Programmen werden keine COBOL-Entries eingebunden.
- Da alle Entries und alle externen Variablen eingebunden werden, können diese mit gleichnamigen Entries oder externen Variablen im Anwenderprogramm in Konflikt kommen (duplicate entries). Auch Redefinitionen von Funktionen des Laufzeitsystems durch gleichnamige Funktionen im Anwenderprogramm sind nicht mehr möglich. Die vom Laufzeitsystem belegten Namen können mit der LMS-Anweisung

  ```
  //SHOW-ELEMENT (SYSLNK.CRTE.COMPL,*,L),LLM-INF=PAR(INF=ESVD)
  ```

  angezeigt werden.

> ¡ Bei COBOL- und gemischten C-/COBOL-Programmen darf die Bindetechnik Complete Partial-Bind nur eingesetzt werden, wenn die Objekte unter POSIX ablaufen und Shared Objects geladen werden.

Complete Partial-Bind bei C++-Programmen

Für das Binden von C++-Programmen (ANSI) wird zusätzlich die Bibliothek SYSLNK.CRTE.CPP-COMPL benötigt.

> ¡ Für C++-Programme im Cfront-Modus wird die Bindetechnik Complete Partial-Bind nicht unterstützt.

Besonderheiten beim Complete Partial-Bind

Beim Binden mit SYSLNK.CRTE.COMPL ist zu beachten:

- Die Funktion *_edt* wird nicht unterstützt.
- Die POSIX- und TIME-Bindeschalter sind nicht enthalten. Beim cc-Kommando unter POSIX werden sie automatisch eingebunden.
- Bei der Erweiterung der C/C++-Laufzeitsysteme um neue Funktionen in den COMPL-Bibliotheken ist keine Aufwärtskompatibilität garantiert. Wenn die Anwendung mit der Bindetechnik Complete Partial-Bind gebunden wird, kann jeder neue Entry potentiell mit einem bereits in der Anwendung vorhandenen gleichnamigen Entry in Konflikt geraten.

Verwendete Subsysteme

Für beide Partial-Bind-Varianten werden folgende Subsysteme verwendet:

- CRTEPART (für C)
- COBPART (für COBOL)
- CRTESIS (für gemeinsame Routinen)

> ¡ Aus Gründen der Performance und um die Vorteile der Bindetechnik Partial-Bind in vollem Umfang nutzen zu können, sollten die genannten Subsysteme vorgeladen sein.

5.2.3 Dynamisches Binden mit dem DBL

Mit dem Dynamischen Bindelader können Sie in einem Arbeitsgang folgende Schritte durchführen:

1. Module temporär zu einer ladbaren Einheit binden
2. Ladbare Einheit in den Arbeitsspeicher laden und starten

Am Ende des Programmablaufs wird die erzeugte Ladeeinheit gelöscht.

Sofern es sich bei dem angegebenen Startmodul um ein vorgebundenes Modul handelt, muss dieses Modul mit offenen Externbezügen auf das CRTE gebunden werden, d.h. ohne RESOLVE-BY-AUTOLINK-Anweisung auf das CRTE.
Mit dem BINDER ist unvollständiges Binden ohne besondere Vorkehrungen möglich.

Für den Fall, dass das C- und/oder COBOL-Subsystem in den oberen Adressraum des Klasse-4-Speichers geladen ist (siehe auch Abschnitt "Gemeinsam benutzbares CRTE“), ist im START-PROGRAM-Kommando die Angabe PROG-MODE=ANY erforderlich.

> ℹ Die CRTE-Bibliotheken für die Bindetechnik Partial-Bind dürfen **nur zum Binden**, **nicht** aber als BLSLIB **für das Laden** von Programmen mit offenen Externverweisen beim dynamischen Binden mit DBL verwendet werden, insbesondere dann nicht, wenn das Subsystem CRTEC vorgeladen ist. Als BLSLIB ist in diesem Fall ausschließlich die Bibliothek SYSLNK.CRTE zu verwenden.

Verwendete Subsysteme

Bei der hier beschriebenen Technik des dynamischen Bindens mit DBL werden folgende Subsysteme benötigt:

- CRTEC (für C)
- CRTECOB (für COBOL)

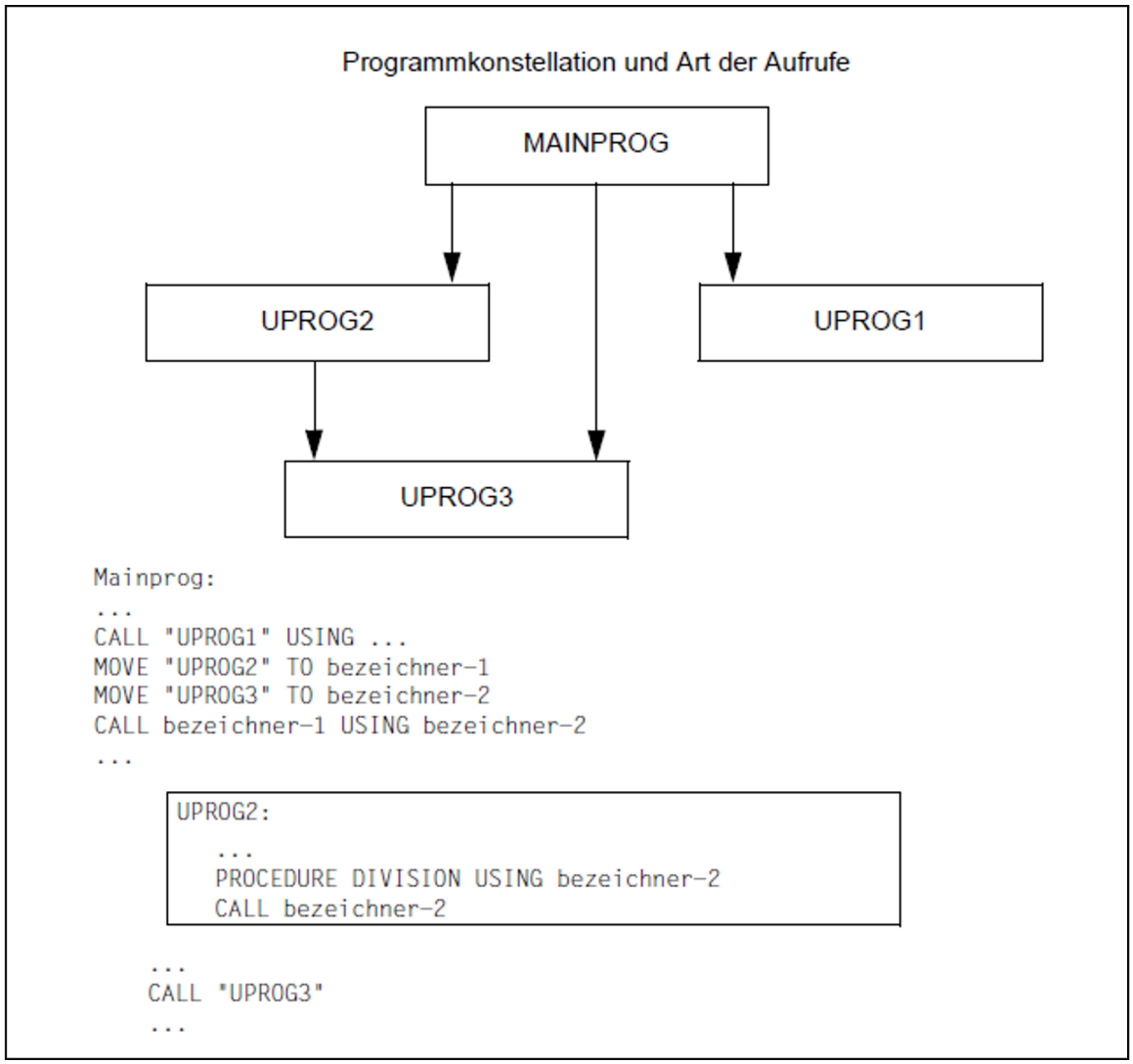
5.2.4 Hinweise zum Binden von Großmodulen

Laufzeitsystem- und ILCS-Module dürfen beim Binden von Großmodulen nicht mit eingebunden werden, wenn diese Großmodule zu einem späteren Zeitpunkt mit anderen Objekten oder Großmodulen zusammengebunden werden sollen. Bestimmte Laufzeitsystem Module wie z.B. IT0PCD oder ITCMPOVH dürfen in einer Anwendung nur einmal vorhanden sein, da anderenfalls der Datenaustausch zwischen den Laufzeitsystem-Modulen nicht korrekt funktioniert. Ausgenommen hiervon sind abgeschottete Subsysteme, auf die im Abschnitt "Abgeschottete Subsysteme“ näher eingegangen wird.

Auch in Assembler- oder COBOL-Anwendungen, die zur Laufzeit Unterprogramme oder Großmodule nachladen, dürfen Laufzeitsystem- und ILCS-Module nicht fest mit eingebunden sein, da anderenfalls die Laufzeitsystem-Module vom nachgeladenen Unterprogramm nicht gefunden werden und dadurch ein zweites Laufzeitsystem nachziehen. Auch hier gilt wieder die Ausnahme: Werden nur abgeschottete Subsysteme nachgeladen, dann kann die Anwendung komplett gebunden werden.

Beispiel

Das folgende Beispiel zeigt Binde- und Ladetechniken für Programmsysteme mit dynamisch nachzuladenden Unterprogrammen.



UPROG1 wird ausschließlich in der Form `CALL` *`literal`* aufgerufen.
UPROG2 wird ausschließlich in der Form `CALL` *`bezeichner`* aufgerufen.
UPROG3 wird auf beide Arten aufgerufen.

Das bedeutet: Für UPROG1 und UPROG3 werden Externverweise abgesetzt, UPROG2 wird dynamisch nachgeladen.

Für diese Programmkonstellation werden im Folgenden die Möglichkeiten gezeigt, das Programm zum Ablauf zu bringen.

Die einzelnen Programme sind als Objektmodule unter den Elementnamen MAINPROG, UPROG1, UPROG2 und UPROG3 in der Bibliothek BENUTZER-PROGRAMME abgelegt

Verwendung des BINDER (LLM-Binden)

Der BINDER lässt standardmäßig alle Externverweise und Einsprungpunkte sichtbar. Dies ist für den anschließenden Bindelader-Lauf unbedingt erforderlich. Ferner können bei Verwendung des BINDER die Externverweise offen bleiben. Deshalb braucht das LZS nicht eingebunden zu werden. Dies ist von Vorteil, wenn für den Programmablauf ein gemeinsam benutzbares LZS verwendet werden soll.

Das nachfolgende Beispiel zeigt das Erzeugen eines einzigen Bindelademoduls.

```
/START-PROGRAM $BINDER
 START-LLM-CREA GROSSMOD .................................(1)
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=MAINPROG ........(2)
 INCLUDE-MODULES LIB=BENUTZER-PROGRAMME,ELEM=UPROG2 ..........(3)
 RESOLVE-BY-AUTOLINK LIB=BENUTZER-PROGRAMME ..................(4)
 SAVE-LLM LIB=MODUL.LIB ..................................(5)
 END
/ADD-FILE-LINK BLSLIB00,$.SYSLNK.CRTE ..................... (6)
/START-PROGRAM *MODULE(LIB=MODUL.LIB,ELEM=GROSSMOD,- .........(7)
 RUN-MODE=ADVANCED(ALT-LIB=YES,UNRES-EXT=DELAY,-
 LOAD-INFO=REFERENCE))
```

(1)	Erzeugen eines Bindelademoduls mit dem Namen GROSSMOD.
(2)	Explizites Einbinden des Hauptprogramm-Moduls MAINPROG aus der Bibliothek BENUTZER-PROGRAMME.
(3)	Explizites Einbinden des Moduls UPROG2 aus der Bibliothek BENUTZER-PROGRAMME, um dynamisches Nachladen zu vermeiden; damit erübrigt sich beim anschließenden Bindeladevorgang die Zuweisung der Bibliothek BENUTZER-PROGRAMME mit dem Linknamen COBOBJCT.
(4)	Einbinden aller weiteren erforderlichen Module (UPROG1, UPROG3) aus der Bibliothek BENUTZER-PROGRAMME.
(5)	Abspeichern des erzeugten Bindelademoduls in der Programmbibliothek MODUL.LIB als Element vom Typ L.
(6)	Zuweisen der Laufzeitbibliothek.
(7)	Aufruf des Bindelademoduls GROSSMOD.

5.2.5 Gegenüberstellung und Bewertung der Bindetechniken

Nachfolgend sind die Vor- und Nachteile der einzelnen Bindetechniken dargestellt.

Vor- und Nachteile des statischen Bindens

Vorteile

- Für die Anwendung ist ohne Bedeutung, ob beim Anwender ein CRTE installiert ist und ggf. welche CRTE-Version beim Anwender installiert ist. Die Anwendung ist somit leichter auf fremde Systeme portierbar.
- Zur Laufzeit sind keine Bindevorgänge mehr erforderlich.
- Es ist kein Linkname erforderlich.

Nachteile

- Höherer Verbrauch von Plattenspeicher; es können bis zu 800 KB Code durch das Laufzeitsystem hinzukommen.
- Längere Bindezeiten
- Längere Ladezeiten, da stets die komplette Anwendung geladen werden muss
- Korrekturen im CRTE wirken sich nicht auf eine bereits gebundene Anwendung aus. Korrekturen im verwendeten CRTE müssen ggf. durch Objektkorrekturen in der Anwendung behoben werden.

Vor- und Nachteile der Bindetechnik Partial-Bind

Vorteile

- Kürzere Produktionszeit auf Grund schnelleren Bindens mit den Partial-Bind-Bibliotheken..
- Benötigter Plattenspeicher ist geringer, da kein Laufzeitsystem angebunden wird.
- Gute Lade-Performance, sofern die entsprechenden Subsysteme geladen sind.
- Korrekturen im CRTE wirken im Allgemeinen automatisch, auch auf bereits gebundene Anwendungen.

Nachteil

CRTE muss im Ablaufsystem installiert sein. (Dies kann jedoch auch als Vorteil angesehen werden.)

Vor- und Nachteile des dynamischen Bindens mit DBL

Vorteile

- Bindelauf bei der Produktion wird eingespart.
- Benötigter Plattenspeicher ist geringer, da kein Laufzeitsystem eingebunden wird.
- Korrekturen im CRTE wirken im Allgemeinen automatisch, auch auf bereits gebundene Anwendungen.

Nachteile

- CRTE muss im Ablaufsystem installiert sein. (Dies kann jedoch auch als Vorteil angesehen werden.)
- Es muss spezifiziert werden, von wo die offenen Referenzen befriedigt werden sollen.
- Bei jedem Laden muss der Bindevorgang nachgeholt werden.

5.3 Binden bei Sprachen-Mix

Beim Sprachen-Mix sind zu unterscheiden:

- Sprachen-Mix zwischen CRTE-Sprachen
- Sprachen-Mix zwischen „Fremdsprachen“

5.3.1 Sprachen-Mix zwischen CRTE-Sprachen

Sind bei einem Sprachmix ausschließlich CRTE-Sprachen beteiligt, gelten bzgl. des CRTE je nach beteiligter Sprache und Modulformat die gleichen Bindetechniken wie beim Binden von Programmen in einer Sprache:

Sind nur C und COBOL beteiligt, müssen die Laufzeitmodule aus den Bibliotheken SYSLNK.CRTE oder SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CRTE.COMPL gebunden werden. Wenn COBOL beteiligt ist, darf die Bibliothek SYSLNK.CRTE.COMPL jedoch nur verwendet werden, wenn die Objekte unter POSIX ablaufen und Shared Objects geladen werden.

Wenn CFront-C++ beteiligt ist, darf SYSLNK.CRTE.COMPL nicht verwendet werden. Ferner sind die Laufzeitmodule aus den Bibliotheken SYSLNK.CRTE.CPP und SYSLNK.CRTE.CFCPP zu binden.

> ¡ Für Cfront-C++ wird die Bindetechnik Complete Partial-Bind nicht unterstützt.

5.3.2 Sprachen-Mix zwischen „Fremdsprachen“

Unter „Fremdsprachen“ sind alle ILCS-fähigen Sprachen zu verstehen, die nicht zu den CRTE-Sprachen zählen (Assembler, Fortran, Pascal, usw.). Mit den Compilern für diese Sprachen werden sprachspezifische Laufzeitsysteme ausgeliefert, die zusätzlich zum CRTE beim Binden berücksichtigt werden müssen.

Dabei müssen die CRTE-Bibliotheken SYSLNK.CRTE oder SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CRTE.COMPL (siehe Abschnitt "Dynamisches Nachladen des C-/COBOL-Laufzeitsystems und der internen Routinen (Bindetechnik Partial-Bind)“) immer vor den jeweils sprachspezifischen Laufzeitbibliotheken gebunden werden.

> ¡ Für das Binden mit SYSLNK.CRTE.COMPL beachten Sie bitte „Besonderheiten beim Complete Partial-Bind“ im Abschnitt "Dynamisches Nachladen des C-/COBOL-Laufzeitsystems und der internen Routinen (Bindetechnik Partial-Bind)".

5.4 Eingabebeispiele

Im Folgenden wird anhand von Eingabebeispielen das statische und dynamische Binden des CRTE mit dem BINDER und DBL dargestellt, jeweils unterschieden nach Programmkonstellation:

- Im Abschnitt "C-, C++- oder COBOL-Programm statisch binden" finden Sie Beispiele zum Binden von Programmen, die das CRTE aus der Bibliothek SYSLNK.CRTE einbinden.
- Im Abschnitt "Programme binden, die das C-oder COBOL-Laufzeitsystem dynamisch nachladen (Bindetechnik Partial-Bind)“ finden Sie Beispiele zum Binden von Programmen, die das CRTE aus der Bibliothek SYSLNK.CRTE.PARTIAL-BIND bzw. SYSLNK.CRTE.COMPL einbinden.
- Im Abschnitt "Dynamisch binden mit dem DBL“ finden Sie Beispiele zum dynamischen Binden von Programmen mit dem DBL, die das CRTE aus der Bibliothek SYSLNK.CRTE einbinden.
- Im Abschnitt "Binden bei Sprachen-Mix zwischen „Fremdsprachen““ finden Sie Beispiele zum Binden bei Sprachen-Mix.

> ¡ Bitte beachten Sie beim Binden von C- und C++-Programmen die Hinweise im Abschnitt "Sprachspezifische Besonderheiten“.

5.4.1 C-, C++- oder COBOL-Programm statisch binden

Statisch binden mit BINDER

1. C- oder COBOL-Programm (Bindemodul oder Bindelademodul)

```
/START-BINDER
//START-LLM-CREATION INT-NAME=interner-name ----------------------- (1)
//INCLUDE-MODULES LIB=..., ELEM=... ------------------------------- (2)
//RESOLVE-BY-AUTOLINK LIB=benutzerbibliothek ---------------------- (3)
//RESOLVE-BY-AUTOLINK LIB=$.SYSLNK.CRTE --------------------------- (4)
//SAVE-LLM LIB=..., ELEM=... -------------------------------------- (5)
//END
```

   (1) Ein Bindelademodul mit dem Namen `interner-name` wird erzeugt.

   (2) Die Module des Programms werden explizit eingefügt.

   (3) Die Benutzerbibliothek `benutzerbibliothek` wird mit der Autolink-Funktion statisch eingebunden.

   (4) Die Bibliothek $.SYSLNK.CRTE, die u.a. das vollständige C- und COBOL-Laufzeitsystem enthält, wird mit der Autolink-Funktion statisch eingebunden.

   (5) Das generierte aktuelle Bindelademodul wird gespeichert.

2. C++-Programm, Cfront-Modus (Bindelademodul) für Compiler-Versionen <= V2.2

```
/START-BINDER
//START-LLM-CREATION INT-NAME=interner-name ----------------------- (1)
//INCLUDE-MODULES LIB=..., ELEM=... ------------------------------- (2)
//RESOLVE-BY-AUTOLINK LIB=benutzerbibliothek ---------------------- (3)
//RESOLVE-BY-AUTOLINK LIB=($.SYSLNK.CRTE,$.SYSLNK.CRTE.CPP) -------- (4)
//SAVE-LLM LIB=..., ELEM=... -------------------------------------- (5)
//END
```

   (1) Ein Bindelademodul mit dem Namen `interner-name` wird erzeugt.

   (2) Die Module des Programms werden explizit eingefügt.

   (3) Die Benutzerbibliothek `benutzerbibliothek` wird mit der Autolink-Funktion statisch eingebunden.

   (4) Mit der Autolink-Funktion statisch eingebunden werden außerdem:

   - Die Bibliothek $.SYSLNK.CRTE, die u.a. das vollständige C- und COBOL-Laufzeitsystem enthält (siehe "Liefereinheit CRTE V10.1A").
   - Die Bibliothek $.SYSLNK.CRTE.CPP, die die Cfront-C++-Bibliotheksfunktionen enthält (siehe Abschnitt "Cfront-C++-Bibliotheksfunktionen und -Laufzeitsystem“).

   (5) Das aktuelle Bindelademodul wird gespeichert.

3. Cfront-C++-Programm (Bindelademodul) für Compiler-Versionen ab V3.0

```
/START-CPLUS-COMPILER ---------------------------------------------- (1)
//MODIFY-BIND-PROPERTIES INCLUDE=*LIB(LIB=...,ELEM=...)
//MODIFY-BIND-PROPERTIES RESOLVE=*AUTOLINK(benutzerbibliothek)
//MODIFY-BIND-PROPERTIES RUNTIME-LANG=CPLUSPLUS(CPP),STDLIB=STATIC
//BIND OUTPUT=*LIB(LIB=...,ELEM=...)
```

(1) Der Binder wird implizit durch den C++-Compiler aufgerufen.
Die nachfolgenden Anweisungen für den C++-Compiler haben die gleiche Funktionalität wie die BINDER-Anweisungen in Beispiel 2.

5.4.2 Programme binden, die das C-oder COBOL-Laufzeitsystem dynamisch nachladen (Bindetechnik Partial-Bind)

Statisch binden mit dem BINDER

1. C- oder COBOL-Programm (Bindemodul oder Bindelademodul)

```
/START-BINDER
//START-LLM-CREATION INT-NAME=interner-name ----------------------- (1)
//INCLUDE-MODULES LIB=..., ELEM=... ------------------------------- (2)
//RESOLVE-BY-AUTOLINK LIB=benutzerbibliothek ---------------------- (3)
//RESOLVE-BY-AUTOLINK LIB=$.SYSLNK.CRTE.PARTIAL-BIND -------------- (4)
//SAVE-LLM LIB=..., ELEM=... -------------------------------------- (5)
//END
```

   (1) Ein Bindelademodul mit dem Namen `interner-name` wird erzeugt.

   (2) Die Module des Programms werden explizit eingefügt.

   (3) Die Benutzerbibliothek wird mit der Autolink-Funktion statisch eingebunden.

   (4) Die Bibliothek $.SYSLNK.CRTE.PARTIAL-BIND (Standard Partial-Bind), die u.a. das Verbindungsmodul auf das dynamisch nachladbare C- und COBOL-Laufzeitsystem enthält (siehe "Liefereinheit CRTE V10.1A"), wird mit der Autolink-Funktion statisch eingebunden.

   Alternativ können Sie mit der Bindetechnik Complete Partial-Bind binden, indem Sie anstatt der Bibliothek $.SYSLNK.CRTE.PARTIAL-BIND die Bibliothek $.SYSLNK.CRTE.COMPL verwenden.

   (5) Das aktuelle Bindelademodul wird gespeichert.

2. C++-Programm (Bindelademodul) für Compiler-Versionen <= V2.2

```
/START-BINDER
//START-LLM-CREATION INT-NAME=interner-name -------------------------- (1)
//INCLUDE-MODULES LIB=..., ELEM=... ---------------------------------- (2)
//RESOLVE-BY-AUTOLINK LIB=benutzerbibliothek ------------------------- (3)
//RESOLVE-BY-AUTOLINK LIB=($.SYSLNK.CRTE.PARTIAL-BIND,$.SYSLNK.CRTE.CPP)(4)
//SAVE-LLM LIB=..., ELEM=... ----------------------------------------- (5)
//END
```

   (1) Ein Bindelademodul mit dem Namen `interner-name` wird erzeugt.

   (2) Die Module des Programms werden explizit eingefügt.

   (3) Die Benutzerbibliothek wird mit der Autolink-Funktion statisch eingebunden.

   (4) Mit der Autolink-Funktion statisch eingebunden werden außerdem:

   - Die Bibliothek $.SYSLNK.CRTE.PARTIAL-BIND, die u.a. das vollständige C- und COBOL-Laufzeitsystem enthält (siehe "Liefereinheit CRTE V10.1A").
   - Die Bibliothek $.SYSLNK.CRTE.CPP, die die Cfront-C++-Bibliotheksfunktionen enthält (siehe Abschnitt "Cfront-C++-Bibliotheksfunktionen und -Laufzeitsystem“).

(5) Das aktuelle Bindelademodul wird gespeichert.

3. C++-Programm (Bindelademodul) für Compiler-Versionen ab V3.0

```
/START-CPLUS-COMPILER ---------------------------------------------- (1)
//MODIFY-BIND-PROPERTIES INCLUDE=*LIB(LIB=...,ELEM=...)
//MODIFY-BIND-PROPERTIES RESOLVE=*AUTOLINK(benutzerbibliothek)
//MODIFY-BIND-PROPERTIES RUNTIME-LANG=CPLUSPLUS(CPP), STDLIB=DYNAMIC
//BIND OUTPUT=*LIB(LIB=...,ELEM=...)
```

(1) Der Binder wird implizit durch den C++-Compiler aufgerufen.
Die nachfolgenden Anweisungen für den C++-Compiler haben die gleiche Funktionalität wie die BINDER-Anweisungen in Beispiel 2.

Laden und Starten von fertig gebundenen Bindelademodulen

```
/START-PROGRAM *MODULE(LIB=...,ELEM=...,RUN-MODE=ADVANCED)
```

5.4.3 Dynamisch binden mit dem DBL

1. C- oder COBOL-Programm (Bindemodul)

```
/SET-TASKLIB $.SYSLNK.CRTE ---------------------------------------- (1)
...
/START-PROGRAM *MODULE(LIB=..., ELEM=..., PROG-MODE=ANY) ------------ (2)
```

   (1) Die Bibliothek $.SYSLNK.CRTE wird als Tasklib zugewiesen.

   (2) Sofern es sich bei dem angegebenen Startmodul um ein vorgebundenes Modul handelt, muss dieses Modul mit offenen Externbezügen auf das CRTE gebunden werden, d.h. ohne RESOLVE-Anweisung auf das CRTE (siehe Abschnitt "Dynamischer Bindelader DBL“).

2. C- oder COBOL-Programm (Bindemodul oder Bindelademodul)

```
/ADD-FILE-LINK LINK-NAME=BLSLIB00,FILE-NAME=benutzerbibliothek ----- (1)
/ADD-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=$.SYSLNK.CRTE
...
/START-PROGRAM *MODULE(LIB=..., ELEM=..., PROG-MODE=ANY, ----------- (2)
/RUN-MODE=ADVANCED(ALT-LIB=YES,AUTO=ALT-LIB))
```

   (1) Den Bibliotheken `benutzerbibliothek` und $.SYSLNK.CRTE werden die Linknamen BLSLIB00 bzw. BLSLIB01 zugewiesen. Dies bewirkt, dass zur Befriedigung unbefriedigter Externbezüge zuerst `benutzerbibliothek` und anschließend SYSLNK.CRTE durchsucht wird.
   Die Linknamen müssen **vor** dem Binderlauf zugewiesen werden und werden nach dem Binderlauf nicht freigegeben.

   (2) Sofern es sich bei dem angegebenen Startmodul um ein vorgebundenes Modul handelt, muss dieses Modul mit offenen Externbezügen auf das CRTE gebunden werden, d.h. ohne RESOLVE-Anweisung auf das CRTE (siehe Abschnitt "Dynamischer Bindelader DBL“).

3. C++-Programm (Bindelademodul) für Compiler-Versionen <= V2.2

```
/ADD-FILE-LINK LINK-NAME=BLSLIB00,FILE-NAME=benutzerbibliothek
/ADD-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=$.SYSLNK.CRTE
/ADD-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=$.SYSLNK.CRTE.CPP ------ (1)
...
/START-PROGRAM *MODULE(LIB=..., ELEM=..., PROG-MODE=ANY,-  --------- (2)
/RUN-MODE=ADVANCED(ALT-LIB=YES,AUTO=ALT-LIB))
```

   (1) Aufbauend auf Beispiel 2 wird zusätzlich der Cfront-C++-Bibliothek $.SYSLNK.CRTE.CPP (siehe Abschnitt "Cfront-C++-Bibliotheksfunktionen und Laufzeitsystem“) der Linkname BLSLIB02 zugewiesen. Damit kann das Programm die C++-Bibliotheksfunktionen für komplexe Mathematik und Standard-E/A verwenden. Beachten Sie bitte, dass bei C++-Programmen, die eine C++-Version <= V2.2 verwenden, die Linknamen in der angegebenen Reihenfolge zugewiesen werden müssen.
   Zur Befriedigung unbefriedigter Externbezüge werden die einzelnen Bibliotheken in der Reihenfolge ihrer Linknamen
   BLSLIB00 -> BLSLIB01 -> BLSLIB02 durchsucht.

(2) Sofern es sich bei dem angegebenen Startmodul um ein vorgebundenes Modul handelt, muss dieses Modul mit offenen Externbezügen auf das CRTE gebunden werden, d.h. ohne RESOLVE-Anweisung auf das CRTE (siehe Abschnitt "Dynamischer Bindelader DBL“).

4. C++-Programm, Cfront-Modus (Bindelademodul) für Compiler-Versionen ab V3.0.

```
/ADD-FILE-LINK LINK-NAME=BLSLIB00,FILE-NAME=benutzerbibliothek
/ADD-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=$.SYSLNK.CRTE.RTSCPP
/ADD-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=$.SYSLNK.CRTE.CFCPP ---- (1)
/ADD-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=$.SYSLNK.CRTE
...
/START-PROGRAM *MODULE(LIB=..., ELEM=..., PROG-MODE=ANY,------------ (2)
 RUN-MODE=ADVANCED(ALT-LIB=YES,AUTO=ALT-LIB))
```

(1) Aufbauend auf Beispiel 3 wird der Bibliothek SYSLNK.CRTE.CFCPP (Cfront-Laufzeitsystem, siehe Abschnitt "Cfront-C++-Bibliotheksfunktionen und -Laufzeitsystem“) ein Linkname zugewiesen. Für C++-Programme ab der C++-Version V3.0 ist die Reihenfolge, in der die Linknamen BLSLIBnn zugewiesen werden, nicht mehr zwingend vorgeschrieben.
Zur Befriedigung unbefriedigter Externbezügen werden die einzelnen Bibliotheken in der Reihenfolge ihrer Linknamen
BLSLIB00 -> BLSLIB01 -> BLSLIB02 -> BLSLIB03 durchsucht.

(2) Sofern es sich bei dem angegebenen Startmodul um ein vorgebundenes Modul handelt, muss dieses Modul mit offenen Externbezügen auf das CRTE gebunden werden, d.h. ohne RESOLVE-Anweisung auf das CRTE (sieh Abschnitt "Dynamischer Bindelader DBL“).

5.4.4 Binden bei Sprachen-Mix zwischen „Fremdsprachen“

Statisch binden mit dem BINDER

```
/START-BINDER
//START-LLM-CREATION INT-NAME=... ------------------------------------ (1)
//INCLUDE-MODULES LIB=..., ELEM=... ---------------------------------- (2)
//RESOLVE-BY-AUTOLINK LIB=benutzerbibliothek ------------------------- (3)
//RESOLVE-BY-AUTOLINK LIB=($.SYSLNK.CRTE,$.lzs-fremdsprache1,//
/                         lzs-fremdsprache2) ------------------------- (4)
// ...
//SAVE-LLM LIB=..., ELEM=...
//END
```

(1) Namen von Lademodul und Ausgabedatei werden spezifiert.

(2) Die Module des Programms werden explizit eingefügt.

(3) Die Benutzerbibliothek wird mit der Autolink-Funktion statisch eingebunden.

(4) Die Bibliotheken $.SYSLNK.CRTE sowie die Laufzeitsysteme lzs-fremdsprache1 und lzs-fremdsprache2 werden mit der Autolink-Funktion statisch eingebunden.

Dynamisch binden mit dem DBL

```
/ADD-FILE-LINK LINK-NAME=BLSLIB00,FILE-NAME=benutzerbibliothek --------- (1)
/ADD-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=$.SYSLNK.CRTE -------------- (1)
/ADD-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lzs-fremdsprache1 ---------- (1)
/ADD-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=lzs-fremdsprache2 ---------- (1)
...
/START-PROGRAM *MODULE(LIB=..., ELEM=..., PROG-MODE=ANY,-
/RUN-MODE=ADVANCED(ALT-LIB=YES,AUTO=ALT-LIB)) -------------------------- (2)
```

(1) Den Bibliotheken benutzerbibliothek, $.SYSLNK.CRTE, lzs-fremdsprache1 und lzs-fremdsprache2 werden der Reihe nach die Linknamen BLSLIB00 bis BLSLIB03 zugewiesen.
Zur Befriedigung unbefriedigter Externbezüge werden die einzelnen Bibliotheken in der Reihenfolge ihrer Linknamen
BLSLIB00 -> BLSLIB01 -> BLSLIB02 -> BLSLIB03 durchsucht.

(2) Sofern es sich bei dem angegebenen Startmodul um ein vorgebundenes Modul handelt, muss dieses Modul mit offenen Externbezügen auf das CRTE gebunden werden, d.h. ohne RESOLVE-Anweisung auf das CRTE (siehe Abschnitt "Dynamischer Bindelader DBL“).

6 Anhang: Anwendung der ILCS-Routinen IT0INITS und IT0ININ

Dieser Abschnitt enthält ein Beispiel zu folgenden Problemstellungen:

- Aufruf eines ILCS-Assemblerprogramms durch ein Nicht-ILCS-Assemblerprogramm, das keine ASSEMBH-Strukturmakros verwendet. In diesem Fall muss ILCS mit der ILCS-Routine IT0INITS nachinitialsiert werden.

  Bei Aufruf eines ILCS-Assemblerprogramms durch ein strukturiertes Nicht-ILCS-Assemblerprogramm (Angabe ILCS=NO beim @ENTR-Makro) ist der Aufruf von IT0IN-ITS nicht notwendig.
- Dynamisch Nachladen von ILCS-Objekten in bisher nicht beteiligten Sprachen. In diesem Fall muss die jeweilige Sprachumgebung mit der ILCS-Routine IT0ININ initialisiert werden. Für den Aufruf von IT0ININ steht das ASSEMBH-Strukturmakro @ININ zur Verfügung.

Im Anschluss an die allgemeine Beispielbeschreibung finden Sie im Abschnitt "Abbildung der Quellprogramme" die Abbildung der Quellprogramme und im Abschnitt "Ablaufprotokolle" ein komplettes Ablaufprotokoll zum Übersetzen, Binden und Programmablauf.

6.1 Allgemeine Beispielbeschreibung

Aufrufschema

```
Nicht-ILCS-Assemblerprogramm CALLINTF
  -->
    ILCS-Assemblerprogramm ILCSINTF
      -->
        ILCS-C-Programm CUPRO
           -->
             ILCS-COBOL-Programm COBUPRO
```

Die C- und COBOL-Programme werden im Assemblerprogramm ILCSINTF mittels des BIND-Makros dynamisch nachgeladen.

Nicht-ILCS-Assemblerprogramm CALLINTF

Abbildung des Quellprogramms:

> "*Nicht-ILCS-Assemblerprogramm CALLINTF*" in "Abbildung der Quellprogramme"

Ablaufprotokoll zum Übersetzen:

> *"Übersetzen der Assemblerprogramme CALLINTF (Nicht-ILCS-Hauptprogramm) und ILCSINTF (ILCS-Unterprogramm)"* in "Ablaufprotokolle"

Im Beispiel ist CALLINTF das Hauptprogramm. Wenn CALLINTF seinerseits durch ein Nicht-ILCS-Programm aufrufbar sein soll, muss es folgendermaßen modifiziert werden:

- Beim Aufruf Register sichern, die später benötigt werden
- Vor dem Rücksprung Register wiederherstellen, die zuvor gesichert wurden
- Rücksprung in das rufende Nicht-ILCS-Programm (RETURN)

Beschreibung von CALLINTF

1. Es wird die Initialisierungsroutine IT0INITS aufgerufen, um ILCS nachzuinitialisieren. Register 0 muss mit dem Wert 0 belegt werden (Anzahl der Parameter = 0).
2. IT0INITS liefert in Register 0 die Adresse der generierten PCD. Diese Adresse muss in der Save Area gesichert werden. Die Save Area selbst wird mit dem CRTE-Makro IT0VSA generiert.
3. Register 13 wird mit der Adresse der Save Area geladen (ILCS-Konvention).
4. Ein ILCS-Assemblerprogramm wird aufgerufen, in diesem Beispiel ein Programm mit dem Namen „ILCSINTF“.

ILCS-Assemblerprogramm ILCSINTF

Abbildung des Quellprogramms:

> "*ILCS-Assemblerprogramm ILCSINTF*" in "Abbildung der Quellprogramme"

Ablaufprotokoll zum Übersetzen:

> *"Übersetzen der Assemblerprogramme CALLINTF (Nicht-ILCS-Hauptprogramm) und ILCSINTF (ILCS-Unterprogramm)"* in "Ablaufprotokolle"

Beschreibung von ILCSINTF

1. Um zu vermeiden, dass die ILCS-C- und COBOL-Module mehrmals in den Speicher geladen werden, wird ein Test durchgeführt, ob die C- und COBOL-Module bereits geladen sind.
2. Wenn die Module bereits geladen sind, wird lediglich das C-Modul aufgerufen.
3. Wenn die Module nicht geladen sind,
   - werden die BIND-Makros ausgeführt,
   - wird die ILCS-Routine IT0ININ mit dem Strukturmakro @ININ aufgerufen, um die Laufzeitumgebung für die Sprachen C und COBOL zu initialisieren,
   - wird das C-Modul aufgerufen.

ILCS-C-Programm CUPRO und ILCS-COBOL-Programm COBUPRO

Abbildung der Quellprogramme:

> "*ILCS-C-Programm CUPRO*" und "*ILCS-COBOL-Programm COBUPRO*" in "Abbildung der Quellprogramme"

Ablaufprotokolle zum Übersetzen und Binden:

> *"Übersetzen und Vorbinden des COBOL-Unterprogramms COBUPRO*" und "*Übersetzen und Vorbinden des C-Unterprogramms CUPRO*" in "Ablaufprotokolle"

Beschreibung der Programme

Das C-Programm ruft lediglich das COBOL-Programm auf, das COBOL-Programm quittiert den erfolgreichen Aufruf mit einer Meldung am Terminal.

Da die Programme mit dem BIND-Makro nachgeladen und in den Klasse-4/5-Speicher vorgeladen werden sollen, wird bei der Übersetzung mit dem C++- bzw. COBOL85-Compiler folgende Option angegeben:

```
COMPILER-ACTION=MODULE-GEN(MOD-FORM=LLM,SHAREABLE-CODE=YES)
```

In einem anschließenden Bindelauf mit dem BINDER werden aus der Code-CSECT eine PUBLIC-Slice und aus der Daten-CSECT eine PRIVATE-Slice gebildet.

6.2 Abbildung der Quellprogramme

In diesem Abschnitt sind folgende Quellprogramme enthalten:

- *Nicht-ILCS-Assemblerprogramm CALLINTF*
- *ILCS-Assemblerprogramm ILCSINTF*
- *ILCS-C-Programm CUPRO*
- *ILCS-COBOL-Programm COBUPRO*

Nicht-ILCS-Assemblerprogramm CALLINTF

```
CALLINTF CSECT
         TITLE 'CALL TO ILCSINTF'
CALLINTF AMODE ANY
CALLINTF RMODE ANY
R15      EQU   15
R14      EQU   14
R12      EQU   12
R13      EQU   13
R3       EQU   3
R4       EQU   4
R0       EQU   0
         PRINT NOGEN
         BALR  R3,0
         USING *,R3
* Aufruf von IT0INITS
* Register 0 muss den Wert 0 enthalten (keine Parameter)
         XR    R0,R0
         L     R15,INIADDR
         BASR  R14,R15
* Abspeichern der PCD-Adresse in die Save Area
         ST    R0,ILCVPCD
* Laden der Save Area Adresse
         LA    R13,ILCVAI
* Aufruf von ILCSINTF, um die C- und COBOL-Module zu binden
* (mit dem BIND-Makro) und das erste Mal auszuführen
         L     R15,INTFADDR
         BASR  R14,R15
         TERM
* Definition der Adresse von IT0INITS
INIADDR  DC    V(IT0INITS)
* Definition der Adresse von ILCSINTF
INTFADDR DC    V(ILCSINTF)
         PRINT GEN
* Definition der Save Area
         IT0VSA ILC
         DROP  R3
         END
```

ILCS-Assemblerprogramm ILCSINTF

```
ILCSINTF START
         PRINT GEN
ILCSINTF @ENTR TYP=E,AMODE=ANY,RMODE=ANY,VERS=000,AUTHOR=LEROY,        X
               FUNCT='ILCSINTF',TITLE=YES,ILCS=YES
* Test, ob die Module bereits eingebunden sind
         L     R15,CMODADDR
         @IF   ZE
         LTR   R15,R15
         @THEN
****
* Pro Sprache und Modul wird ein eigenes BIND-Makro abgesetzt
****
* Laden des C-Moduls
         BIND  SYMBOL=PRNT,SYMBLAD=CMODADDR,ALTLIB=YES,MSG=ERROR
* Laden des COBOL-Moduls
         BIND  SYMBOL=COBUPRO,ALTLIB=YES,MSG=ERROR
* Inititialisieren der C- und COBOL-Sprachumgebung durch Aufruf
* der ILCS-Routine IT0ININ mit dem Strukturmakro @ININ
         @ININ
         @BEND
* Aufruf des C-Moduls
         @PASS ADDR=CMODADDR
         @EXIT
* Definition der Adresse des C-Moduls
CMODADDR DS    F
         @END  LTORG=YES,DROP=()
         END
```

ILCS-C-Programm CUPRO

```
#include <stdio.h>
extern void cobupro(void);
void prnt()
{
  printf("CUPRO before\n");
  cobupro();
  printf("CUPRO after1\n");
  cobupro();
  printf("CUPRO after2\n");
}
```

ILCS-COBOL-Programm COBUPRO

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBUPRO.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
DIALOG SECTION.
DIAL.
    DISPLAY "COBUPRO WAS HERE" UPON T.
BYEBYE.
    EXIT PROGRAM.
```

6.3 Ablaufprotokolle

In diesem Abschnitt sind folgende Ablaufprotokolle enthalten:

- *Übersetzen der Assemblerprogramme CALLINTF (Nicht-ILCS-Hauptprogramm) und ILCSINTF (ILCS-Unterprogramm)*
- *Übersetzen und Vorbinden des COBOL-Programms COBUPRO*
- *Übersetzen und Vorbinden des C-Programms CUPRO*
- *Binden, Laden und Starten der Programmausführungseinheit*

Übersetzen der Assemblerprogramme CALLINTF (Nicht-ILCS-Hauptprogramm) undILCSINTF (ILCS-Unterprogramm)

```
(IN)     /START-ASSEMBH
(OUT)    %  BLS0523 ELEMENT 'ASSEMBH', VERSION '013', TYPE 'C' FROM LIBRARY
        ':2OSH:$TSOS.SYSPRG.ASSEMBH.013' IN
PROCESS
(OUT)    %  BLS0500 PROGRAM 'ASSEMBH', VERSION '01.3A02' OF '2012-04-04'
         LOADED
(OUT)    %  BLS0552 COPYRIGHT (C) FUJITSU TECHNOLOGY SOLUTIONS   2012.
         ALL RIGHTS RESERVED
(OUT)    %  ASS6010 V01.3A02 OF BS2000 ASSEMBH  READY
(IN)     //COMPILE SOURCE=*LIB(LIB=PLAM.ASSEMBH,ELEM=CALLINTF.ASM), -
         MACRO-LIB=($.SYSLNK.CRTE,$.SYSLIB.BS2CP.180),MODULE-
LIB=PLAM.ASSEMBH, -
         LISTING=*PAR(OUTPUT=*LIB(LIB=PLAM.ASSEMBH,ELEM=CALLINTF))
(OUT)    %  ASS6011 ASSEMBLY TIME: 202 MSEC
(OUT)    %  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
(OUT)    %  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
(OUT)    %  ASS6006 LISTING GENERATOR TIME: 78 MSEC
(IN)     //COMPILE SOURCE=*LIB(LIB=PLAM.ASSEMBH,ELEM=ILCSINTF.ASM),-
         MACRO-LIB=($.SYSLIB.ASSEMBH.013,$.SYSLIB.BS2CP.180,
         (MODULE-LIB=PLAM.ASSEMBH, -
         LISTING=*PAR(OUTPUT=*LIB(LIB=PLAM.ASSEMBH,ELEM=ILCSINTF))
(OUT)    %  ASS6011 ASSEMBLY TIME: 579 MSEC
(OUT)    %  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
(OUT)    %  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
(OUT)    %  ASS6006 LISTING GENERATOR TIME: 126 MSEC
(IN)     //END
(OUT)    %  ASS6012 END OF ASSEMBH
```

Übersetzen und Vorbinden des COBOL-Programms COBUPRO

```
(IN)     /START-COBOL2000-COMPILER -
         SOURCE=*LIB(LIB=PLAM.COBOL,ELEM=COBUPRO.COB),-
         COMPILER-ACTION=MODULE-GEN(SHAREABLE-CODE=YES,MODULE-FORM=LLM),-
         MODULE-OUTPUT=*LIB(LIB=PLAM.COBOL,ELEM=COBUPRO),-
         LISTING=*PAR(OUTPUT=*LIB(LIB=PLAM.COBOL))
(OUT)    %  BLS0500 PROGRAM 'COBOL2000', VERSION '01.5A00' OF '2009-03-12'
LOADED
(OUT)         %  CBL9000 FUJITSU TECHNOLOGY SOLUTIONS 2009
         ALL RIGHTS RESERVED
(OUT)    %  CBL9017 COMPILATION INITIATED, VERSION IS  V01.5A00
(OUT)    %  CBL9097 COMPILATION COMPLETED WITHOUT ERRORS
(OUT)    %  CBL9004 COMPILATION OF COBUPRO USED  0.3494 CPU SECONDS
(IN)     /START-BINDER
(OUT)    %  BND0500 BINDER VERSION 'V02.6A30' STARTED
(IN)     //START-LLM-CREAT INT-NAME=COB,SLICE-DEFINITION=BY-
ATTRIB(PUBLIC=YES)
(IN)     //INCLUDE-MODULE LIB=PLAM.COBOL,ELEM=COBUPRO
(IN)     //SAVE-LLM LIB=PLAM.COBOL-SHARE,ELEM=COBUPROSHARE
(OUT)    %  BND3101 SOME EXTERNAL REFERENCES UNRESOLVED
(OUT)    %  BND1501 LLM FORMAT: '2'
(IN)     //END
(OUT)    %  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED
         EXTERNAL'
```

Übersetzen und Vorbinden des C-Programms C

```
(IN)     /START-CPLUS-COMPILER
(OUT)    %  BLS0523 ELEMENT 'SDFCC', VERSION '03.2D11', TYPE 'L' FROM LIBRARY
         ':2OSH:$TSOS.SYSLNK.CPP-S.032' IN PROCESS
(OUT)    %  BLS0524 LLM 'SDFCC', VERSION '03.2D11' OF '2012-04-06 08:31:13'
LOADED
(OUT)    %  BLS0551 COPYRIGHT (C) Fujitsu Technology Solutions 2012.
         ALL RIGHTS RESERVED
(OUT)    %  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.2C11
(IN)     //MOD-MOD-PROP SHAREABLE-CODE=*YES
(IN)     //MOD-SOU-PROP C
(IN)     //MOD-LIS-PROP OPTIONS=*YES,SOURCE=*YES(MIN-MSG-WEI=*WARN),-
         OUTPUT=*LIB(LIB=PLAM.C)
(IN)     //COMPILE SOURCE=*LIB(LIB=PLAM.C,ELEM=CUPRO.C),-
         MODULE-OUTPUT=*LIB(LIB=PLAM.C)
(OUT)    %  CDR9907 : NOTES: 7  WARNINGS: 0  ERRORS: 0  FATALS: 0
(OUT)    %  CDR9937 : MODULES GENERATED, CPU TIME USED = 0.0113 SEC
(OUT)    %  CDR9992 : END
(OUT)    %  CCM0998 CPU TIME USED: 1.1996 SECONDS
(IN)        /START-PROG $BINDER
(OUT)    %  BLS0500 PROGRAM 'BINDER', VERSION 'V02.6A30' OF '2010-10-19'
LOADED
(OUT)    %  BLS0552 COPYRIGHT (C) FUJITSU TECHNOLOGY SOLUTIONS   2009. ALL
RIGHTS          RESERVED
(OUT)    (MSG) %  %  BLS0519 PROGRAM '$BINDER' LOADED
(IN)     //START-LLM-CREAT INT-NAME=C,SLICE-DEFINITION=BY-ATTRIB(PUBLIC=YES)
(IN)     //INCLUDE-MODULE LIB=PLAM.C,ELEM=CUPRO
(IN)     //SAVE-LLM LIB=PLAM.C-SHARE,ELEM=CUPROSHARE
(OUT)    %  BND3101 SOME EXTERNAL REFERENCES UNRESOLVED
(OUT)    %  BND3102 SOME WEAK EXTERNS UNRESOLVED
(OUT)    %  BND1501 LLM FORMAT: '2'
(IN)     //END
```

Binden, Laden und Starten der Programmausführungseinheit

```
(IN)     /START-PROG $BINDER
(OUT)    %  BLS0500 PROGRAM 'BINDER', VERSION 'V02.6A30' OF '2010-10-19' LOADED
(OUT)    %  BLS0552 COPYRIGHT (C) FUJITSU TECHNOLOGY SOLUTIONS   2009.
         ALL RIGHTS RESERVED
(IN)     //START-LLM-CREAT INT-NAME=ASS
(IN)     //INCLUDE-MODULE LIB=PLAM.ASSEMBH,ELEM=CALLINTF
(IN)     //INCLUDE-MODULE LIB=PLAM.ASSEMBH,ELEM=ILCSINTF
(IN)     //RESOLVE LIB=$.SYSLNK.CRTE.PARTIAL-BIND
(IN)     //MODIFY-SYMBOL-VISIBILITY VISIBLE=NO
*)
(OUT)    %  BND1111 '304' SYMBOL(S) PROCESSED IN CURRENT STATEMENT
(IN)     //SAVE-LLM LIB=PLAM.PROGRAM,ELEM='START-CALLINTF'
(OUT)    %  BND3101 SOME EXTERNAL REFERENCES UNRESOLVED
(OUT)    %  BND3102 SOME WEAK EXTERNS UNRESOLVED
(OUT)    %  BND1501 LLM FORMAT: '1'
(IN)     //END
(OUT)    %  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED
         EXTERNAL'
(IN)     /ADD-FILE-LINK LINK=BLSLIB01,FILE-NAME=PLAM.COBOL-SHARE
(IN)     /ADD-FILE-LINK LINK=BLSLIB02,FILE-NAME=PLAM.C-SHARE
(IN)     /ADD-FILE-LINK LINK=BLSLIB03,FILE-NAME=$.SYSLNK.CRTE
*)
(IN)     /ADD-FILE-LINK LINK=BLSLIB04,FILE-NAME=$.SYSLIB.ASSEMBH.013
(IN)     /START-PROG *MOD(LIB=PLAM.PROGRAM,ELEM=START-CALLINTF,-
         RUN-MODE=ADV(ALT-LIB=YES),PROG-MODE=ANY)
(OUT)    %  BLS0523 ELEMENT 'START-CALLINTF', VERSION '@' FROM LIBRARY
         ':2OSC:$MANUBSP.PLAM.PROGRAM' IN PROCESS
(OUT)    %  BLS0524 LLM 'ASS', VERSION ' ' OF '2012-05-14 19:42:23' LOADED
(OUT)    (MSG) %  %  BLS0519 PROGRAM 'START-CALLINTF' LOADED
(OUT)    CUPRO before
(OUT)    COBUPRO WAS HERE
(OUT)    CUPRO after1
(OUT)    COBUPRO WAS HERE
(OUT)    CUPRO after2
```

*) Diese beiden Anweisungen sind nicht zwingend vorgeschrieben. Es wird aber empfohlen, sie anzugeben.

7 Literatur

Die Handbücher finden Sie im Internet unter http://manuals.ts.fujitsu.com. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie auch in gedruckter Form bestellen.

[1] **BS2000**
Softbooks Deutsch
CD-ROM

[2] **COBOL2000** (BS2000)
COBOL-Compiler
Benutzerhandbuch

[3] **COBOL2000** (BS2000)
COBOL-Compiler
Sprachbeschreibung

[4] **COBOL85** (BS2000)
COBOL-Compiler
Benutzerhandbuch

[5] **COBOL85** (BS2000)
COBOL-Compiler
Beschreibung

[6] **C** (BS2000)
C-Compiler
Benutzerhandbuch

[7] **C-Bibliotheksfunktionen** (BS2000)
Benutzerhandbuch

[8] **C-Bibliotheksfunktionen** (BS2000)
für POSIX-Anwendungen
Referenzhandbuch

[9] **C/C++** (BS2000)
C/C++-Compiler
Benutzerhandbuch

[10] **C++** (BS2000)
C++-Bibliotheksfunktionen
Referenzhandbuch

[11] **C/C++** (BS2000)
POSIX-Kommandos des C/C++-Compilers
Benutzerhandbuch

[12] **ASSEMBH** (BS2000)
Benutzerhandbuch

[13] **POSIX** (BS2000)
Grundlagen für Anwender und Systemverwalter
Benutzerhandbuch

[14] **BLSSERV**
Bindelader-Starter in BS2000
Benutzerhandbuch

[15] **BINDER**
Binder in BS2000
Benutzerhandbuch

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

[17] **IMON** (BS2000)
Installationsmonitor
Benutzerhandbuch

[18] **DSSM/SSCM**
Verwaltung von Subsystemen in BS2000
Benutzerhandbuch

[19] **BS2000 OSD/BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

[20] **BS2000 OSD/BC**
Einführung in die Systembetreuung
Benutzerhandbuch

[21] **BS2000 OSD/BC**
Systeminstallation
Benutzerhandbuch