Inhalt

Einleitung

Die Sprache Pascal-XT ist eine Erweiterung von Norm-Pascal und wird auf verschiedenen Rechnern mit unterschiedlichen Betriebssystemen angeboten. Dieses Handbuch beschreibt die Handhabung des Compilers unter dem Betriebssystem BS2000.

Welche Vorkenntnisse sind notwendig?

Sie sollten über Kenntnisse der Sprache Pascal-XT, der BS2000-Kommandosprache, des Datenverwaltungssystems und der Binder TSOSLNK und DLL verfügen.

Literaturhinweise werden im Text mit Ziffern in eckigen Klammern angegeben. Der genaue Titel jeder Druckschrift, auf die verwiesen wird, ist im Literaturverzeichnis aufgeführt.

Für diese Version 2.2A gilt weiterhin der Sprachumfang von Pascal-XT V2.1, der im Handbuch Pascal-XT (SINIX, BS2000), Sprachbeschreibung [1] beschrieben ist.

Was enthält das Benutzerhandbuch?

Das Benutzerhandbuch beschreibt

- die Bedienung des Pascal-XT Programmiersystems
- die BS2000 spezifischen Eigenschaften des Pascal-XT Compilers
- die Zuordnung von BS2000-Dateien zu Pascal-Dateien
- das Binden und Ausführen von Programmen
- die Sprachverknüpfung mit externen Programmen (ILCS-Schnittstelle)
- den UTM-Anschluß
- die Testhilfe PATH und das Testen von Programmen
- die Laufzeit-Fehlermeldungen
- die Gegenüberstellung der Sprachen Pascal (BS2000) V3.x und Pascal-XT
- die vordefinierten BS2000-spezifischen Pakete.

Benutzereingaben

In den Beispielen werden Benutzereingaben durch Rasterung hervorgehoben, z. B.:

```
/EXEC $USERID.PASCAL-XT
```

Änderungen gegenüber der Vorversion

Abschnitt	Stichwort	neu	geän-dert	ent-fallen
2.3	Beispiel			x
2.6.3	CALL-STATEMENT-FILE, PLAM-Versionsbezeichnung		 x	
2.6.4	Compileroption MESSAGE-LEVEL COMPILE-UNIT, PLAM-Versionsbezeichnung	x	 x	
2.6.6	EDIT-UNIT, PLAM-Versionsbezeichnung		x	
2.6.8	MODIFY-COMPILE, PLAM-Versionsbezeichnung		x	
2.6.9	MODIFY-EDIT, PLAM-Versionsbezeichnung		x	
2.6.11	RUN-PROGRAM bei Sprachverknüpfungen	x		
2.6.12	SHOW-ATTRIBUTES, PLAM-Versionsbezeichnung		x	
3	Projektdatei		x	
4.1	Pascal-XT-Compiler, Benutzung der Projektdatei		 x	
4.2	Ulp-genaue mathematische Routinen (5)	x		
4.5	Compileroption MESSAGE-LEVEL Tabelle der Compileroptionen Beschreibung der Angabe von Optionen	x	 x x	
4.6	Vierstellige Jahreszahl in der Übersetzungsliste	x		
4.6.2	Compilation summary: Ausgabe der unterdrückten Meldungen		x	
4.6.3	Ausgabe der unterdrückten Meldungen im Compilation summary und auf SYSOUT		x	
5.2.3	PLAM-Versionsbezeichnung Hinweis		x	 x
5.3.3	ASSIGNFILE, PLAM-Versionsbezeichnung		x	
6.1	Binden, Allgemeines Pascal-XT-Laufzeitsystem und ILCS Kompatibilität zw. versch. Versionen	 x x	x	
6.2.2	Vorbinden zu Großmodulen		x	
6.5	Lizenzschutz	x		

Abschnitt	Stichwort	neu	geän-dert	ent-fallen
7	Sprachverknüpfungen, allgemeiner Teil	x		
7.1	Allgemeines Programm-Kommunikationsschnittstelle ILCS	 x		x
7.2	Anschluß fremdsprachiger Unterprogramme		x	
7.3	Aufruf durch fremdsprachige Programme		x	
8	UTM-Anschluß Neue Schnittstelle ILCS Sprachverknüpfungen Speicheranforderung mit "New" Fehlerbehandlung Binden	 x x x	x x x	
9	Die Testhilfe PATH		x	
9.1.4.3	Post-Mortem-Testpunkt		x	
9.1.4.4	Exception-Testpunkt		x	
9.2.2.7	Vierstellige Jahreszahl bei dem PATH-Kommando SHOW-UNITS	x		
9.2.2.9	PATH-Kommando KILL		x	
9.5	Restart des Testlings		x	
10	Laufzeitfehler und Fehlerbehandlung		x	
10.1	STXIT-Ereignisse und ILCS		x	
10.2	Fehlerbehandlung und Ausgaben im Fehlerfall		x	
10.2.1	Behandlung von SEH-Ereignissen	x		
10.2.2	Pascal-XT-Break_Error		x	
10.2.3	Sprachverknüpfungen zwischen Pascal-XT und Assembler	x		
10.2.4	Ausgaben bei einem Laufzeitfehler Dynamische Aufrufkette Einschränkungen zu Versionen <= 2.1A		x x	 x
A.10	Kompatibilitätsprobleme zwischen den Pascal-XT-Versionen V2.2 und V3.0	x		

Neben diesen Änderungen sind Beispiele ergänzt und korrigiert sowie Textformatierungen und Schreibfehler verbessert worden.

Das Pascal-XT Programmiersystem

Aufruf und Überblick

Das Pascal-XT Programmiersystem stellt eine komfortable Umgebung zur Verfügung, um interaktiv Programme erstellen, übersetzen und testen zu können, ohne dabei das Programmiersystem verlassen zu müssen. Daneben bietet das System noch weitere Funktionen zur Erleichterung der Softwareentwicklung an. Der Normalbetrieb des Programmiersystems ist der Dialogbetrieb. Es kann aber auch mit seinem vollen Funktionsumfang im Stapelbetrieb benutzt werden.

Im Bild 2-1 sind die Anweisungen des Programmiersystems und die Zugriffsmöglichkeiten auf die verschiedenen BS2000-Dateien dargestellt.

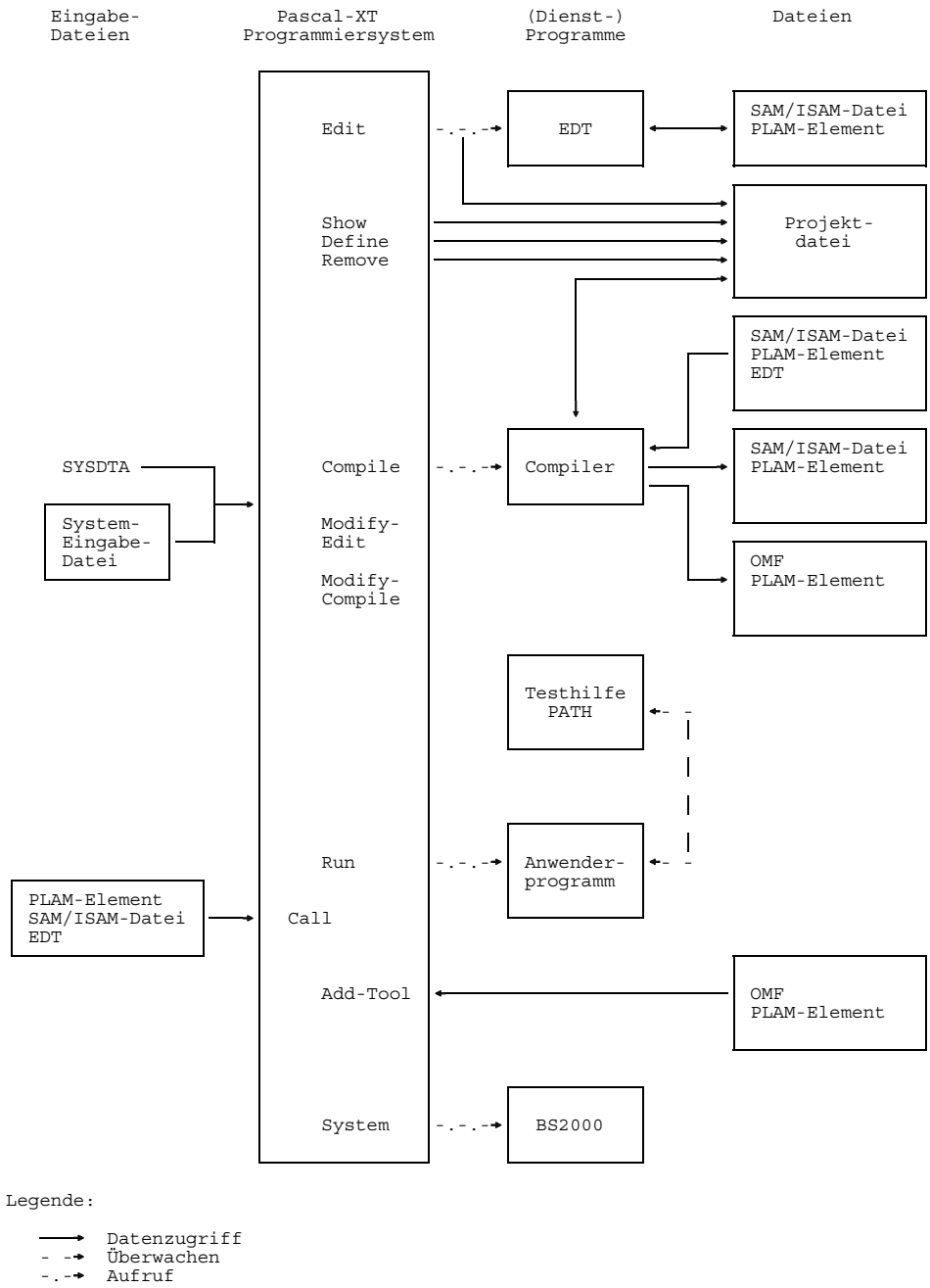


Bild 2-1 Anweisungen und Zugriffsmöglichkeiten des Pascal-XT Programmiersystems

Aufruf des Pascal-XT Programmiersystems

Das Pascal-XT-Programmiersystem wird mit dem EXEC-Kommando aufgerufen (siehe [7]). Im folgenden wird angenommen, daß das Programmiersystem unter der Systemkennung $TSOS verfügbar ist. In diesem Fall lautet der Aufruf:

```
/EXEC $PASCAL-XT
```

Ist das Programmiersystem unter einer anderen Benutzerkennung (USERID) verfügbar, lautet der Aufruf:

```
/EXEC $USERID.PASCAL-XT
```

Anweisungen an das Programmiersystem interpretiert der Kommandoprozessor SDF [2], der verschiedene Stufen der Benutzerführung anbietet (siehe nächster Abschnitt). In Abhängigkeit von der eingestellten Stufe meldet sich das Programmiersystem mit einer Eingabeaufforderung (`//` oder `STMT`) oder mit einem Menü. Der Benutzer kann dann die folgenden Anweisungen eingeben. Ihre genaue Syntax und Bedeutung ist in Abschnitt 2.6 beschrieben.

ADD-TOOL Pascal-Programm in den Hauptspeicher laden. Es kann vom Programmiersystem aus beliebig oft ausgeführt werden.

CALL-STATEMENT-FILE
Anweisungen an das Programmiersystem aus einer Datei ausführen.

COMPILE-UNIT Pascal-XT-Compiler aufrufen.

DEFINE-PROJECT-FILE
Projektdatei einstellen, die der Compiler für den Zugriff auf Spezifikationen benötigt.

EDIT-UNIT Editor EDT (siehe [13]) aufrufen.

END Programmiersystem beenden.

MODIFY-COMPILE Operanden-Standardwerte der COMPILE-Anweisung temporär ändern.

MODIFY-EDIT Operanden-Standardwerte der EDIT-Anweisung temporär ändern.

MODIFY-SDF-OPTIONS
SDF-Optionen (siehe [2]) ändern.

REMOVE-DIRECTORY-ENTRY
Eintrag in der aktuell eingestellten Projektdatei löschen.

RUN-PROGRAM Tool oder Programm mit oder ohne Pascal-XT-Testhilfe ausführen.

SHOW-ATTRIBUTES
: Informationen über das Programmiersystem und die aktuell eingestellte Projektdatei ausgeben.

SHOW-SDF-OPTIONS
: SDF-Optionen ausgeben (siehe [2]).

STEP
: Wiederaufsetzpunkt nach in Batch-Jobs oder DO-Prozeduren aufgetretenen Fehlern definieren.

SYSTEM-COMMAND
: In den BS2000-Kommandomodus wechseln oder BS2000-Systemkommando ausführen.

Benutzerführung

Das Pascal-XT Programmiersystem benutzt zur Analyse der eingegebenen Anweisungen an das Programmiersystem den Kommandoprozessor SDF (System Dialog Facility). Dieser bietet mehrere Stufen der Benutzerführung an. Im *Expertenmodus* können die Anweisungen in kürzester Form eingegeben werden, wobei im Fehlerfall eine Anweisung nochmals vollständig wiederholt werden muß. Der *ungeführte Dialog*, als nächsthöhere Stufe, bietet dem Benutzer bei einer fehlerhaften Eingabe die fehlerhaften Operanden zur Korrektur an und übernimmt die korrekt eingegebenen Werte. Der *geführte Dialog* schließlich bietet dem Benutzer die möglichen Anweisungen in Form geführter Dialog-Menüs an, in denen nur noch die Operandenwerte einzutragen sind. Im geführten Dialog kann nochmals eine von drei Stufen (minimum, medium, maximum) ausgewählt werden. Bei Angabe von minimum werden nur die Operanden, bei medium die Operanden mit den möglichen Wertealternativen und bei maximum zusätzliche Hilfetexte ausgegeben.

In diesem Manual werden nur die wichtigsten Eigenschaften des Expertenmodus beschrieben. Weiterführende Informationen sind unter [2] zu finden.

Eingabeaufforderung

Die Eingabeaufforderung des Pascal-XT-Programmiersystems im Expertenmodus besteht aus zwei Schrägstrichen //. Die Eingabeaufforderung der BS2000 Kommandoebene besteht aus einem Schrägstrich /.

Hilfestufen

Wer sich nicht mehr sicher ist, welche Anweisungen, Operanden und Operandenwerte möglich sind, kann sich in folgenden Fällen Hilfen ausgeben lassen.

– Welche Anweisungen gibt es im Pascal-XT-Programmiersystem?

```
//?
```

Wird hinter der Eingabeaufforderung ein Fragezeichen eingegeben, dann werden alle verfügbaren Anweisungen des Programmiersystems ausgegeben.

– Welche Operanden kann man für eine Anweisung angeben?

```
//anweisung?
```

Wird unmittelbar hinter dem Namen einer Anweisung ein Fragezeichen angegeben, dann werden die Operanden dieser Anweisung in einem Menü angeboten. Nur die gewünschten Operandenwerte müssen dann noch eingesetzt werden.

– Welche Operandenwerte kann man für einen Operanden angeben?

```
//anweisung operand=?
```

Wird anstelle eines Operandenwertes ein Fragezeichen eingegeben, werden die für den angegebenen Operanden zulässigen Operandenwerte ausgegeben.

Eingabe im Blockmodus

Mehrere Anweisungen können in einem Block eingegeben werden, indem sie durch ein "logisches Zeilenende" (Taste LZE) getrennt werden. Tritt bei der Analyse oder Ausführung einer Anweisung ein Fehler auf, dann werden die restlichen Anweisungen des Blocks nicht mehr ausgeführt.

Funktionstasten

K1 Im ungeführten und geführten Dialog wird das aktuelle Menü abgebrochen und zur übergeordneten Menüstufe gewechselt.

K2 Die eingegebene Anweisung wird unterbrochen und in den BS2000 Kommandomodus verzweigt. Nach Eingabe von /RESUME wird die Anweisung fortgesetzt, nach Eingabe von /INTR abgebrochen.
In einem ablaufenden Pascalprogramm wird durch die Eingabe von K2 /INTR ein Break_Error erzeugt (siehe Kap. 10), der unbehandelt zum Programmabbruch führt.

Erweiterung des Programmiersystems

Das Pascal-XT Programmiersystem bietet nur eine minimale Anzahl von Anweisungen für den Software Entwicklungsprozeß an. Es ist jedoch als offenes Programmiersystem konzipiert, um dem Anwender eine Erweiterung um individuelle Anweisungen, die er häufiger benötigt, zu ermöglichen. Die Realisierung dieser Anweisungen erfolgt durch Programme, die im folgenden als Werkzeuge oder Tools bezeichnet werden. Sie müssen vor ihrer Benutzung mit der Anweisung ADD-TOOL (siehe 2.6.2) geladen und dem Programmiersystem bekannt gemacht werden. Im Gegensatz zu den vordefinierten Anweisungen des Programmiersystems können diese Werkzeuge nicht durch Angabe ihres Namens aufgerufen werden, sondern sie werden durch die RUN-Anweisung (siehe 2.6.11) gestartet.

Anforderungen an Werkzeuge

Werkzeuge sind Pascal-Programme oder Programme, die von anderen Sprachübersetzern erzeugt worden sind und von einem Pascal-Rahmenprogramm aufgerufen werden. Programme, die nicht wenigstens einen "Pascal-Rahmen" besitzen, können nicht in das Programmiersystem aufgenommen werden.

Programme, die bereits als Lademodule (Phasen) vorliegen, können nicht als Werkzeuge eingesetzt werden.

Aufnahme von Werkzeugen ins Programmiersystem

Werkzeuge werden durch die ADD-TOOL-Anweisung in den Hauptspeicher geladen und können bis zum Verlassen des Programmiersystems nicht mehr entladen werden. Externreferenzen werden durch den Autolink-Mechanismus des DLL aufgelöst.

Ein Werkzeug ist automatisch unter dem Namen bekannt, der als Elementnamen im Operanden TOOL der ADD-TOOL-Anweisung angegeben wurde. Es ist aber auch möglich, das Werkzeug über einen kürzeren oder verständlicheren Namen anzusprechen, einen sogenannten *Aliasnamen*. Dieser Aliasname kann in der ADD-TOOL-Anweisung angegeben werden. Das Werkzeug ist dann nur noch unter diesem Aliasnamen bekannt.

Hinweis

Tools können erst ab BS2000 Version 8 aus PLAM-Bibliotheken geladen werden.

Aufruf eines Werkzeugs

Die Ausführung eines Werkzeugs erfolgt durch die Anweisung RUN-PROGRAM (siehe 2.6.11) mit Angabe des Namens des Werkzeuges. Sofern das Werkzeug Programmparameter besitzt, können diesen im Operanden PARAMETER aktuelle BS2000-Dateien zugewiesen werden. Das Programmiersystem merkt sich diese Zuordnungen für weitere Aufrufe und setzt sie als Standardwerte ein, wenn bei folgenden Aufrufen keine neuen Zuordnungen angegeben werden.

Informationen über vorhandene Werkzeuge

Die Anweisung SHOW-ATTRIBUTES (siehe 2.6.12) gibt eine Übersicht über die vorhandenen Werkzeuge aus. Neben den Namen der Werkzeuge werden, sofern bei einem früheren Aufruf angegeben, die Zuordnungen von aktuellen BS2000-Dateien zu den Programmparametern ausgegeben.

Benutzung von Werkzeugen

Werkzeuge bleiben bis zum Verlassen des Programmiersystems im Benutzerspeicher geladen. Dadurch können sie schnell aufgerufen werden, da bei den Aufrufen das Laden entfällt. Andererseits belegen sie permanent Speicher, was bei Übersetzungen oder Ausführungen großer Programme zu Speicherengpässen führen kann. Es empfiehlt sich daher, nur kleinere Programme als Werkzeuge einzusetzen.

Das erste Beispiel unter 2.7 *Entwickeln eines Hauptprogramms* zeigt, wie mit ADD-TOOL ein Programm, das eine Datei zeilenweise auf Standardausgabe schreibt, als Tool in die Programmierumgebung geladen werden kann (siehe Erläuterung (20)).

Beendigungsverhalten des Programmiersystems

Das Beendigungsverhalten des Programmiersystems entspricht den Konventionen für das Programm-Beendigungsverhalten von Benutzerprogrammen im BS2000. Durch das Beendigungsverhalten sollen Schäden (Zerstörung von Daten) bei fehlerhaften Programmen minimiert werden.

Normale Programm-Beendigung

Das Programmiersystem ist normal beendet worden. Alle Anweisungen des Programmiersystems wurden fehlerfrei ausgeführt.

Abnormale Programm-Beendigung

Beim Aufruf einer Anweisung des Programmiersystems ist ein Fehler aufgetreten oder es trat ein Fehler im Programmiersystem bzw. Compiler auf. Vom Betriebssystem wird die Meldung `ABNORMAL PROGRAM TERMINATION` ausgegeben. Im Stapelbetrieb oder in einer DO-Prozedur wird anschließend der Spin-Off-Mechanismus aktiviert und zum nächsten Jobstep verzweigt.

In Jobvariablen [8] für die Programmüberwachung wird das Beendigungsverhalten mitprotokolliert bzw. näher erläutert. Eine solche Jobvariable kann beim Aufruf des Programmiersystems angegeben werden (siehe Beispiel am Ende des Abschnitts).

Jobvariable für die Programmüberwachung bestehen aus einer 3 Byte Zustandsanzeige und einer 4 Byte Rückkehrcode-Anzeige. Das erste Byte im Rückkehrcode gibt den Beendigungscode an, die restlichen 3 Byte liefern weitere Programminformationen.

Der Beendigungscode und die Zustandsanzeige werden erst bei Beendigung eines Programms bzw. des Programmiersystems gesetzt.

Zustandsanzeige

Die Zustandsanzeige wird vom Betriebssystem gesetzt. Beim Starten eines Programms wird sie auf den Wert ’$R ’, bei normaler Programm-Beendigung auf den Wert ’$T ’ und bei abnormaler Beendigung auf ’$A ’ gesetzt.

Beendigungscode

Im Programmiersystem liefert jede Anweisung einen Beendigungscode (siehe auch 6.4). Der Beendigungscode der RUN-Anweisung ist gleich dem Beendigungscode des ausgeführten Programms. Der größte auftretende Beendigungscode bestimmt das Beendigungsverhalten des Programmiersystems. Der Beendigungscode kann folgende Werte annehmen:

’0’ = Alle Anweisungen des Programmiersystems wurden fehlerfrei ausgeführt.

’1’ = Alle Anweisungen des Programmiersystems wurden fehlerfrei ausgeführt. Vom Compiler wurden allerdings bei einer Übersetzung Warnungen gemeldet.

’2’ = Eine (oder mehrere) Anweisung(en) des Programmiersystems war(en) entweder syntaktisch falsch oder führte(n) während der Ausführung zu einem Fehler. Tritt während einer Übersetzung ein Fehler auf, dann wird die Ausführung der COMPILE-Anweisung als fehlerhaft interpretiert.

’3’ = Es ist ein Fehler im Programmiersystem aufgetreten. Der Fehler sollte der Wartung mitgeteilt werden.

Die Beendigungscodes ’0’ und ’1’ führen zu einer normalen, die Codes ’2’ und ’3’ zu einer abnormalen Beendigung des Programmiersystems.

Programminformation

Die Programminformation liefert detaillierte Hinweise zum Beendigungscode. Standardmäßig ist sie mit 3 Leerzeichen (Blanks) besetzt. Im Programmiersystem setzt ausschließlich der Compiler die Programminformation. Werden mehrere Übersetzungen durchgeführt, dann wird beim Verlassen des Programmiersystems die größte Programminformation zurückgegeben.

Der Compiler liefert entsprechend der BS2000-Konvention folgende Programminformationen:

’000’: Die Übersetzung ist fehlerfrei gelaufen.

’002’: Der Compiler hat Warnungen gemeldet.

’004’: Der Compiler hat Fehler gemeldet.

’005’: Es ist ein Benutzerfehler aufgetreten. Mögliche Fehler sind:

- Projektdatei wurde nicht eingestellt.
- eine Spezifikation wurde aus dem EDT übersetzt.
- Speicherüberlauf (Übersetzungseinheit zu groß).
- Unterbrechung mit INTR.
- Fehler beim Eröffnen der angegebenen Dateien.
- Für den Paket- bzw. Programm-Namen existierte kein Eintrag in der Projektdatei.

’006’: Es ist ein Compilerfehler aufgetreten.

Eine Programminformation größer oder gleich ’004’ führt beim Verlassen des Programmiersystems zu einer abnormalen Beendigung.

Beispiel

Das Beispiel zeigt eine DO-Prozedur zum Übersetzen und statischen Binden eines Programms. Nach einem Übersetzungsfehler soll das Programm nicht gebunden werden. Die Jobvariable MONITOR wird zur Programmüberwachung eingesetzt.

```
/PROC A
/DCLJV MONITOR
/SYSFILE SYSDTA=(SYSCMD)
/EXEC $USERID.PASCAL-XT, MONJV=MONITOR
//COMPILE (PLAM.EXAMPLE,FEHLER.PROG), (*STD,*STD), *STD
//END
/EXEC $TSOSLNK
PROGRAM TEST
INCLUDE BEISPIE, PLAM.EXAMPLE
RESOLVE        , PLAM.EXAMPLE
END
/STEP
/GETJV MONITOR
/ENDP
```

Ausführung dieser Prozedur:

```
/DCLJV MONITOR
/SYSFILE SYSDTA=(SYSCMD)
/EXEC $USERID.PASCAL-XT, MONJV=MONITOR
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//COMPILE (PLAM.EXAMPLE,FEHLER.PROG), (*STD,*STD), *STD
   >>> 4 COMPILATION ERRORS DETECTED (WARNINGS: 1;  NOTES: 3)
%  CMD0230 ERROR IN PRECEDING STATEMENT: ALL STATEMENTS WILL BE IGNORED UNTIL '//STEP' IS RECOGNIZED
//END
   END OF THE PASCAL SESSION -  USED TIME = 0.298 SECONDS
%  EXC0732 ABNORMAL PROGRAM TERMINATION. ERROR CODE 'NRT0101' /HELP-MSG NRT0101
%  CMD0206 TERMJ: COMMANDS WILL BE IGNORED UNTIL /STEP OR /LOGOFF OR /ABEND IS RECOGNIZED
/STEP
/GETJV MONITOR
%$A 2004
```

Betriebsarten des Programmiersystems

Anweisungen an und Daten für das Programmiersystem werden von der Systemdatei SYSDTA gelesen. Ausgaben erfolgen auf die Systemdatei SYSOUT.

Das Programmiersystem kann im Dialog- und Stapelbetrieb gleichermaßen eingesetzt werden. Es verhält sich, mit Ausnahme des Spin-Off-Mechanismus, in beiden Betriebsarten gleich. Der Spin-Off-Mechanismus wird aktiviert, wenn eine Anweisung syntaktisch fehlerhaft ist oder wenn bei der Ausführung der Anweisung ein Fehler auftritt. Es werden dann alle Anweisungen bis zur nächsten STEP- oder END-Anweisung überlesen.

Dialogbetrieb

Der Spin-Off-Mechanismus hat keine Wirkung, wenn Anweisungen des Programmiersystems vom Terminal gelesen werden. Wird SYSDTA mit dem SYSFILE-Kommando an eine Datei zugewiesen, dann wird im Fehlerfall der Spin-Off-Mechanismus aktiviert. In einer DO-Prozedur ist SYSDTA einer Datei zugeordnet und eine fehlerhafte Anweisung des Programmiersystems führt folglich zum Aktivieren des Spin-Off-Mechanismus. Eine fehlerhafte Anweisung in einer Anweisungsdatei (siehe CALL-STATEMENT-FILE) aktiviert ebenfalls den Spin-Off-Mechanismus.

Stapelbetrieb

Bei fehlerhaften Anweisungen wird der Spin-Off-Mechanismus aktiviert.

Anweisungen an das Programmiersystem

Definitionen und Hinweise

Für die Beschreibung der Anweisungen an das Programmiersystem werden folgende Metazeichen verwendet und Vereinbarungen getroffen:

`GROSSBUCHSTABEN`
: Großbuchstaben bezeichnen Konstanten und Schlüsselwörter, die wortwörtlich, so wie angegeben (aber nicht als Großbuchstaben), eingegeben werden müssen. Sie können von rechts nach links so weit abgekürzt werden, wie sie eindeutig bleiben.

`ABK``KÜRZUNG`
: Fettgedruckte Großbuchstaben bezeichnen mögliche, eindeutige Abkürzungen von Konstanten und Schlüsselwörtern.

`wert`
: Kleinbuchstaben bezeichnen Metavariable, die bei der Eingabe durch aktuelle Werte ersetzt werden (siehe Metavariable weiter unten).

`( )`
: Runde Klammern gehören zum Operanden und müssen mit eingegeben werden.

<u>NO</u>
: Unterstreichung kennzeichnet einen Standardwert, der automatisch eingesetzt wird, wenn kein Wert eingegeben wird.

: Geschweifte Klammern schließen alternative Werte ein. Sofern keiner dieser Werte als Standardwert gekennzeichnet ist, muß einer der Werte eingegeben werden.

`NO | YES`
: Ein senkrechter Strich zwischen Operandenwerten bedeutet ebenfalls, daß es sich um alternative Werte handelt. Sofern keiner dieser Werte als Standardwert gekennzeichnet ist, muß einer der Werte eingegeben werden.

`[ ]`
: Eckige Klammern schließen optionale Angaben ein, die weggelassen werden dürfen. Die eckigen Klammern selbst sind nicht einzugeben.

Schlüsselwörter

Die Werte von Operanden einer Anweisung können Schlüsselwörter (in der Beschreibung in Großbuchstaben angegeben) sein. Einem Schlüsselwort kann bei der Eingabe ein Stern * vorangestellt werden. Enthält die Syntaxbeschreibung einen Stern, *muß* er bei der Eingabe angegeben werden.

Stellungs- und Schlüsselwortoperanden

Jeder Operand kann entweder als Schlüsselwortoperand oder als Stellungsoperand eingegeben werden.

Ein Schlüsselwortoperand besteht aus dem Namen des Operanden, einem Gleichheitszeichen und dem Operandenwert.

```
//anweisung operand=wert
```

Ein Stellungsoperand besteht aus einer Anzahl von Kommas, die die Position angibt, an der in der Syntaxbeschreibung der Anweisung der Operand steht, und aus dem Operandenwert. Soll z.B. für den dritten Operanden einer Anweisung ein Wert eingegeben werden, müssen für die beiden fehlenden Operanden davor Kommas eingegeben werden.

```
//anweisung ,,wert
```

Die Reihenfolge von Schlüsselwortoperanden ist beliebig. Wenn ein Schlüsselwortoperand eingegeben wurde, kann dahinter für dieselbe Anweisung kein Stellungsoperand mehr angegeben werden.

Abkürzungsregeln

Die Namen von Anweisungen, Operanden und Schlüsselworten können von rechts nach links so weit abgekürzt werden, wie sie eindeutig bleiben.

Es kann abgekürzt werden
- innerhalb eines Namens
- innerhalb eines jeden durch Bindestrich getrennten Teilnamens.

Eindeutige Abkürzungen der jeweiligen Konstanten und Schlüsselwörter sind in der Beschreibung durch Fettdruck hervorgehoben.

Aliasnamen

Ein Aliasname für eine Anweisung oder einen Operanden ist ein Zweitname, unter dem die Anweisung bzw. der Operand ebenfalls eingegeben werden kann. Aliasnamen sind nicht mehr abkürzbar. Sie dienen dazu, häufig benutzte Anweisungen oder Operanden durch einen einzigen Buchstaben oder eine kurze Zeichenfolge angeben zu können.

Ein Aliasname für eine Anweisung oder einen Operanden wird, durch einen senkrechten Strich getrennt, hinter dem Namen der Anweisung oder des Operanden angegeben.

Metavariable

In der Syntaxbeschreibung treten einige Metavariable wiederholt auf. Zur Vermeidung von Wiederholungen sind ihre Definitionen hier erklärt.

pascal-name	Bezeichnet einen Paket- oder Programmnamen. Unterstriche im Bezeichner müssen als Bindestrich angegeben werden.
name	Steht für einen Dateinamen oder einen Programm- oder Paketnamen.
dateiname	Bezeichnet einen Dateinamen entsprechend den BS2000-Konventionen [7].
element	Bezeichnet ein PLAM-Bibliothekselement. Es werden nur Elementnamen mit derselben Syntax wie bei LMS akzeptiert.
vers	Bezeichnet die Version eines PLAM-Elements. Der Bezeichner kann 1 bis 24 Zeichen lang sein und hat dieselbe Syntax wie bei LMS.
typ	Bezeichnet den Typ eines PLAM-Elementes. Der Bezeichner kann 1 bis 8 Zeichen lang sein und hat dieselbe Syntax wie bei LMS.
tool-name	Bezeichnet den Namen eines Tools. Er hat dieselbe Syntax wie ein Dateiname.

Beschreibung von Bibliothekselementen

Bei einigen Operanden können als Werte PLAM-Elemente, bestehend aus Bibliotheksname, Elementname, Typ und Version, angegeben werden. Diese Angaben werden in einer Struktur zusammengefaßt, die durch das Schlüsselwort *LIBRARY eingeleitet wird. In den Syntaxdefinitionen der Operanden wird dieses Schlüsselwort aus Gründen der Übersichtlichkeit weggelassen, da seine Eingabe von SDF nicht verlangt wird. In den Menüs wird das Schlüsselwort von SDF allerdings ausgegeben.

Beispiel

Das Beispiel zeigt äquivalente Angaben des SOURCE-Operanden in der COMPILE-Anweisung.

```
/EXEC $PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//C *LIBRARY(SOURCE-LIBRARY=PLAM.TOOL,SOURCE-ELEMENT=PAS.LMSCALL), *DUMMY

   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C *LIBRARY(PLAM.TOOL, PAS.LMSCALL), *DUMMY
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (PLAM.TOOL, PAS.LMSCALL), *DUMMY
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C SOURCE-LIBRARY=PLAM.TOOL,SOURCE-ELEMENT=PAS.LMSCALL, L=*DUMMY  ——— (01)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//END
   END OF THE PASCAL SESSION - USED TIME = 1.586 SECONDS
/
```

(01) Der Operand SOURCE besteht aus einer Struktur, die die zwei Operanden

SOURCE-LIBRARY und SOURCE-ELEMENT enthält. Da diese beiden Operandennamen in der gesamten COMPILE-Anweisung eindeutig sind, kann sie SDF eindeutig dem Operanden SOURCE zuordnen. Die Angabe der Strukturklammern kann dadurch entfallen.
Die Eingabe des Schlüsselworts L (LISTING) ist zwingend, da nach Eingabe eines Schlüsselwortparameters keine Stellungsparameter mehr angegeben werden dürfen.

Der Operandenwert *UNCHANGED

Im geführten Dialog (Menü) wird bei den Anweisungen COMPILE, MODIFY-COMPILE, EDIT und MODIFY-EDIT als zusätzlicher Operandenwert *UNCHANGED angezeigt, der in den Syntaxbeschreibungen in den entsprechenden Abschnitten nicht aufgeführt ist. Dieser Wert bedeutet, daß für einen fehlenden Operanden bei der Eingabe der aktuell eingestellte Standardwert unverändert (unchanged) übernommen wird. Der Wert kann auch vom Anwender eingegeben werden mit der Wirkung, daß der eingestellte Standardwert nicht verändert wird.
Gibt es aber noch keinen Standardwert für einen Operanden, dann wird ein Fehler gemeldet. Das ist z.B. dann der Fall, wenn bei der Übersetzung aus einer Bibliothek nur der Elementname angegeben wird und zuvor der Bibliotheksname nicht mit MODIFY-COMPILE eingestellt wurde. Der Operandenwert für den Bibliotheksnamen ist *UNCHANGED und steht damit für einen undefinierten Wert.

Beispiel

```
//C (,TEST.PROG),*DUMMY
```

bedeutet in der ausgeschriebenen Form

```
//C (*UNCHANGED, TEST.PROG(*HIGHEST-EXISTING)), *DUMMY
```

und führt zu einer Fehlermeldung, wenn nicht zuvor mittels MODIFY-COMPILE eine Bibliothek eingestellt wurde, z.B.

```
//MODIFY-COMPILE (PLAM.TEST,)
//C (,TEST.PROG),*DUMMY
```

bedeutet in der ausgeschriebenen Form dann

```
//C (PLAM.TEST, TEST.PROG(*HIGHEST-EXISTING)), *DUMMY
```

ADD-TOOL

Programm in den Hauptspeicher laden.

```
ADD-TOOL

     TOOL         =  (LIBRARY = dateiname | *OMF
                     ,ELEMENT = element)

   [,ALIAS-NAME   =  *ELEMENT-NAME | tool-name ]
```

Funktion

ADD-TOOL lädt das im Operanden TOOL angegebene Programm in den Hauptspeicher. Es kann dann beliebig oft mit der RUN-Anweisung ausgeführt werden (siehe 2.3).

Der Spin-Off-Mechanismus wird aktiviert, wenn ein Tool nicht geladen werden kann oder ein Tool mit dem angegebenen Namen bereits existiert.

TOOL = (dateiname,element)
: "element" bezeichnet das Starter-Modul des Programms, das aus der Bindemodulbibliothek "dateiname" bzw. der temporären EAM-Bindemoduldatei (*OMF) geladen wird.

ALIAS-NAME
: Definiert einen neuen Namen, über den das Tool ansprechbar ist. Der Name muß von den Namen bereits geladener Tools verschieden sein.

= *ELEMENT-NAME
: Das Tool ist nur durch den im TOOL-Operanden angegebenen Elementnamen "element" aufrufbar.

= tool-name
: Ist ein frei wählbarer Name für das Tool.

Hinweis

Das Laden eines Tools aus einer PLAM-Bibliothek ist ab BS2000 Version 8 möglich.

CALL-STATEMENT-FILE

Anweisungen aus einer (Anweisungs-)Datei ausführen.

```
CALL-STATEMENT-FILE

                    ⎧dateiname                                                  ⎫
                    ⎪*EDT (WORKFILE = *STD | 0..9)                              ⎪
                    ⎨                                                           ⎬
      STMT-FILE  =  ⎪(LIBRARY = dateiname,                                      ⎪
                    ⎪,ELEMENT = element(VERSION = *HIGHEST-EXISTING | vers      ⎪
                    ⎩                   ,TYP = J | typ ))                       ⎭

     [,PROTOCOL = NO | YES ]
```

In DO-Prozeduren und Batch-Jobs wird als Versionsangabe weiterhin *STD akzeptiert.

Funktion

CALL-STATEMENT-FILE führt eine Folge von Anweisungen aus, die in einer Datei abgespeichert sind. Diese Anweisungsdatei kann eine SAM- oder ISAM-Datei oder ein Bibliotheks-Element sein. Die CALL-Anweisung eröffnet die Datei und liest die Anweisungen sequentiell aus der Datei und führt sie sofort aus.

Die Anweisungsdatei muß *keine* speziellen Eröffnungs- oder Terminierungs-Anweisungen enthalten. Jede Anweisung muß in einer eigenen Zeile stehen.

Aufrufe von CALL-Anweisungen können beliebig geschachtelt sein. Nach Verlassen der Anweisungsdatei wird mit der Anweisung fortgefahren, die unmittelbar dem letzten CALL-Aufruf folgt. Die Anzahl solcher geschachtelten Aufrufe wird nur durch den verfügbaren Hauptspeicher begrenzt.

Enthält die Anweisungsdatei eine END-Anweisung, dann wird nach deren Ausführung das Programmiersystem sofort beendet.

Ein Fehler bei der Analyse oder Ausführung einer Anweisung löst den Spin-Off-Mechanismus aus und es werden alle folgenden Anweisungen bis zum Auftreten einer STEP- oder END-Anweisung überlesen.

STMT-FILE Bezeichnet die Datei oder das Bibliothekselement, aus der die Anweisungen gelesen werden.

= dateiname
Name der Anweisungsdatei.

= *EDT(WORKFILE = *STD | 0..9)

Die Anweisungen werden aus dem Arbeitsbereich (WORKFILE) des Editors EDT gelesen. Es können die Arbeitsbereiche 0 bis 9 angegeben werden. Standardmäßig wird aus dem aktuell eingestellten Arbeitsbereich (*STD) gelesen.

= (dateiname,element(version,typ))

Die Anweisungen werden aus dem Bibliothekselement "element" mit der Version "version" vom Typ "typ" aus der PLAM-Bibliothek "dateiname" gelesen.

Die Angaben für "version" und "typ" sind optional.
Standardwert für "vers" = *HIGHEST-EXISTING
für "typ" = "J".

Die Versionsangabe *HIGHEST-EXISTING bewirkt, daß die Anweisungen aus der höchsten existierenden Version des angegebenen Bibliothekselements gelesen werden.

Die Versionsangabe *STD ist ab Pascal-XT V2.2A im Dialog nicht mehr möglich. In DO-Prozeduren und Batch-Jobs wird weiterhin *STD akzeptiert.

Die Versionsangabe *INCREMENT ist nicht möglich.

PROTOCOL Die aus der Datei gelesenen Anweisungen können auf der Systemdatei SYSOUT mitprotokolliert werden. Jede Anweisung wird bei der Ausgabe mit dem Präfix "(%STMT)" versehen. Fehlerhafte Anweisungen werden immer auf SYSOUT ausgegeben.

= NO Die Protokollierung wird unterdrückt.

= YES Die Anweisungen werden mitprotokolliert.

COMPILE-UNIT

Pascal-XT-Compiler aufrufen.

```
COMPILE-UNIT | C

                  ⎡*EDT (WORKFILE = *STD | 0..9)                                  ⎤
                  ⎢name (KIND = FILE | SPEC | BODY | PROG)                        ⎥
  [ SOURCE     =  ⎢                                                              ⎥
                  ⎢(SOURCE-LIBRARY = dateiname                                    ⎥
                  ⎣,SOURCE-ELEMENT = element(VERSION = ⎧*HIGHEST-EXISTING⎫))     ⎦]
                                                      ⎩vers             ⎭

                  ⎡*SYSOUT                                  ⎤
                  ⎢*SYSLST                                  ⎥
                  ⎢*EDT (WORKFILE = *STD | 0..9)            ⎥
  [,LISTING    =  ⎢                                         ⎥]
                  ⎢*DUMMY                                   ⎥
                  ⎢(LIST-LIBRARY = *STD | dateiname         ⎥
                  ⎣,LIST-ELEMENT = *STD | element)          ⎦

  [,MODULE-LIBRARY =  *OMF (ERASE = YES | NO) | dateiname | *STD ]
  [,ASSEMBLER      =  *BY-SOURCE | ON | OFF ]
  [,CHECK          =  *BY-SOURCE | ON | OFF ]
  [,DEBUG          =  *BY-SOURCE | ON | OFF | RESTRICTED ]
  [,GENERATE       =  *BY-SOURCE | ON | OFF ]
  [,INITIALIZE     =  *BY-SOURCE | ON | OFF ]
  [,LIST | L       =  *BY-SOURCE | ON | OFF ]
  [,MAP            =  *BY-SOURCE | ON | OFF ]
  [,OPTIMIZE       =  *BY-SOURCE | ON | OFF ]
  [,STANDARD       =  *BY-SOURCE | ON | L0 | OFF ]
  [,XREF           =  *BY-SOURCE | ON | OFF ]
  [,LINES-PER-PAGE =  *STD | 11..2147483639 ]
  [,MESSAGE-LEVEL  =  NOTES | WARNINGS | ERRORS ]
```

In DO-Prozeduren und Batch-Jobs wird als Versionsangabe für den SOURCE-Operanden weiterhin *STD akzeptiert.

Funktion

COMPILE-UNIT ruft den Pascal-XT-Compiler auf. Es wird die im SOURCE-Operanden angegebene Quelle übersetzt. Alle vom Compiler erzeugten Listen werden in die im LISTING-Operanden angegebene Datei ausgegeben. Bei fehlerfreier Übersetzung eines Hauptprogramms oder einer Paket-Implementierung werden Objektmodule erzeugt und in der im Operanden MODULE-LIBRARY angegebenen Bindemodulbibliothek abgespeichert. Durch Angabe von Compileroptionen kann der Übersetzungslauf gesteuert werden.

Im SOURCE-Operanden kann anstelle eines Dateinamens auch der Pascal-Bezeichner eines Paketes oder Hauptprogramms angegeben werden. Die Unterscheidung zwischen Dateinamen und Pascal-Bezeichner wird durch den Operanden KIND gesteuert. Diese Arbeitsweise setzt aber voraus, daß eine Projektdatei eingestellt ist und in dieser das Programm bzw. Paket eingetragen ist. Damit eine Übersetzungseinheit (Spezifikation, Implementierung oder Hauptprogramm) in die Projektdatei eingetragen wird, muß diese zuerst aus einer Datei (nicht dem EDT) fehlerfrei übersetzt werden.

Zur Übersetzung eines Pakets oder eines Hauptprogramms, das Pakete importiert, benötigt der Compiler eine Projektdatei, die mit der DEFINE-Anweisung eingestellt wird. Über die Projektdatei findet der Compiler die Spezifikationen importierter Pakete, die für Schnittstellenüberprüfungen benötigt werden. Nach einer fehlerfreien Übersetzung wird in der Projektdatei die Übersetzungseinheit als "gültig" markiert. Diese Information wird benötigt, damit die Rekompilierungen korrekt angezeigt werden können. Die Funktionen der Projektdatei sind in Kapitel 3 ausführlich beschrieben.

Quellprogramme können vom Compiler auch aus einem EDT-Arbeitsbereich gelesen werden. In diesem Fall wird die Projektdatei nicht aktualisiert, da die Quelle i.a. später nicht mehr verfügbar ist. Nach einer fehlerfreien Übersetzung einer Paket-Spezifikation aus einem EDT-Arbeitsbereich wird zusätzlich eine Meldung ausgegeben, daß kein Eintrag in der Projektdatei erfolgt ist.

Die Standardwerte der Operanden können während einer Sitzung mit der MODIFY-COMPILE-Anweisung verändert werden. Diese Änderungen gelten bis zur nächsten Änderung, längstens bis zum Beenden des Programmiersystems.

Der Spin-Off-Mechanismus wird nach folgenden Fehlern aktiviert:

(a) Fehler in der Syntax der COMPILE-Anweisung

(b) Bei der Übersetzung wurden Fehler gefunden

(c) Eine Paket-Spezifikation wurde aus einem EDT-Arbeitsbereich übersetzt

(d) Zur Übersetzung wurde die Projektdatei benötigt. Sie war aber nicht eingestellt

(e) Im SOURCE-Operanden wurde der Name eines Paketes oder Programms angegeben, für das kein Eintrag in der Projektdatei existiert

(f) In SOURCE und LISTING wurde dieselbe Datei angegeben

Bei Angabe einer Jobvariablen beim Aufruf des Programmiersystems wird in der Programminformation des Beendigungscodes (siehe 2.4) der schwerwiegendste Fehler beschrieben, der den Spin-Off-Mechanismus aktiviert hat.

SOURCE Gibt den Namen der Datei an, in der das Quellprogramm enthalten ist, oder den Namen eines Pakets oder Hauptprogramms, das übersetzt werden soll.

= *EDT (WORKFILE = *STD | 0..9)
Das Quellprogramm steht in einem Arbeitsbereich (0..9) des EDT. Standardmäßig wird der aktuell eingestellte Arbeitsbereich (*STD) angesprochen.

= dateiname
Name der Quellprogrammdatei.

= name (KIND = FILE | SPEC | BODY | PROG)
"name" bezeichnet eine Datei, ein Paket oder ein Hauptprogramm. Der Operand KIND legt fest, wie "name" zu interpretieren ist:
= FILE "name" ist der Name einer Datei
= SPEC "name" ist der Pascal-Bezeichner eines Pakets, dessen Spezifikation übersetzt werden soll
= BODY "name" ist der Pascal-Bezeichner eines Pakets, dessen Implementierung übersetzt werden soll
= PROG "name" ist der Pascal-Bezeichner eines Hauptprogramms, das übersetzt werden soll
Die Angabe von SPEC, BODY, oder PROG setzt voraus, daß eine Projektdatei eingestellt ist und daß für die Paket-Spezifikation, Paket-Implementierung bzw. das Hauptprogramm ein Eintrag existiert. Ansonsten wird ein Fehler gemeldet. Die Pascalquelle wird aus der Datei gelesen, die in der Projektdatei angegeben ist. Unterstriche im Pascal-Bezeichner müssen in "name" als Bindestriche ("-") angegeben werden.

= (dateiname, element(vers))
Das Quellprogramm wird aus dem Bibliothekselement "element" mit der Version "vers" aus der PLAM-Bibliothek "dateiname" gelesen. Das Bibliothekselement *muß* vom Typ "S" (für Source) sein. Standardmäßig wird das Element mit der höchsten existierenden Version (*HIGHEST-EXISTING) gelesen.

Die Versionsangabe *STD ist ab Pascal-XT V2.2A im Dialog nicht mehr möglich. In DO-Prozeduren und Batch-Jobs wird *STD weiterhin akzeptiert.

LISTING Gibt den Namen der Datei an, in die der Compiler die erzeugten Listen ausgibt. Die Anzahl der Zeilen pro Seite kann mit der Option LINES-PER-PAGE beeinflußt werden. Die Übersetzung wird mit einer Meldung abgebrochen, wenn die Operanden SOURCE und LISTING dieselbe Datei bezeichnen.

= *SYSOUT
Ausgabe auf die Systemdatei SYSOUT

= *SYSLST
Ausgabe auf die Systemdatei SYSLST

= *EDT (WORKFILE = *STD | 0..9)
Ausgabe in den angegebenen Arbeitsbereich des EDT. Standardmäßig (*STD) ist dies der aktuell eingestellte Bereich. Der Arbeitsbereich wird zuvor gelöscht. Wurde im Operanden SOURCE ebenfalls ein EDT-Arbeitsbereich angegeben, dann müssen die beiden Arbeitsbereiche verschieden sein.

= *DUMMY
Ausgabe auf die Systemdatei *DUMMY. Damit sind alle vom Compiler erzeugte Listen verloren.

= dateiname
Name der Ausgabedatei (SAM-Datei). Eine bereits existierende Datei wird überschrieben.

= (dateiname, element)
Der Compiler schreibt die erzeugte Liste in das Element "element" der PLAM-Bibliothek "dateiname". Der Typ dieses Listenelements ist "P" (für Printfile) und kann vom Benutzer nicht verändert werden.
Existiert bereits ein Listenelement mit demselben Namen, demselben Typ und derselben Version, wird es überschrieben.

Wurde im SOURCE-Operanden ein Bibliothekselement angegeben, dann gilt für das Listenelement:

- Wird für LIST-LIBRARY der Wert *STD angegeben, wird das Listenelement in dieselbe Bibliothek geschrieben, die für SOURCE-LIBRARY angegeben wurde.
- Wird für LIST-ELEMENT der Wert *STD angegeben, dann bekommt das Listenelement (vom Typ "P") denselben Elementnamen, den das Sourceelement (vom Typ "S") hat.

Welche Version das Listenelement bekommt, hängt davon ab, ob das Quellprogramm selbst ein Bibliothekselement ist.

- Wenn ja, erhält das Listenelement dieselbe Version wie das Source-

element. Hat die Version des Sourceelements den Standardwert (ab Pascal-XT V2.2A = *HIGHEST-EXISTING, vorher = *STD), erhält also das Listenelement nicht mehr die höchstmögliche Version, sondern dieselbe Version wie das Sourceelement.

– Wenn nein, erhält das Listenelement die Version *UPPER-LIMIT (= die höchstmögliche Version).

MODULE-LIBRARY

Gibt die Bindemodulbibliothek an, in der nach einer fehlerfreien Übersetzung eines Hauptprogramms (program) oder einer Paket-Implementierung (package body) die erzeugten Objektmodule abgespeichert werden. Die CSECT-Namen der erzeugten Module werden aus dem Namen der Übersetzungseinheit generiert (siehe 4.4).

= *OMF (ERASE = YES | NO)

Die erzeugten Objektmodule werden in der temporären Bindemoduldatei (*OMF) der Benutzertask abgelegt. Mit dem Operanden ERASE wird festgelegt, ob *OMF vor der Übersetzung gelöscht werden soll:
ERASE = YES: *OMF wird zuvor gelöscht
ERASE = NO: *OMF wird zuvor nicht gelöscht (Standardwert)

= *STD Diese Angabe ist nur erlaubt, wenn im SOURCE-Operanden ein Bibliothekselement angegeben wurde. Die erzeugten Objektmodule werden in derselben Bibliothek abgelegt, in der das Sourceelement steht.
Jedes erzeugte Objektmodul wird als ein eigenes Bibliothekselement vom Typ "R" (für Relocatable) abgespeichert. Die Elemente erhalten denselben Namen wie die Objektmodule (CSECT-Namen). Die Elemente erhalten dieselbe Versionsbezeichnung, die die Sourceelemente haben. Existieren bereits Elemente mit demselben Namen, demselben Typ und derselben Version, werden sie überschrieben.

= dateiname

Ist der Name einer PLAM-Bibliothek. Ansonsten gelten dieselben Aussagen wie bei der Angabe *STD.

ASSEMBLER, CHECK, DEBUG, GENERATE, INITIALIZE, LIST, MAP, OPTIMIZE, STANDARD, XREF

Compileroptionen zur Steuerung der Übersetzung. Sie können auch im Pascal-XT-Quellprogramm angegeben werden. Ihre Bedeutung ist in Abschnitt 4.5 beschrieben.

= *BY-SOURCE

steht für den im Quellprogramm angegebenen Wert der Option bzw., wenn dort die betreffende Option nicht angegeben ist, für den Standardwert.

= ON die Compileroption ist eingeschaltet.

Bei der Option STANDARD bewirkt ON, daß der Compiler nur Sprachkonstrukte akzeptiert, die dem Pascal-Standard ISO 7185 Level 1 entsprechen.

= OFF die Compileroption ist ausgeschaltet.

Bei der Option STANDARD bewirkt ON, daß der Compiler alle Sprachkonstrukte akzeptiert, die zum Sprachumfang von Pascal-XT V2.1 gehören.

= L0 bewirkt bei der Option STANDARD, daß der Compiler nur Sprachkonstrukte akzeptiert, die dem Pascal-Standard ISO 7185 Level 0 entsprechen.

LINES-PER-PAGE

Compileroption, die nur als Operand der COMPILE-UNIT-Anweisung angegeben werden kann. Sie legt die Anzahl der Zeilen pro Seite in der Übersetzungsliste fest. Die Zeilenanzahl muß in dem vorgegebenen Bereich 11.. 2147483639 liegen. Der Standardwert (*STD) ist 63.

MESSAGE-LEVEL

Compileroption, die nur als Operand der COMPILE-UNIT-Anweisung angegeben werden kann. Sie legt fest, welche Meldungen in der Übersetzungsliste ausgegeben werden sollen. Es gibt drei Arten von Meldungen: Notes, Warnings und Errors (siehe 4.6.3).

= NOTES

Standardwert.
Als Errors (Syntax- und Semantikfehler), Warnings (potentielle Laufzeitfehler) oder Notes (Hinweise) eingestufte Meldungen werden in der Übersetzungsliste ausgegeben.

= WARNINGS

Als Errors oder als Warnings eingestufte Meldungen werden ausgegeben.

= ERRORS

Nur als Errors eingestufte Meldungen werden ausgegeben.

Hinweis

Im Programmentwicklungs-Zyklus EDIT-COMPILE-RUN für ein Hauptprogramm muß darauf geachtet werden, daß die temporäre EAM-Bindemoduldatei (*OMF) vor jeder Übersetzung gelöscht wird, damit durch RUN das zuletzt übersetzte Programm ausgeführt wird.

Beispiele

Aufruf mit den standardmäßig voreingestellten bzw. durch die MODIFY-COMPILE-Anweisung eingestellten Parametern:

```
//C
```

Angabe eines Dateinamens:

`//C BEISPIEL.PROG`	Dateiname bei Standardeinstellung von KIND
`//C BEISPIEL.PROG(F)`	Dateiname bei veränderter Voreinstellung von KIND=FILE

Angabe eines Paket- oder Programm-Namens:

`//C A(S)`	Spezifikation des Pakets A (*S*pezifikation)
`//C A(B)`	Implementierung des Pakets A (*B*ody)
`//C MAIN(P)`	Hauptprogramm MAIN (*P*rogram)

Angabe eines Bibliothekselements:

`//C (PLAMLIB, ELEM)`	Element ELEM aus der Bibliothek PLAMLIB
`//MC (PLAMLIB,)`	Voreinstellen der Bibliothek PLAMLIB
`//C (,ELEM)`	Element ELEM aus der voreingestellten Bibliothek PLAMLIB
`//C S-E=ELEM`	entspricht der Zeile zuvor
`//`C (BSPLIB, BSP(2.0))	Version 2.0 des Elements BSP aus der Bibliothek BSPLIB

Angabe eines EDT-Arbeitsbereichs:

`//C *E`	Aktueller Arbeitsbereich
`//C *E(5)`	Arbeitsbereich 5

Verschiedene Möglichkeiten für die Ausgabe des Compilerlistings (für SOURCE gilt der Standardwert oder eine andere Voreinstellung):

`//C ,*D`	Listing nach *DUMMY (Listing interessiert nicht)
`//C ,*SYSOUT`	Listing auf den Bildschirm
`//C ,*SYSLST`	Listing auf den Drucker
`//C ,*E`	Listing in den aktuellen EDT-Arbeitsbereich
`//C ,*E(7)`	Listing in den EDT-Arbeitsbereich 7
`//C ,LST.BSP`	Listing in die Datei LST.BSP
`//C ,(LIB, LST)`	Listing in das Element LST der Bibliothek LIB
`//MC (SOURCES,A)`	Bibliothek für den Operanden SOURCE einstellen
`//C ,(*S,LST.A)`	Listing in das Element LST.A der Bibliothek SOURCES
`//C ,(*S,*S)`	Listing in das Element A (Typ P) in der Bibliothek SOURCES
`//C ,(LSTLIB,*S)`	Listing in das Element A (Typ P) in der Bibliothek LSTLIB

Die verschiedenen Möglichkeiten für die Angabe der Objektmodulbibliothek (für SOURCE und LISTING werden die Voreinstellungen genommen):

`//C ,,(Y)`	Objektmodule in den Sternbereich. Dieser wird vorher gelöscht.
`//C ,,MODLIB`	Objektmodule in die PLAM-Bibliothek MODLIB
`//C ,,*S`	Objektmodule (Typ R) in die Quellbibliothek. Dies setzt voraus, daß im Operanden SOURCE eine Bibliothek angegeben wurde.

DEFINE-PROJECT-FILE

Projektdatei einstellen.

```
DEFINE-PROJECT-FILE

     DIRECTORY  =  dateiname
```

Funktion

DEFINE-PROJECT-FILE stellt eine Projektdatei ein, die der Pascal-XT Compiler für Zugriffe auf Paketspezifikationen benötigt. Die Einstellung gilt bis zum Verlassen des Programmiersystems, oder bis mit der DEFINE-Anweisung eine neue Projektdatei eingestellt wird. Existiert noch keine Projektdatei mit diesem Namen, dann wird eine neue Projektdatei erzeugt.

Die eingestellte Projektdatei bleibt während der ganzen Sitzung geöffnet. Sie kann von mehreren Anwendern gleichzeitig benutzt werden.

Wird während einer Sitzung die Projektdatei gewechselt, dann gibt das Programmiersystem eine Liste der vorhandenen Zuordnungen zwischen Paketen und EDT-Arbeitsbereichen aus (siehe EDIT-UNIT) und stellt die Frage, ob das Directory tatsächlich gewechselt werden soll. Bei Eingabe von "N" oder "n" wird das Kommando mit einer Meldung abgebrochen, bei Eingabe von "Y" oder "y" wird das Directory umgeschaltet und die im Programmiersystem gespeicherten Zuordnungen werden gelöscht. Die Inhalte der EDT-Arbeitsbereiche bleiben dabei erhalten. Die Abfrage entfällt, wenn keine Zuordnungen existieren. Durch dieses Verfahren bleibt die Konsistenz der Projektdatei gesichert.

Der Spin-Off-Mechanismus wird aktiviert, wenn eine nicht gültige Projektdatei eingestellt werden soll.

DIRECTORY = dateiname
: Name der Projektdatei.

EDIT-UNIT

Den Editor EDT aufrufen.

```
EDIT-UNIT | E

                    ⎧*EDT                                                          ⎫
                    ⎪name(KIND = FILE | SPEC | BODY | PROG)                        ⎪
                    ⎪                                                              ⎪
                    ⎪                                                              ⎪
      [UNIT      = ⎨(LIBRARY = dateiname,                                         ⎬]
                    ⎪,ELEMENT = element(VERSION = *HIGHEST-EXISTING | vers         ⎪
                    ⎪                    ,TYP = S | typ ))                         ⎪
                    ⎩                                                              ⎭

      [,WORKFILE = *STD | 0..9]

      [,QUERY    = YES | NO]
```

In DO-Prozeduren und Batch-Jobs wird als Versionsangabe für den UNIT-Operanden weiterhin *STD akzeptiert.

Funktion

EDIT ruft den Editor EDT auf. Der Operand UNIT legt fest, ob nur der EDT aufgerufen oder ob vor dem Aufruf eine Datei in den durch den Operanden WORKFILE eingestellten Arbeitsbereich geladen werden soll. Mit dem Operanden QUERY wird beim Verlassen des EDT das Sichern der Datei gesteuert.

Der Arbeitsbereich, der beim Verlassen des EDT eingestellt ist, wird als der aktuell eingestellte Arbeitsbereich (*STD) bezeichnet. Wird im folgenden der EDT ohne Angabe eines Arbeitsbereichs aufgerufen, dann wird in diesen Arbeitsbereich verzweigt. Wurde aber mit der MODIFY-EDIT-Anweisung ein Arbeitsbereich fest eingestellt, dann wird beim Aufruf in diesen verzweigt.

Im UNIT-Operanden kann anstelle eines Dateinamens auch der Pascal-Bezeichner eines Paketes oder Hauptprogramms angegeben werden. Die Unterscheidung zwischen Dateinamen und Pascal-Bezeichner wird durch den Operanden KIND gesteuert. Diese Arbeitsweise setzt eine Projektdatei voraus, in der für den Bezeichner ein Eintrag vorhanden ist.

Das Programmiersystem merkt sich für jeden EDT-Arbeitsbereich den Namen der Datei, deren Inhalt geladen wurde. Nun können die Arbeitsbereiche beliebig gewechselt werden und es ist sichergestellt, daß nach dem Verlassen des EDT der Inhalt des aktuellen Arbeitsbereichs in die zugehörige Datei abgespeichert wird. Diese Arbeitsweise funktioniert nur, wenn *nicht* die EDT-Kommandos READ und @READ verwendet werden, da sonst der neue Inhalt des Arbeitsbereichs zurückgeschrieben wird.

Mit dem Operanden QUERY wird festgelegt, ob das Programmiersystem nach der Rückkehr aus dem EDT den Inhalt des EDT-Arbeitsbereichs automatisch in die Datei zurückschreibt, oder ob eine Abfrage am Terminal erfolgen soll. Die Arbeitsbereiche des EDT bleiben erhalten.

Wenn Übersetzungseinheiten verändert werden, müssen diese und meist auch andere Übersetzungseinheiten desselben Programms neu übersetzt werden. Die Anweisung SHOW-ATTRIBUTES (siehe dort) gibt aus, welche Übersetzungseinheiten neu übersetzt werden müssen. Die notwendigen Neuübersetzungen (recompilations) können aber nur dann korrekt angezeigt werden, wenn Änderungen an Paket-Spezifikationen, Paket-Implementierungen und Hauptprogrammen ausschließlich unter der Kontrolle des Programmiersystems durchgeführt werden. Das Programmiersystem erkennt und speichert Modifikationen in Übersetzungseinheiten nur unter folgenden Bedingungen:

– Der EDT wird mit dem Operanden UNIT wie folgt aufgerufen:

```
UNIT = name (KIND = SPEC | BODY | PROG)
```

– Der EDT wird mit "h" verlassen und der EDT-Arbeitsbereich wird *nicht* mit einem EDT-Kommando in die Datei zurückgeschrieben.

Eine Spezifikation, Implementierung, oder ein Hauptprogramm sollte unmittelbar nach einer Änderung übersetzt werden, damit mögliche Änderungen in den Paketbeziehungen (WITH-Klauseln) in der Projektdatei vermerkt werden können.

Die Standardwerte der Operanden können während einer Sitzung mit der MODIFY-EDIT-Anweisung verändert werden. Diese Änderungen gelten bis zur nächsten Änderung, längstens bis zum Beenden des Programmiersystems.

Die Ausgaben in verschiedenen Fehlersituationen sind weiter unten beschrieben.

Der Spin-Off-Mechanismus wird nach folgenden Fehlern aktiviert:

(a) Fehler in der Syntax der EDIT-Anweisung

(b) Das gewünschte Paket bzw. Programm kann nicht geladen werden, da keine Projektdatei eingestellt ist

(c) Im UNIT-Operanden wurde der Name eines Paketes oder Programms angegeben, für das aber kein Eintrag in der Projektdatei existiert

(d) Die Datei existiert nicht oder kann nicht eröffnet werden

UNIT Dieser Operand legt fest, ob lediglich der EDT aufgerufen oder vor dem Aufruf eine Datei in den Arbeitsbereich geladen werden soll.

= *EDT Es wird in den aktuell eingestellten Arbeitsbereich des EDT verzweigt. Der Inhalt des Arbeitsbereichs wird nicht verändert.

= name (KIND = FILE | SPEC | BODY | PROG)

"name" ist der Name einer Datei, eines Pakets oder eines Hauptprogramms, das in den eingestellten EDT-Arbeitsbereich geladen wird. Vor dem Laden wird dieser Bereich gelöscht.
Der Operand KIND legt fest, wie "name" zu interpretieren ist:
= FILE als Name einer BS2000-Datei
= SPEC als Pascal-Bezeichner einer Paket-Spezifiaktion
= BODY als Pascal-Bezeichner einer Paket-Implementierung
= PROG als Pascal-Bezeichner eines Hauptprogramms.
Die Angabe von SPEC, BODY oder PROG setzt voraus, daß eine Projektdatei eingestellt ist und für die angegebene Übersetzungseinheit ein Eintrag existiert, ansonsten wird ein Fehler gemeldet (siehe unten). Die Pascalquelle wird aus der Datei gelesen, die in der Projektdatei angegeben ist. Der Name dieser Datei wird über der EDT-Kommandozeile eingeblendet. Unterstriche im Pascal-Bezeichner müssen in "name" als Bindestriche ("-") angegeben werden.

= (dateiname, element (vers, typ))

Der aktuelle EDT-Arbeitsbereich wird gelöscht und das Bibliothekselement "element" mit der angegebenen Version "vers" und dem Typ "typ" wird aus der PLAM-Bibliothek "dateiname" in den EDT-Arbeitsbereich geladen. Beim Verlassen des EDT wird immer die Version des Bibliothekselements überschrieben, die gelesen wurde.

Die Angaben für "vers" und "typ" sind optional.
Standardwert für "vers" = *HIGHEST-EXISTING,
für "typ" = "S".

Die Versionsangabe *HIGHEST-EXISTING bewirkt:

- wenn das angegebene Bibliothekselement existiert, wird die höchste Version des Bibliothekselements gelesen und beim Verlassen des EDT überschrieben. Ab Pascal-XT V2.2A wird also nicht mehr, wie früher bei der Versionsangabe *STD, beim Zurückschreiben automatisch eine neue Version angelegt.
- wenn das angegebene Bibliothekselement nicht existiert, wird ein Element mit Version "001" angelegt.

Die Versionsangabe *STD ist im Dialog nicht mehr möglich. In DO-Prozeduren und Batch-Jobs kann weiterhin *STD verwendet werden, jedoch mit geänderter Bedeutung:

- wenn das angegebene Bibliothekselement existiert, wird die höchste Version des Bibliothekselements gelesen und beim Verlassen des EDT überschrieben.

– wenn das angegebene Bibliothekselement nicht existiert, wird ein Element mit Version *UPPER-LIMIT (höchstmögliche Version) angelegt.

Die Versionsangabe *INCREMENT ist nicht möglich.

WORKFILE Mit diesem Operanden wird der EDT-Arbeitsbereich eingestellt.

= *STD Bezeichnet den aktuell eingestellten Arbeitsbereich.

= 0..9 Nummer des EDT-Arbeitsbereichs.

QUERY Nach dem Verlassen des EDT kann eine Abfrage erfolgen, ob der Inhalt des Arbeitsbereichs in die Datei bzw. das Bibliothekselement zurückgeschrieben werden soll, aus der zuvor geladen wurde. Standardmäßig (YES) erfolgt die Abfrage. Die Abfrage und das Zurückschreiben entfallen, wenn
– der EDT-Arbeitsbereich leer ist oder
– im Programmiersystem keine Zuordnung des Arbeitsbereiches zu einer Datei vermerkt ist (z.B. nach einem Wechsel des Arbeitsbereichs im EDT).

= YES Es erfolgt eine Abfrage. Nach Eingabe von "Y" oder "y" wird zurückgeschrieben, nach Eingabe von "N" oder "n" wird nicht geschrieben.

= NO Es erfolgt keine Abfrage und der Inhalt des Arbeitsbereichs wird in die Datei geschrieben.

Fehlermeldungen

Treten beim Laden einer Datei in den EDT Fehler auf, dann gibt in Abhängigkeit der Fehlerart entweder das Programmiersystem eine Meldung (mit dem Präfix "`>>>`") aus oder es wird in der Meldezeile des EDT (unmittelbar über der Kommandozeile) eine Meldung mit einem Fehlercode ausgegeben. Die Fehlercodes err-nr sind im Abschnitt 10.4 beschrieben.

(a) Datei existiert nicht. Es wird in den EDT verzweigt und in der Meldezeile

```
FILE "name" DOES NOT EXIST
```

ausgegeben. Ist der EDT-Arbeitsbereich nach Verlassen des EDT nicht leer, dann wird bei QUERY=NO die Datei "name" erzeugt und der Inhalt des Arbeitsbereichs darin abgespeichert. Bei QUERY=YES erfolgt die Abfrage, ob die Datei erzeugt werden soll.

(b) Datei existiert, kann aber nicht zum Lesen eröffnet werden. Es wird die Meldung

```
>>> OPEN ERROR ON SPECIFIED FILE (err-nr)
```

ausgegeben und nicht in den EDT verzweigt.

(c) Das angegebene Bibliothekselement existiert nicht. Es wird in den EDT verzweigt und dort die Meldung

```
ELEMENT "elem (version,typ)" DOES NOT EXIST
```

ausgegeben. Nach Verlassen des EDT wird wie bei (a) verfahren.

(d) Die angegebene Bibliothek kann nicht eröffnet werden. Es wird in den EDT verzweigt und dort die Meldung

```
OPEN ERROR ON LIBRARY "dateiname"
```

ausgegeben. Nach Verlassen des EDT wird wie bei (a) verfahren, wenn die Bibliothek nicht existiert hat, bei gesperrter Bibliothek wird abgebrochen.

(e) Der Name ist in der Projektdatei nicht eingetragen. Es wird die Meldung

```
>>> UNIT NOT FOUND IN PROJECT DIRECTORY
```

ausgegeben und nicht in den EDT verzweigt.

(f) Die in der Projektdatei angegebene Datei kann nicht eröffnet werden. Es wird die Meldung

```
>>> OPEN ERROR ON FILE FOUND IN PROJECT DIRECTORY (err-nr)
```

ausgegeben und nicht in den EDT verzweigt.

Beispiele

Aufruf mit den standardmäßig voreingestellten bzw. durch die MODIFY-EDIT-Anweisung eingestellten Parametern:

```
//E
```

Editieren eines EDT-Arbeitsbereichs:

`//E *E`	Aktueller Arbeitsbereich
`//E *E,5`	Arbeitsbereich 5
`//E ,5`	Arbeitsbereich 5

Editieren einer Datei:

`//E A.PROG`	Datei A.PROG
`//E A.PROG(F)`	Datei A.PROG mit Zusatz KIND=FILE

Angabe eines Paket- oder Programm-Namens:

`//E BSP(S)`	Spezifikation des Pakets BSP (*S*pezifikation)
`//E BSP(B)`	Implementierung des Pakets BSP (*B*ody)
`//E BEISPIEL(P)`	Hauptprogramm BEISPIEL (*P*rogram)

Editieren eines Bibliothekselements:

`//E (TOOLLIB, LMS.PROG)`	Bibliothek TOOLLIB und Element LMS.PROG
`//ME (TOOLLIB, )`	Bibliothek TOOLLIB fest einstellen
`//E (,LMS.PROG)`	LMS.PROG aus der ingestellten Bibliothek TOOLLIB
`//E (LIB, TOOLS.S (V1.0))`	Version V1.0 des Elements TOOLS.S aus der Bibliothek LIB
`//E (TOOLLIB,LMS (,R))`	Element LMS mit der höchsten Version (*HIGHEST-EXISTING) und Typ R

Abschalten der Rückfrage vor dem Überschreiben:

`//E ,,N`	oder
`//ME ,,N`	Einstellung als neuer Standardwert

END

Programmiersystem beenden.

END

Funktion

END beendet das Pascal-XT-Programmiersystem. Alle noch geöffneten Dateien werden geschlossen. Eine beim Aufruf des Programmiersystems angegebene Jobvariable zur Programmüberwachung wird besetzt.

Trat bei der Analyse oder Ausführung einer Anweisung ein Fehler auf, dann wird nach dem Verlassen des Programmiersystems vom Betriebssystem die Meldung

```
ABNORMAL PROGRAM TERMINATION
```

ausgegeben. Im Stapelbetrieb wird anschließend zum nächsten Jobstep verzweigt.

MODIFY-COMPILE

Operanden-Standardwerte der COMPILE-Anweisung temporär ändern.

```
MODIFY-COMPILE | MC

                  ┌*EDT (WORKFILE = *STD | 0..9)                                  ┐
                  │name (KIND = FILE | SPEC | BODY | PROG)                        │
[ SOURCE      = ──┤                                                               ├─]
                  │(SOURCE-LIBRARY = dateiname                                    │
                  │,SOURCE-ELEMENT = element(VERSION = ┌*HIGHEST-EXISTING┐))    │
                  └                                    └vers             ┘       ┘

                  ┌*SYSOUT                                  ┐
                  │*SYSLST                                  │
                  │*EDT (WORKFILE = *STD | 0..9)            │
[,LISTING     = ──┤                                         ├─]
                  │*DUMMY                                   │
                  │(LIST-LIBRARY = *STD | dateiname         │
                  │,LIST-ELEMENT = *STD | element)          │
                  └                                         ┘

[,MODULE-LIBRARY =  *OMF (ERASE = YES | NO) | dateiname | *STD ]
[,ASSEMBLER      =  *BY-SOURCE | ON | OFF ]
[,CHECK          =  *BY-SOURCE | ON | OFF ]
[,DEBUG          =  *BY-SOURCE | ON | OFF | RESTRICTED ]
[,GENERATE       =  *BY-SOURCE | ON | OFF ]
[,INITIALIZE     =  *BY-SOURCE | ON | OFF ]
[,LIST | L       =  *BY-SOURCE | ON | OFF ]
[,MAP            =  *BY-SOURCE | ON | OFF ]
[,OPTIMIZE       =  *BY-SOURCE | ON | OFF ]
[,STANDARD       =  *BY-SOURCE | ON | L0 | OFF ]
[,XREF           =  *BY-SOURCE | ON | OFF ]
[,LINES-PER-PAGE =  *STD | 11..2147483639 ]
[,MESSAGE-LEVEL  =  NOTES | WARNINGS | ERRORS ]
```

In DO-Prozeduren und Batch-Jobs wird für den SOURCE-Operanden weiterhin die Angabe *STD akzeptiert.

Funktion

Die MODIFY-COMPILE-Anweisung hat denselben Aufbau wie die COMPILE-Anweisung. Die Beschreibung der Operanden ist dort nachzulesen.

Mit der MODIFY-COMPILE-Anweisung können die Operanden-Standardwerte der COMPILE-Anweisung verändert werden. Diese Änderung gilt bis zur nächsten Änderung, längstens jedoch bis zum Verlassen des Programmiersystems. In den Menüs der COMPILE- und MODIFY-COMPILE-Anweisungen werden stets die aktuellen Standardwerte angezeigt.

Bei Bibliothekselementen können Bibliotheks- und Elementname unabhängig voneinan-

der eingestellt werden. Bei Änderung nur eines Operanden behält der andere seinen bisherigen Wert. Die Version beim Operanden SOURCE-ELEMENT kann immer nur zusammen mit dem Elementnamen eingestellt werden.

Der Spin-Off-Mechanismus wird aktiviert, wenn die Eingabe der MODIFY-COMPILE-Anweisung syntaktisch falsch ist.

Hinweis

Der Operandenwert *UNCHANGED, der im Menü gezeigt wird, ist im Abschnitt 2.6.1 beschrieben.

MODIFY-EDIT

Operanden-Standardwerte der EDIT-Anweisung ändern.

```
MODIFY-EDIT | ME

                    ┌*EDT                                                          ┐
                    │name(KIND = FILE | SPEC | BODY | PROG)                        │
                    │                                                              │
                    │(LIBRARY = dateiname,                                         │
      [ UNIT     = {│,ELEMENT = element(VERSION = *HIGHEST-EXISTING | vers        │}]
                    │                    ,TYP = S | typ ))                         │
                    └                                                              ┘

      [,WORKFILE = *STD | 0..9]

      [,QUERY    = YES | NO]
```

In DO-Prozeduren und Batch-Jobs wird für den UNIT-Operanden weiterhin *STD akzeptiert.

Funktion

Die MODIFY-EDIT-Anweisung hat denselben Aufbau wie die EDIT-UNIT-Anweisung. Die Beschreibung der Operanden ist dort nachzulesen.

Mit dieser Anweisung können die Operanden-Standardwerte der EDIT-Anweisung verändert werden. Diese Änderung gilt bis zur nächsten Änderung, längstens bis zum Verlassen des Programmiersystems. In den Menüs der EDIT- und MODIFY-EDIT-Anweisung werden stets die aktuellen Standardwerte angezeigt.

Bei Bibliothekselementen können Bibliotheks- und Elementname unabhängig voneinander eingestellt werden. Bei Änderung nur eines Operanden behält der andere seinen bisherigen Wert. Version und Typ beim Operanden ELEMENT können immer nur zusammen mit dem Elementnamen eingestellt werden.

Hinweis

Der Operandenwert *UNCHANGED, der im Menü gezeigt wird, ist im Abschnitt 2.6.1 beschrieben.

REMOVE-DIRECTORY-ENTRY

Eintrag in der aktuell eingestellten Projektdatei löschen.

```
REMOVE-DIRECTORY-ENTRY

     UNIT =  pascal-name
```

Funktion

REMOVE löscht in der aktuell eingestellten Projektdatei alle Einträge für den im Operanden UNIT angegebenen Bezeichner. Die zugehörigen Quelldateien werden durch diese Funktion nicht verändert.

Durch das Löschen eines Eintrags werden alle Übersetzungseinheiten ungültig, die das ausgetragene Paket importieren. Sie werden in der Projektdatei entsprechend gekennzeichnet.

Der Spin-Off-Mechanismus wird aktiviert, wenn

(a) keine Projektdatei eingestellt ist

(b) der Name des Pakets bzw. Hauptprogramms nicht in der Projektdatei existiert

UNIT = pascal-name

Name des Pakets oder Hauptprogramms, dessen Eintrag gelöscht werden soll.

RUN-PROGRAM

Tool oder Programm mit oder ohne Pascal-Testhilfe ausführen.

```
RUN-PROGRAM | R

                      ┌*LAST-COMPILED-PROG                  ┐
                      │tool-name                            │
   [ PROGRAM      = { ┤                                     }]
                      │(LIBRARY = *OMF | dateiname          │
                      └,ELEMENT = element)                  ┘

   [,PARAMETER    = *NONE | string ]

   [,DEBUG        = NO | YES ]
```

Funktion

RUN führt ein Pascal-XT-Programm oder ein Tool aus, ohne das Programmiersystem zu verlassen. Nach Beendigung des Programms wird das Programmiersystem fortgesetzt. Vor Ausführung des Programms können den Programmparametern aktuelle BS2000-Dateien zugewiesen werden (siehe Kapitel 5). Im Operanden DEBUG wird angegeben, ob das Programm mit der Pascal-Testhilfe PATH getestet werden soll (siehe Kapitel 9).

Ab Pascal-XT V2.2A dürfen mit RUN keine Pascal-XT-Programme mit Fremdsprachanschluß ausgeführt werden, da dies mit ILCS (siehe 7.1) unverträglich ist.

Das mit RUN gestartete Programm bleibt bis zur nächsten Übersetzung geladen. Damit kann ein Programm mehrfach nacheinander ausgeführt werden, ohne daß es jedesmal geladen werden muß.

Beim Auftreten einer nicht behandelten Ausnahme (Laufzeitfehler) im Programm wird die dynamische Aufrufkette (siehe 10.2) ausgegeben, das Programm beendet und in das Programmiersystem zurückgekehrt.

Ein Programm wird nicht ausgeführt, wenn bereits ein Tool geladen ist, das denselben Programm-Namen wie das Programm besitzt. Es wird eine Fehlermeldung ausgegeben und der Spin-Off-Mechanismus aktiviert. Benötigt das auszuführende Programm ein Paket (oder auch mehrere), das bereits von einem geladenen Tool geladen und benutzt wird, dann kann die Ausführung des Programms wegen des mehrfach verwendeten Pakets zu Fehlern führen.

Der Spin-Off-Mechanismus wird aktiviert, wenn das Programm nicht geladen werden kann, oder das Programm mit einem Fehler beendet wird.

PROGRAM Bezeichnet das auszuführende Pascal-Programm. Es wird mit dem DLL geladen, sofern dies nicht schon zuvor erfolgt ist. Externreferenzen werden entsprechend dem Autolink-Mechanismus des DLL aufgelöst (siehe auch 6.3).

= *LAST-COMPILED-PROG
Das zuletzt übersetzte Pascal-Programm wird aus der in der COMPILE-Anweisung angegebenen Bibliothek geladen und ausgeführt.

= tool-name
Name des auszuführenden Tools. Das Tool muß zuvor mit der Anweisung ADD-TOOL geladen worden sein.

= (*OMF,element)
Das mit "element" bezeichnete Programm wird aus der temporären Bindemoduldatei (*OMF) geladen und ausgeführt. "element" ist der Name des Starter-Moduls des Programms. *OMF muß vor jeder Übersetzung mit ERASE * gelöscht werden, wenn ein Programm mehrfach übersetzt wird und immer das zuletzt übersetzte ausgeführt werden soll.

= (dateiname,element)
Aus der PLAM-Bibliothek "dateiname" wird das mit "element" bezeichnete Programm geladen und ausgeführt. "element" ist der name des Starter-Moduls des Programms.

Hinweis
Der DLL kann erst ab Version 8 aus PLAM-Bibliotheken laden.

PARAMETER Beschreibt die Zuordnung von BS2000-Dateien zu externen Dateien (Programmparameter) des Programms.

= *NONE
Es werden keine aktuellen Dateien angegeben. Wurde das Programm unmittelbar vorher schon einmal mit Angabe von aktuellen Dateien ausgeführt, dann werden diese Dateien bei folgenden Aufrufen als Standardwerte angenommen.

= string
Die Zuordnungen von BS2000-Dateien zu den externen Pascal-Dateien werden in einem String eingegeben, der in Hochkommas '...' einzuschließen ist. Jede einzelne Dateizuordnung ist in der Form

```
pascal-datei = aktuelle-datei
```

anzugeben. Leerzeichen werden überlesen. Mehrere Zuordnungen sind innerhalb des Strings durch Kommas zu trennen:

```
'pas-dat1 = dat1, pas-dat2 = dat2, ...'
```

Die richtige Anzahl der Parameter und die Gültigkeit der Pascal-Dateibezeichner können nicht überprüft werden. Ein Fehler wird erst zur Laufzeit entdeckt.
Das Programmiersystem merkt sich diese Dateizuordnungen, solange das Programm geladen ist. Nach einer Übersetzung oder nach Ausführung eines anderen Programms sind die Zuweisungen nicht mehr verfügbar. Für die permanent geladenen Tools bleiben die Zuweisungen bis zum Verlassen des Programmiersystems gespeichert. Werden bei einem folgenden Aufruf der RUN-Anweisung andere aktuelle Dateien angegeben, dann ersetzen diese die zuvor abgespeicherten Werte. Für nicht angegebene Zuordnungen werden, sofern vorhanden, die zuvor angegebenen übernommen.
Als "aktuelle-datei" kann eine SAM- oder ISAM-Datei angegeben werden. Für ISAM-Dateien sind die im Abschnitt 5.2.2 gemachten Anmerkungen zu beachten.
Den vordefinierten Dateien INPUT und OUTPUT können keine aktuelle Dateien zugewiesen werden. Sie sind fest den Systemdateien SYSDTA bzw. SYSOUT zugeordnet. Für sie werden deshalb auch keine Zuordnungen abgespeichert.

DEBUG Legt fest, ob das Programm unter der Kontrolle der Testhilfe PATH ablaufen soll.

= NO Ausführung ohne Testhilfe.

= YES Ausführung mit Testhilfe.

Hinweis

Programme mit großem Hauptspeicherbedarf sollten statisch gebunden und außerhalb des Programmiersystems ausgeführt werden, da dann zusätzlich der sonst vom Programmiersystem und Compiler benötigte Hauptspeicher zur Verfügung steht.

Beispiele

`//R` Ausführen des zuletzt übersetzten Programms

`//R LMS` Ausführen des Tools mit dem Namen LMS (muß zuvor mit der ADD-TOOL-Anweisung geladen worden sein)

`//R (,TEST)`
Ausführen des Programms TEST aus dem *-Bereich (*OMF) (Achtung: evtl. benötigte Pakete können vom DLL nicht aus *OMF nachgeladen werden. Sie müssen in einer Bindemodulbibliothek enthalten sein, die über SYSFILE TASKLIB eingestellt werden muß).

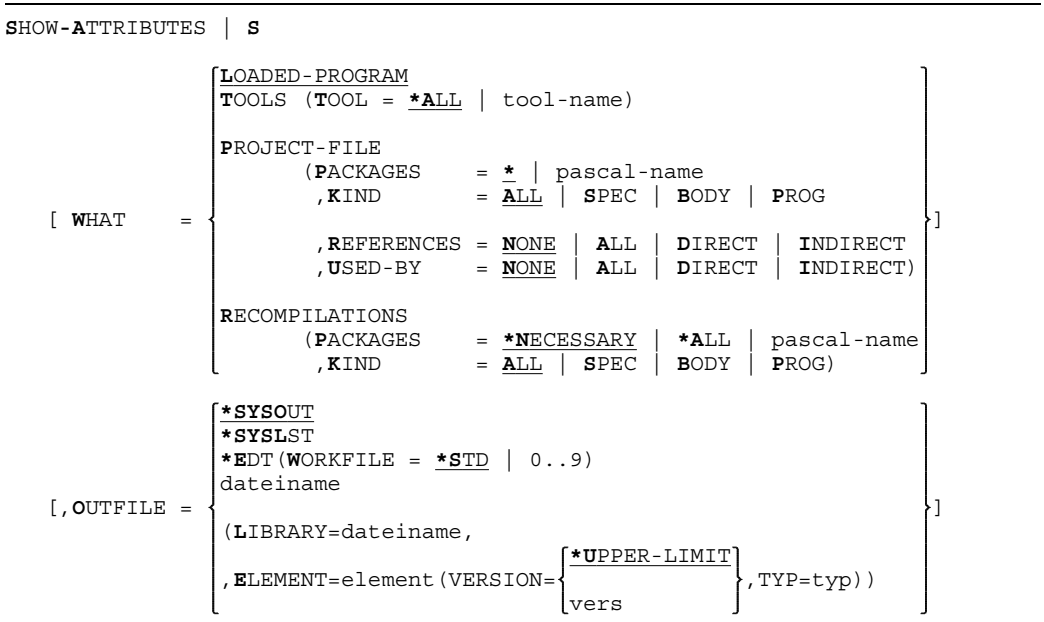
`//R (MODLIB, BEISPIE)`
Ausführen des Programms BEISPIE (Name des Starter-Moduls) aus der Bindemodulbiblitohek MODLIB

`//R d=y` Ausführen des zuletzt übersetzten Programms unter Kontrolle der Testhilfe PATH

`//R ,'SRC=QUELLE,DEST=ZIEL'`
Ausführen des zuletzt übersetzten Programms mit Zuordung der BS2000-Dateien QUELLE und ZIEL an die Pascal-Dateien SRC bzw. DEST

SHOW-ATTRIBUTES

Informationen über das Programmiersystem und die aktuell eingestellte Projektdatei ausgeben.



In DO-Prozeduren und Batch-Jobs wird als Versionsangabe für den OUTFILE-Operanden weiterhin *STD akzeptiert.

Funktion

SHOW-ATTRIBUTES liefert Informationen über das zuletzt ausgeführte Programm, vorhandene Tools, die Projektdatei und neu zu übersetzende Übersetzungseinheiten. Die Informationen werden standardmäßig in die Systemdatei SYSOUT ausgegeben, können aber in eine beliebige andere Datei umgelenkt werden. Das Ausgabeformat der SHOW-Anweisung ist weiter unten beschrieben.

Durch die Angabe von WHAT=*LOADED-PROGRAM bzw. TOOLS wird für das zuletzt ausgeführte Programm bzw. die vorhandenen Tools der Programmname und bereits angegebene Zuordnungen von Dateien zu Programmparametern ausgegeben (siehe auch RUN-Anweisung).

Durch Angabe von WHAT=PROJECT werden die in der aktuell eingestellten Projektdatei gespeicherten Informationen über die Übersetzungseinheiten ausgegeben. Standardmäßig werden der Paket- bzw. Programm-Name und der Dateiname, in der die Einheit abgespeichert ist, ausgegeben. Zusätzlich können

(a) alle direkt und indirekt importierten Paket-Spezifikationen

(b) alle Übersetzungseinheiten, die das angegebene Paket direkt oder indirekt importieren,

ausgegeben werden.

Die Informationen von (b) sind von besonderem Interesse, da alle von einer Änderung einer Spezifikation betroffenen Übersetzungseinheiten angegeben werden.

Durch Angabe von RECOMPILATIONS werden COMPILE-Anweisungen zur Rekompilierung von Übersetzungseinheiten erzeugt. Eine Rekompilierung kann aus folgenden Gründen notwendig werden:

(a) Eine Übersetzungseinheit wurde editiert

(b) Die Spezifikation eines importierten Pakets wurde verändert oder nochmals übersetzt (ohne daß sie geändert wurde)

(c) Bei der Übersetzung der Übersetzungseinheit wurden Fehler gemeldet

(d) Bei der Übersetzung einer Paket-Implementierung oder eines Hauptprogramms wurde die Option GENERATE=OFF angegeben.

Standardmäßig werden die COMPILE-Anweisungen nur für die Übersetzungseinheiten erzeugt, die nochmals übersetzt werden müssen, damit alle zu einem Programm gehörigen Übersetzungseinheiten in einem konsistenten Zustand vorliegen. Es können auch COMPILE-Anweisungen zur Übersetzung aller Pakete erzeugt werden.

Die COMPILE-Anweisungen werden so angeordnet, daß die Übersetzungen in der richtigen Reihenfolge durchgeführt werden. Das setzt voraus, daß Übersetzungseinheiten ausschließlich über die EDIT-Anweisung verändert werden. Nach Änderungen der WITH-Klausel (Liste der importierten Pakete) sollte die Übersetzungseinheit übersetzt werden, damit die neuen Paketbeziehungen in der Projektdatei aktualisiert werden können. Wird die Übersetzung nicht durchgeführt, dann werden zur Bestimmung der Übersetzungsreihenfolge die vor der Änderung gültigen Paketbeziehungen angenommen. Dies kann zu einer fehlerhaften Kompilierungsreihenfolge führen.

Der Spin-Off-Mechanismus wird nach folgenden Fehlern aktiviert ((b) bis (c) nur bei Angabe von WHAT=PROJECT oder WHAT=RECOMPILATIONS):

(a) Fehler in der Syntax der SHOW-Anweisung

(b) Die Projektdatei war nicht eingestellt

(c) Es wurde der Name eines Paketes oder Programms angegeben, für das kein Eintrag in der Projektdatei existiert

WHAT Spezifiziert die gewünschten Informationen.

= LOADED PROGRAMM
Gibt den Namen des zuletzt mit RUN ausgeführten Programms aus. Wurden bei der RUN-Anweisung Dateizuordnungen angegeben, dann werden diese mit ausgegeben.

= TOOLS (*ALL | tool-name)
Standardmäßig (*ALL) werden die Namen aller Tools ausgegeben. Für jedes Tool werden die Zuweisungen von aktuellen Dateien zu den Programmparametern mit angegeben, wenn das Tool bereits ausgeführt wurde (siehe RUN-Anweisung). Bei Angabe von "tool-name" werden die Informationen nur von diesem Tool ausgegeben.

= PROJECT-FILE (PACKAGES=.., KIND=.., REFERENCES=.., USED-BY=...)
Durch Angabe dieses Operanden werden die Informationen aus der Projektdatei ausgegeben. Der Umfang der Ausgabe wird durch die zusätzlichen Operanden gesteuert. Standardmäßig werden die Namen aller Übersetzungseinheiten (PACKAGE NAME), die Art der Übersetzungseinheiten (KIND) und die Dateinamen (LOCATION), unter denen sie abgespeichert sind, ausgegeben. Neu zu übersetzende Pakete sind vor der Angabe der Art der Übersetzungseinheit (KIND) mit einem "!" gekennzeichnet.

PACKAGES = * | [*] pascal-name [*]

Legt die Namen der Pakete fest, für welche Informationen auszugeben sind. Standardmäßig ("*") werden alle Pakete betrachtet. Bei Angabe von "pascal-name" werden nur die Informationen zu diesem Paket bzw. Hauptprogramm ausgegeben.
Es kann auch das Wildcard-Zeichen "*" angegeben werden, das für eine beliebige (auch leere) Zeichenfolge steht. "*pascal-name" liefert alle Einträge, die auf der in "pascal-name" angegebenen Zeichenfolge enden, "pascal-name*" alle, die die mit der Zeichenfolge beginnen und "*pascal-name*" alle, die die Zeichenfolge enthalten.

KIND = ALL | SPEC | BODY | PROG

Mit diesem Operanden kann die Ausgabe auf Spezifikationen (SPEC), oder Implementierungen (BODY), oder Hauptprogramme (PROG) der in PACKAGE angegebenen Namen beschränkt werden. Standardmäßig (ALL) werden Informationen zu allen Übersetzungseinheiten ausgegeben.

REFERENCES = NONE | ALL | DIRECT | INDIRECT

Für jedes durch PACKAGE und KIND angegebene Paket können alle direkt und/oder indirekt importierten Pakete ausgegeben werden. Standardmäßig (NONE) werden die Paketbeziehungen nicht ausgegeben.
Durch die Angabe von ALL werden alle direkt und indirekt importierten Pakete ausgegeben.
Durch die Angabe von DIRECT werden nur die direkt importierten Pakete ausgegeben.
Durch die Angabe von INDIRECT werden nur die indirekt importierten Pakete ausgegeben. Dies sind all die Pakete, die von den direkt importierten Paket-Spezifikationen direkt oder indirekt importiert werden. Importiert eine Übersetzungseinheit keine Pakete, dann wird dafür "--" ausgegegeben. Wurde die Übersetzungseinheit modifiziert und noch nicht übersetzt, dann wird "- undefined -" ausgegeben.

USED-BY = NONE | ALL | DIRECT | INDIRECT

Für jedes durch PACKAGE und KIND angegebene Paket können alle Pakete ausgegeben werden, die dieses Paket direkt oder indirekt importieren. Dies sind also alle Pakete, die bei Änderung des angegebenen Paketes neu übersetzt werden müssen. Dies entspricht damit der umgekehrten Relation wie bei REFERENCES.
Standardmäßig (NONE) werden diese Informationen nicht ausgegeben.
Durch die Angabe von ALL werden alle Pakete ausgegeben, die dieses Paket direkt oder indirekt importieren.
Durch die Angabe von DIRECT werden alle Pakete ausgegeben, die dieses Paket direkt importieren, d.h. die dieses Paket in ihrer WITH-Klausel angegeben haben.
Durch die Angabe von INDIRECT werden die Pakete ausgegeben, die das angegebenen Paket indirekt importieren.

= RECOMPILATIONS (PACKAGES=.., KIND=..)

Dieser Operand steuert die Erzeugung von COMPILE-Anweisungen für Übersetzungseinheiten. Mit den Operanden PACKAGES und KIND werden die Übersetzungseinheiten ausgewählt, für die COMPILE-Anweisungen zu erzeugen sind.
In den erzeugten COMPILE-Anweisungen werden nur die SOURCE-Operanden (Name der Übersetzungseinheit und die KIND) angegeben. Die Anweisungen können, sofern sie in eine Datei ausgegeben wurden, mit der CALL-Anweisung zur Ausführung gebracht werden (für die nicht angegebenen COMPILE-Operanden werden die aktuell gültigen Werte der COMPILE-Anweisung angenommen).
Die Reihenfolge der Anweisungen stellt eine gültige Übersetzungsreihenfolge dar (siehe Anmerkungen weiter oben).

PACKAGES = *NECESSARY | *ALL | pascal-name

Standardmäßig (*NECESSARY) werden COMPILE-Anweisungen für modifizierte und davon abhängige Übersetzungseinheiten ausgegeben.
Durch Angabe von *ALL werden COMPILE-Anweisungen für alle Übersetzungseinheiten erzeugt, bei Angabe von "pascal-name" nur die genannte Übersetzungseinheit und die davon abhängigen Übersetzungseinheiten.

KIND = ALL | SPEC | BODY | PROG

Standardmäßig (ALL) werden alle Übersetzungseinheiten daraufhin untersucht, ob sie neu übersetzt werden müssen. Durch die Angabe von KIND kann die Überprüfung auf Spezifikationen (SPEC), oder Implementierungen (BODY), oder Hauptprogramme (PROG) der in PACKAGES spezifizierten Namen beschränkt werden.

OUTFILE Die Ausgaben werden in die spezifizierte Datei ausgegeben.

= *SYSOUT
Ausgabe auf die Systemdatei SYSOUT

= *SYSLST
Ausgabe auf die Systemdatei SYSLST

= *EDT (WORKFILE = *STD | 0..9)
Ausgabe in den angegebenen Arbeitsbereich des EDT. Standardmäßig (*STD) ist dies der aktuell eingestellte Bereich.

= dateiname
Name der Ausgabedatei

= (dateiname, element (vers, typ))
Die Ausgabe erfolgt in das Bibliothekselement "element" der PLAM-Bibliothek "dateiname". Der Typ des Bibliothekselementes ist standardmäßig "P" (für Printfile). Soll das erzeugte Element mit der CALL-Anweisung ausgeführt werden, z.B. wenn es COMPILE-Anweisungen enthält, dann muß explizit der Typ "J" (für Job Control) angegeben werden. Die Version des Elements ist standardmäßig *UPPER-LIMIT (höchstmögliche Version). Existiert bereits ein Element mit demselben Namen, derselben Version und vom selben Typ, dann wird es überschrieben.

Die Versionsangabe *STD ist ab Pascal-XT V2.2A im Dialog nicht mehr möglich. In DO-Prozeduren und Batch-Jobs wird *STD weiterhin akzeptiert.

Ausgabeformat der SHOW-Anweisung für die Projektdatei

Das Ausgabeformat wird anhand des Beispiels aus Kapitel 3 erläutert.

```
//D TEST.DIRECTORY
//S P(,,ALL,ALL)  ──────────────────────────────────────────────────────── (01)


     CONTENTS OF THE PROJECT DIRECTORY FILE   (TEST.DIRECTORY)

PACKAGE NAME    KIND  LOCATION

A              (SPEC) ($USERID.PLAM.SPEC,A.SPEC(*UPPER-LIMIT,S)) ──────── (02)
    REFERENCES ──────────────────────────────────────────────────────────── (03)
      direct:   - -  ──────────────────────────────────────────────────────── (04)
      indirect: - -
    USED BY ─────────────────────────────────────────────────────────────── (05)
      direct:   A (OWN BODY), B (SPEC) ──────────────────────────────────── (06)
      indirect: B (BODY), C (SPEC), C (BODY)

A              (BODY) $USERID.A.BODY
    REFERENCES
      direct:   A (OWN SPEC)
      indirect: - -

B             !(SPEC) ($USERID.PLAM.SPEC,B.SPEC(*UPPER-LIMIT,S)) ──────── (07)
    REFERENCES
      direct:   - undefined - ──────────────────────────────────────────── (08)
      indirect: - undefined -

B             !(BODY) ($USERID.PLAM.BODY,B.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   B (OWN SPEC)
      indirect: A

C             !(SPEC) ($USERID.PLAM.SPEC,C.SPEC(*UPPER-LIMIT,S))
    REFERENCES
      direct:   B
      indirect: A
    USED BY
      direct:   C (OWN BODY)
      indirect: - -

C             !(BODY) ($USERID.PLAM.BODY,C.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   C (OWN SPEC), D
      indirect: A, B
```

```
D              (SPEC) ($USERID.PLAM.SPEC,D.SPEC(*UPPER-LIMIT,S))
    REFERENCES
      direct:   - -
      indirect: - -
    USED BY
      direct:   C (BODY), D (OWN BODY)
      indirect: - -

D              (BODY) ($USERID.PLAM.BODY,D.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   D (OWN SPEC)
      indirect: - -
```

(01) Mit der SHOW-Anweisung wird der Inhalt der Projektdatei einschließlich der Beziehungen zwischen den Paketen ausgegeben. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = PROJECT-FILE (REFERENCES = ALL, USED-BY = ALL)
```

(02) Für jeden Eintrag in der Projektdatei wird der Paket- bzw. Programmname ausgegeben, gefolgt von der Angabe, ob es sich um eine Paketspezifikation (SPEC), eine Paketimplementierung (BODY) oder um ein Hauptprogramm (PROG) handelt und schließlich dem Namen der zugehörigen Quelldatei. Bei Bibliothekselementen wird der Bibliotheksname, der Elementname, die Version und der Typ des Elements ausgegeben. Steht die Quelle in einer Datei, dann wird der entsprechende BS2000-Dateiname ausgegeben.

(03) Unter der Überschrift REFERENCES werden die von diesem Paket oder Hauptprogramm importierten Pakete aufgeführt. Als direkt importiert werden jene Pakete betrachtet, die direkt in der WITH-Klausel angegeben wurden, bei Paketimplementierungen außerdem noch die eigene Spezifikation (OWN SPEC). Indirekt importiert werden all jene Pakete, die von den Spezifikationen der importierten Pakete importiert werden (auch über mehrere Stufen hinweg).

(04) Zwei aufeinanderfolgende Minuszeichen ("--") bedeuten, daß keine Pakete importiert werden bzw. daß dieses Paket nicht von anderen Übersetzungseinheiten importiert wird.

(05) Unter der Überschrift USED-BY werden diejenigen Übersetzungseinheiten aufgelistet, die dieses Paket importieren. Für direkt und indirekt gelten sinngemäß die bei REFERENCES gemachten Aussagen. Diese Ausgabe erfolgt nur bei Spezifikationen, da Paketimplementierungen und Hauptprogramme nicht von anderen Paketen importiert werden können.

(06) Bezieht sich eine Angabe auf die eigene Spezifikation oder die eigene Implementierung, dann ist das Wort OWN vorangestellt.

(07) Alle Übersetzungseinheiten, die neu übersetzt werden müssen, sind mit einem Ausrufungszeichen ("!") gekennzeichnet. Dazu gehören alle modifizierten Übersetzungseinheiten und die davon abhängigen Übersetzungseinheiten.

(08) Nach dem Editieren einer Übersetzungseinheit wird "- undefined -" ausgegeben, da dem System nicht bekannt ist, ob die WITH-Klausel der Übersetzungseinheit verändert wurde.

STEP

Wiederaufsetzpunkt definieren.

STEP

Funktion

STEP definiert den Punkt, an dem wiederaufgesetzt werden soll, wenn in einer Anweisung ein Fehler auftritt. Diese Anweisung kann nur in einer Anweisungsdatei (siehe CALL-Anweisung), einer DO-Prozedur oder einem ENTER-Job angegeben werden.

SYSTEM-COMMAND

In den BS2000-Kommandomodus wechseln oder BS2000-Systemkommando ausführen.

```
SYSTEM-COMMAND

   [COMMAND  =  bs2000-cmd]
```

Funktion

SYSTEM-COMMAND wechselt in den BS2000 Kommandomodus (Wirkung wie die K2-Taste) oder führt ein BS2000-Kommando aus, ohne das Programmiersystem zu beenden.

Tritt bei der Ausführung des Kommandos "bs2000-cmd" ein Fehler auf, dann wird der Spin-Off-Mechanismus im Programmiersystem aktiviert. Ein Fehler bei der Ausführung eines Kommandos, nachdem in den BS2000 Kommandomodus gewechselt wurde, hat keine Wirkung auf den Spin-Off-Mechanismus des Programmiersystems.

COMMAND Legt fest, ob in den BS2000 Kommandomodus verzweigt oder nur ein BS2000 Kommando ausgeführt werden soll. Bei fehlender Angabe des Operanden wird das Programmiersystem unterbrochen und es können beliebige BS2000-Kommandos eingegeben werden. Der Operandenname COMMAND darf nicht eingegeben werden. Er wird sonst als Teil des BS2000-Kommandos aufgefaßt. Nach Eingabe des Kommandos RESUME wird das Programmiersystem fortgesetzt.

= bs2000-cmd
Bezeichnet ein BS2000 Kommando, das in der gewohnten Form angegeben wird.

Hinweis

Kommandos wie EXEC entladen das Programmiersystem. Der Aufruf solcher Kommandos sollte vermieden werden, da das Programmiersystem nicht ordnungsgemäß beendet werden kann.

Entwickeln eines Hauptprogramms

In dieser Beispielsitzung wird ein Programm entwickelt, das eine BS2000-Textdatei auf dem Bildschirm ausgibt. Anschließend wird dieses Programm als Tool in die Programmierumgebung geladen.

```
/EXEC $userid.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//e list.pas ———————————————————————————————————————————————— (01)
```

```
   1.00 ..........................................................................
   2.00 ..........................................................................
   3.00 ..........................................................................
   4.00 ..........................................................................
   5.00 ..........................................................................
   6.00 ..........................................................................
   7.00 ..........................................................................
   8.00 ..........................................................................
   9.00 ..........................................................................
  10.00 ..........................................................................
  11.00 ..........................................................................
  12.00 ..........................................................................
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
                  FILE "$userid.LIST.PAS" DOES NOT EXIST                    - (02)
.....................................................................0000.00:001(0)
```

(01) Aufruf des EDT mit Angabe des Namens der zu editierenden Datei. Die ausführliche Schreibweise dieser Anweisung lautet:

```
EDIT-UNIT UNIT = LIST.PAS (KIND = FILE)
```

(02) In der Statuszeile des EDT wird die Meldung ausgegeben, daß die Datei list.pas noch nicht existiert.

```
   1.00   program list (output, f);
   2.00   var
   3.00      f : text;
   4.00      zeile : string;
   5.00   begin
   6.00      reset (f);
   7.00      while not eof (f) do
   8.00         begin
   9.00         readln (f, zeile)
  10.00         writeln (zeile);
  11.00         end;
  12.00   end.
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
  23.00 ..........................................................................
h.....................................................0001.00:001(0)  - (03)

   >>> (OVER)WRITE "$userid.LIST.PAS" (y/n) ? ------------------------------ (04)
*y
   >>> "$userid.LIST.PAS" (OVER)WRITTEN ---------------------------------- (05)
//mc *e(0),*e(5),(y),ch=on,in=on,op=on ---------------------------------- (06)
//c ------------------------------------------------------------------------ (07)
   >>> 1 COMPILATION ERROR DETECTED (WARNINGS:  0;  NOTES:  0)
//e ,5 ----------------------------------------------------------------------- (08)
```

(03) Nach Eingabe des Quelltextes wird der EDT mittels HALT verlassen. Das Abspeichern des Programms wird anschließend vom Pascal-XT-Programmiersystem übernommen, d.h. es ist kein WRITE-Kommando im EDT erforderlich.

(04) Das Programmiersystem fragt, ob der Inhalt des EDT-Arbeitsbereichs in die Datei list.pas abgespeichert werden soll. Diese Abfrage kann unterdrückt werden, wenn beim Aufruf in der EDIT-Anweisung der Parameter QUERY=NO angegeben wird. In diesem Fall wird dann immer zurückgeschrieben. Da die Datei list.pas bis zu diesem Zeitpunkt noch nicht existiert, wird sie jetzt angelegt.

(05) Diese Meldung bestätigt das erfolgreiche Schreiben in die Datei. Sie erscheint auch bei QUERY=NO.

(06) Mit der MODIFY-COMPILE-Anweisung werden nun die gewünschten Voreinstellungen für die COMPILE-Anweisung getroffen. Die ausführliche Schreibweise dieser Anweisung lautet:

```
MODIFY-COMPILE SOURCE = *EDT (WORKFILE=0),
               LISTING = *EDT (WORKFILE=5),
               MODULE-LIBRARY = *OMF (ERASE = YES),
               CHECK = ON, INITIALIZE = ON, OPTIMIZE = ON
```

Bei einer nachfolgenden COMPILE-Anweisung erwartet der Compiler die Quelle im EDT-Arbeitsbereich 0, schreibt das Listing in den EDT-Arbeitsbereich 5 und das erzeugte Objektprogramm in den *-Bereich, der vor jeder Übersetzung gelöscht wird.

(07) Aufruf des Compilers mit den vorher eingestellten Parametern.

(08) Da die Übersetzung fehlerhaft war, wird nun im Listing nach dem Fehler gesucht. Dazu wird in den Arbeitsbereich 5 des EDT gegangen. Die ausführliche Schreibweise dieser Anweisung lautet:

```
EDIT-UNIT UNIT = *EDT, WORKFILE = 5
```

Die Angabe von *EDT bei Parameter UNIT bewirkt, daß der angegebene Arbeitsbereich beim Aufruf des EDT nicht verändert wird, also nichts eingelesen wird.

```
  1.00   A*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00
  2.00
  3.00
  4.00    GLOBAL OPTIONS FOR THIS COMPILATION
  5.00
  6.00    CHECK      =    ON           BY COMMAND
  7.00    INITIALIZE =    ON           BY COMMAND
  8.00    OPTIMIZE   =    ON           BY COMMAND
  9.00    DEBUG      =    OFF          BY OPTIMIZE OPTION
 10.00    GENERATE   =    ON           BY DEFAULT
 11.00    MAP        =    OFF          BY DEFAULT
 12.00    STANDARD   =    OFF          BY DEFAULT
 13.00    XREF       =    OFF          BY DEFAULT
 14.00
 15.00
 16.00    CURRENT COMPILATION UNIT (SOURCE FILE)
 17.00
 18.00                   *EDT(0)
 19.00
 20.00         1      program list (output, f);
 21.00         2      var
 22.00         3         f : text;

+.....................................................................0001.00:001(5)
```

```
 23.00         4         zeile : string;
 24.00         5      begin
 25.00         6         reset (f);
 26.00         7         while not eof (f) do
 27.00         8            begin
 28.00         9            readln (f, zeile)                                  - (09)
 29.00        10            writeln (zeile);
 30.00    ____________________1
 31.00    >>> 1: ERROR    218: ";" INSERTED
 32.00
 33.00        11            end;
 34.00        12      end.
 35.00
 36.00
 37.00    ************************************************************
 38.00    *                    COMPILATION SUMMARY                   *
 39.00    ************************************************************
 40.00    *  ERRORS DETECTED         :        1                      *
 41.00    *  WARNINGS                :        0                      *
 42.00    *  NOTES                   :        0                      *
 43.00    *  SIZE OF CODE MODULE     :        0 BYTES                *
 44.00    *  SIZE OF DATA MODULE     :        0 BYTES                *
 45.00    *  COMPILATION TIME        :    0.174 SEC                  *
 46.00    ************************************************************
 47.00
0.....................................................................0024.00:001(5)  - (10)
```

(09) In Zeile 9 wurde beim Eingeben des Programms ein Semikolon vergessen.

(10) Zur Korrektur wird nun in den Arbeitsbereich 0 des EDT gewechselt.

```
   1.00   program list (output, f);
   2.00   var
   3.00      f : text;
   4.00      zeile : string;
   5.00   begin
   6.00      reset (f);
   7.00      while not eof (f) do
   8.00         begin
   9.00         readln (f, zeile);
  10.00         writeln (zeile);
  11.00         end;
  12.00   end.
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
  23.00..........................................................................
h.....................................................................0001.00:001(0)
```

```
   >>> (OVER)WRITE "$userid.LIST.PAS" (y/n) ? ———————————————————— (11)
*y
   >>> "$userid.LIST.PAS" (OVER)WRITTEN
//c ———————————————————————————————————————————————————————————— (12)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//sy file list.pas,link=f ———————————————————————————————————— (13)
//r ————————————————————————————————————————————————————————————— (14)
program list (output, f);
var
   f : text;
   zeile : string;
begin
   reset (f);
   while not eof (f) do
      begin
      readln (f, zeile);
      writeln (zeile);
      end;
end.
```

(11) Der EDT wird wieder mit HALT verlassen. Da das Programmiersystem über den Inhalt der EDT-Arbeitsbereiche Buch führt, fragt es nun ab, ob die Datei list.pas zurückgeschrieben werden soll.

(12) Die Übersetzung ist nun erfolgreich.

(13) Vor Ausführung des Programms wird über das FILE-Kommando des BS2000 der Pascal-Datei F eine BS2000-Datei zugeordnet. In diesem Fall wird die Quelle des Programms, die Datei list.pas, als Eingabedatei verwendet.

(14) Mit der RUN-Anweisung wird das Programm list aus dem *-Bereich geladen und gestartet. Die ausführliche Schreibweise dieser Anweisung lautet:

```
RUN-PROGRAM PROGRAM = *LAST-COMPILED-PROGRAM
```

```
//s ——————————————————————————————————————————————————————————— (15)

                    LAST LOADED PROGRAM

     list
//sy rel f ——————————————————————————————————————————————————— (16)
//r ,'f=list.pas' ———————————————————————————————————————————— (17)
program list (output, f);
var
   f : text;
   zeile : string;
begin
   reset (f);
   while not eof (f) do
      begin
      readln (f, zeile);
      writeln (zeile);
      end;
end.
```

(15) Die SHOW-Anweisung informiert über das zuletzt geladene Programm. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = LOADED-PROGRAM
```

(16) Mit dem RELEASE-Kommando des BS2000 wird die Verbindung zwischen BS2000-Datei und Pascal-Datei F aufgelöst.

(17) Die Zuordnung der Pascal-Datei F zu einer BS2000-Datei kann auch mit der RUN-Anweisung hergestellt werden. Die ausführliche Schreibweise dieser Anweisung lautet:

```
RUN-PROGRAM PROGRAM = *LAST-COMPILED-PROGRAM,
            PARAMETER = 'f = list.pas'
```

Diese Zuordnung bleibt solange erhalten, bis sie durch erneute Angabe in der RUN-Anweisung überschrieben, bis ein anderes Programm geladen oder das Programmiersystem verlassen wird.

```
//s ------------------------------------------------------------------------ (18)

                    LAST LOADED PROGRAM

     list
          F  <- - - LIST.PAS
//s t ------------------------------------------------------------------------ (19)
   >>> NO TOOL(S) AVAILABLE
//a (*omf,list),l ------------------------------------------------------------ (20)
//s t ------------------------------------------------------------------------ (21)

     TOOLS OF THE PROGRAMMING ENVIRONMENT

     L
//r l,'f=list.pas' ----------------------------------------------------------- (22)
program list (output, f);
var
   f : text;
   zeile : string;
begin
   reset (f);
   while not eof (f) do
      begin
      readln (f, zeile);
      writeln (zeile);
      end;
end.
```

(18) Die SHOW-Anweisung gibt den Programmnamen und die der Pascal-Datei F zugeordnete BS2000-Datei aus.

(19) Da das Programm als Tool in die Programmiersystemumgebung geladen werden soll, werden mit der SHOW-Anweisung zuerst Informationen über evtl. bereits geladene Tools angefordert. Dadurch wird vermieden, daß ein Aliasname doppelt vergeben wird. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = TOOLS (TOOL = *ALL)
```

(20) Mit der ADD-Anweisung wird das Programm list in der Programmierumgebung unter dem Namen l als Tool geladen. Die ausführliche Schreibweise dieser Anweisung lautet:

```
ADD-TOOL (LIBRARY = *OMF, ELEMENT = LIST), ALIAS-NAME = L
```

(21) Die SHOW-Anweisung zeigt nun, daß ein Tool mit Namen L verfügbar ist.

(22) Vor Ausführung des Tools muß die Zuordnung zwischen der Pascal-Datei F und einer BS2000-Datei entweder über ein FILE-Kommando oder wie hier mit der RUN-Anweisung hergestellt werden. Die ausführliche Schreibweise dieser Anweisung lautet:

```
RUN-PROGRAM PROGRAM = L, PARAMETER = 'f = list.pas'
```

Diese Zuordnung bleibt solange erhalten, bis sie durch erneute Angabe in der RUN-Anweisung überschrieben wird oder das Programmiersystem verlassen wird.

```
//s t ──────────────────────────────────────────────────────────────── (23)

     TOOLS OF THE PROGRAMMING ENVIRONMENT

     L
          F  <- - - LIST.PAS
//r l ──────────────────────────────────────────────────────────────── (24)
program list (output, f);
var
   f : text;
   zeile : string;
begin
   reset (f);
   while not eof (f) do
      begin
      readln (f, zeile);
      writeln (zeile);
      end;
end.
//end ──────────────────────────────────────────────────────────────── (25)
   END OF THE PASCAL SESSION  -  USED TIME = 2.918 SECONDS
%  E732 ABNORMAL PROGRAM TERMINATION APTT101 ──────────────────────── (26)
```

(23) Die SHOW-Anweisung gibt den Namen des Tools und die in der RUN-Anweisung getroffene Dateizuordnung aus.

(24) Da die Dateizuordnung bereits erfolgt ist, kann beim nochmaligen Aufruf des Tools l auf die Angabe des Parameters verzichtet werden.

(25) Das Pascal-XT-Programmiersystem wird beendet. Alle Änderungen von Operandenvoreinstellungen z.B. durch die MODIFY-COMPILE-Anweisung (siehe (06)) gehen jetzt verloren und werden beim nächsten Start des Programmiersystems nicht wieder hergestellt.

(26) Sobald im Programmiersystem eine Anweisung fehlerhaft war oder bei einer Übersetzung Fehler aufgetreten sind, wird beim Beenden des Programmiersystems der SPIN-OFF-Mechanismus des BS2000 aktiviert. Dies bewirkt, daß im Dialog die Meldung `ABNORMAL PROGRAM TERMINATION` ausgegeben wird, bzw. in DO-Prozeduren und ENTER-Jobs zum nächsten STEP oder ENDP bzw. LOGOFF verzweigt wird.

Arbeiten mit PLAM-Bibliotheken und der Projektdatei

In dieser Beispielsitzung wird ein Programm entwickelt, das ein Hilfsmittel zur Bereinigung von Benutzerkennungen darstellt. Es gibt eine Liste aller katalogisierten Dateien aus und fragt anschließend für jede Datei, ob sie gelöscht werden soll.

Das Programm besteht aus dem Hauptprogramm ERAQ und den Paketen FSTAT und DIALOG. Außerdem werden die beiden vordefinierten Pakete BS2000CALLS und DMSIO verwendet.

Es wird besonders die Benutzung der Projektdatei sowie die Programmentwicklung unter Verwendung von PLAM-Bibliotheken gezeigt.

```
/EXEC $userid.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ... LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//E (ERAQ.PLAM,FSTAT.SPEC) ———————————————————————————————————————— (01)
```

```
   1.00 ..........................................................................
   2.00 ..........................................................................
   3.00 ..........................................................................
   4.00 ..........................................................................
   5.00 ..........................................................................
   6.00 ..........................................................................
   7.00 ..........................................................................
   8.00 ..........................................................................
   9.00 ..........................................................................
  10.00 ..........................................................................
  11.00 ..........................................................................
  12.00 ..........................................................................
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
                  OPEN ERROR ON LIBRARY "$userid.ERAQ.PLAM"                      - (02)
.....................................................................0000.00:001(0)
```

(01) Aufruf des EDT mit Angabe des Bibliotheksnamens und des Namens des zu editierenden Elements. Weder die Bibliothek noch das Element existieren zu diesem Zeitpunkt. Die ausführliche Schreibweise dieser Anweisung lautet:

```
EDIT-UNIT UNIT = *LIBRARY (LIBRARY = ERAQ.PLAM,
                           ELEMENT = FSTAT.SPEC)
```

(02) Da die Bibliothek eraq.plam noch nicht existiert, wird in der Statuszeile des EDT die Meldung ausgegeben, daß sie nicht geöffnet werden kann.

```
   1.00 package fstat;
   2.00 type
   3.00    fntype = string[54];
   4.00
   5.00 procedure makefstat (filename: fntype);
   6.00
   7.00 end.
   8.00 ..........................................................................
   9.00 ..........................................................................
  10.00 ..........................................................................
  11.00 ..........................................................................
  12.00 ..........................................................................
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
  23.00..........................................................................
h.....................................................................0001.00:001(0)
```

```
  >>> (OVER)WRITE "($userid.ERAQ.PLAM,FSTAT.SPEC(*UPPER-LIMIT,S))" (y/n) ? ———— (03)
*y
  >>> "($userid.ERAQ.PLAM,FSTAT.SPEC(*UPPER-LIMIT,S))" (OVER)WRITTEN
//ME (ERAQ.PLAM) ———————————————————————————————————————————————————— (04)
//E (,FSTAT.BODY),1 ——————————————————————————————————————————————————— (05)
```

(03) Nach Eingabe des Quelltextes wird der EDT mittels HALT verlassen. Das Programmiersystem fragt, ob der Inhalt des EDT-Arbeitsbereichs in das Bibliothekselement fstat.spec in der Bibliothek eraq.plam abgespeichert werden soll. Diese Abfrage kann unterdrückt werden, wenn beim Aufruf in der EDIT-Anweisung der Parameter QUERY=NO angegeben wird. In diesem Fall wird dann ohne Abfrage immer zurückgeschrieben. Da die Bibliothek eraq.plam bis zu diesem Zeitpunkt noch nicht existiert, wird sie jetzt als PLAM-Bibliothek angelegt und anschließend das Element fstat.spec vom Typ S hineingeschrieben.

(04) Als Schreiberleichterung wird mit der MODIFY-EDIT-Anweisung der Name der Bibliothek eraq.plam voreingestellt. Die ausführliche Schreibweise dieser Anweisung lautet:

```
MODIFY-EDIT UNIT = *LIBRARY (LIBRARY = ERAQ.PLAM)
```

(05) Ab jetzt genügt es, in der EDIT-Anweisung den Elementnamen anzugeben. Die Implementierung des Pakets soll im Arbeitsbereich 1 eingegeben werden, damit, falls notwendig, die Spezifikation im Arbeitsbereich 0 noch zur Verfügung steht. Die ausführliche Schreibweise dieser Anweisung lautet:

```
EDIT-UNIT UNIT = *LIBRARY (, ELEMENT = FSTAT.BODY), WORKFILE = 1
```

```
   1.00 ..................................................................
   2.00 ..................................................................
   3.00 ..................................................................
   4.00 ..................................................................
   5.00 ..................................................................
   6.00 ..................................................................
   7.00 ..................................................................
   8.00 ..................................................................
   9.00 ..................................................................
  10.00 ..................................................................
  11.00 ..................................................................
  12.00 ..................................................................
  13.00 ..................................................................
  14.00 ..................................................................
  15.00 ..................................................................
  16.00 ..................................................................
  17.00 ..................................................................
  18.00 ..................................................................
  19.00 ..................................................................
  20.00 ..................................................................
  21.00 ..................................................................
  22.00 ..................................................................
                  ELEMENT "FSTAT.BODY (*UPPER-LIMIT,S)" DOES NOT EXIST          - (06)
.....................................................................0000.00:001(1)
```

```
   1.00 with BS2000CALLS;
   2.00 package body fstat;
   3.00
   4.00 procedure makefstat (filename: fntype);
   5.00 var
   6.00    b    : boolean;
   7.00 begin
   8.00    BS2000CALLS.cmd (concat('/SYSFILE SYSLST=',filename), b);
   9.00    BS2000CALLS.cmd ('/OPTION MSG=FHL', b);
  10.00    BS2000CALLS.cmd ('/FSTAT', b);
  11.00    BS2000CALLS.cmd ('/OPTION MSG=F', b);
  12.00    BS2000CALLS.cmd ('/SYSFILE SYSLST=(PRIMARY)', b);
  13.00 end;
  14.00
  15.00 begin
  16.00 end.
  17.00 ..................................................................
  18.00 ..................................................................
  19.00 ..................................................................
  20.00 ..................................................................
  21.00 ..................................................................
  22.00 ..................................................................
  23.00..................................................................
h.....................................................................0001.00:001(1)
```

```
  >>> (OVER)WRITE "($userid.ERAQ.PLAM,FSTAT.BODY(*UPPER-LIMIT,S))" (y/n) ?
*y
  >>> "($userid.ERAQ.PLAM,FSTAT.BODY(*UPPER-LIMIT,S))" (OVER)WRITTEN
```

(06) In der Statuszeile des EDT wird die Meldung ausgegeben, daß das Element fstat.body noch nicht existiert.

```
//E (,DIALOG.SPEC),2 ──────────────────────────────────────────────────────── (07)
```

```
   1.00 ..................................................................
   2.00 ..................................................................
   3.00 ..................................................................
   4.00 ..................................................................
   5.00 ..................................................................
   6.00 ..................................................................
   7.00 ..................................................................
   8.00 ..................................................................
   9.00 ..................................................................
  10.00 ..................................................................
  11.00 ..................................................................
  12.00 ..................................................................
  13.00 ..................................................................
  14.00 ..................................................................
  15.00 ..................................................................
  16.00 ..................................................................
  17.00 ..................................................................
  18.00 ..................................................................
  19.00 ..................................................................
  20.00 ..................................................................
  21.00 ..................................................................
  22.00 ..................................................................
                 ELEMENT "DIALOG.SPEC (*UPPER-LIMIT,S)" DOES NOT EXIST
.....................................................................0000.00:001(2)
```

```
   1.00 package dialog;
   2.00 type
   3.00    fntype = string[54];
   4.00
   5.00 procedure makedialog (filename: fntype);
   6.00
   7.00 end.
   8.00 ..................................................................
   9.00 ..................................................................
  10.00 ..................................................................
  11.00 ..................................................................
  12.00 ..................................................................
  13.00 ..................................................................
  14.00 ..................................................................
  15.00 ..................................................................
  16.00 ..................................................................
  17.00 ..................................................................
  18.00 ..................................................................
  19.00 ..................................................................
  20.00 ..................................................................
  21.00 ..................................................................
  22.00 ..................................................................
  23.00..................................................................
h.....................................................................0001.00:001(2)
```

```
  >>> (OVER)WRITE "($userid.ERAQ.PLAM,DIALOG.SPEC(*UPPER-LIMIT,S))" (y/n) ?
*y
  >>> "($userid.ERAQ.PLAM,DIALOG.SPEC(*UPPER-LIMIT,S))" (OVER)WRITTEN
```

(07) Die Spezifikation des Pakets dialog wird im Arbeitsbereich 2 des EDT eingegeben.

```
//E (,DIALOG.BODY),3 ──────────────────────────────────────────────────────── (08)
```

```
   1.00 ..........................................................................
   2.00 ..........................................................................
   3.00 ..........................................................................
   4.00 ..........................................................................
   5.00 ..........................................................................
   6.00 ..........................................................................
   7.00 ..........................................................................
   8.00 ..........................................................................
   9.00 ..........................................................................
  10.00 ..........................................................................
  11.00 ..........................................................................
  12.00 ..........................................................................
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
                 ELEMENT "DIALOG.BODY (*UPPER-LIMIT,S)" DOES NOT EXIST
.....................................................................0000.00:001(3)
```

```
   1.00 with BS2000CALLS, DMSIO;
   2.00 package body dialog (input, output, tmp);
   3.00 var
   4.00    tmp  : text;
   5.00
   6.00 procedure makedialog (filename: fntype);
   7.00 var
   8.00    c    : char;
   9.00    b    : boolean;
  10.00    zeile: string;
  11.00 begin
  12.00    assignfile (tmp, filename);
  13.00    reset (tmp);
  14.00    while not eoln (tmp) do
  15.00       begin
  16.00       readln (tmp, zeile);
  17.00       delete (zeile, 1, 27);
  18.00       writeln ('Soll die Datei ''', zeile,
  19.00                ''' geloescht werden?(Y/N/E):');
  20.00       readln;
  21.00       read (c);
  22.00       if c in ['Y','y','j','J'] then
  23.00           begin
+.....................................................................0001.00:001(3)
```

(08) Die Implementierung des Pakets dialog wird im Arbeitsbereich 3 des EDT eingegeben.

```
  24.00           BS2000CALLS.cmd (concat ('/ERASE ',zeile), b);
  25.00           if not b then
  26.00              begin
  27.00              writeln ('Datei ''', zeile, ''' geloescht!');
  28.00              writeln;
  29.00              end
  30.00           end
  31.00       else if (c = 'E') or (c = 'e') then
  32.00           exit;
  33.00       end;
  34.00    DMSIO.close (tmp ) ;
  35.00    BS2000CALLS.cmd (concat('/ERASE', filename), b )
  36.00 end;
  37.00
  38.00 begin
  39.00 end.
  40.00 ..........................................................................
  41.00 ..........................................................................
  42.00 ..........................................................................
  43.00 ..........................................................................
  44.00 ..........................................................................
  45.00 ..........................................................................
  46.00 ..........................................................................
h.....................................................................0024.00:001(3)
```

```
  >>> (OVER)WRITE "($userid.ERAQ.PLAM,DIALOG.BODY(*UPPER-LIMIT,S))" (y/n) ?
*y
  >>> "($userid.ERAQ.PLAM,DIALOG.BODY(*UPPER-LIMIT,S))" (OVER)WRITTEN
//E (,ERAQ.PROG),4 ──────────────────────────────────────────────────────────── (09)
```

```
   1.00 ..........................................................................
   2.00 ..........................................................................
   3.00 ..........................................................................
   4.00 ..........................................................................
   5.00 ..........................................................................
   6.00 ..........................................................................
   7.00 ..........................................................................
   8.00 ..........................................................................
   9.00 ..........................................................................
  10.00 ..........................................................................
  11.00 ..........................................................................
  12.00 ..........................................................................
  13.00 ..........................................................................
  14.00 ..........................................................................
  15.00 ..........................................................................
  16.00 ..........................................................................
  17.00 ..........................................................................
  18.00 ..........................................................................
  19.00 ..........................................................................
  20.00 ..........................................................................
  21.00 ..........................................................................
  22.00 ..........................................................................
                  ELEMENT "ERAQ.PROG (*UPPER-LIMIT,S)" DOES NOT EXIST
.....................................................................0000.00:001(4)
```

(09) Das Hauptprogramm eraq.prog wird im Arbeitsbereich 4 des EDT eingegeben.

```
   1.00 with FSTAT, DIALOG;
   2.00 program eraq (input, output);
   3.00 const
   4.00    tempfile = 'TMP.SYSLST';
   5.00
   6.00 begin
   7.00    FSTAT.makefstat (tempfile);
   8.00    DIALOG.makedialog (tempfile);
   9.00 end.
  10.00 .....................................................................
  11.00 .....................................................................
  12.00 .....................................................................
  13.00 .....................................................................
  14.00 .....................................................................
  15.00 .....................................................................
  16.00 .....................................................................
  17.00 .....................................................................
  18.00 .....................................................................
  19.00 .....................................................................
  20.00 .....................................................................
  21.00 .....................................................................
  22.00 .....................................................................
  23.00.....................................................................
h.............................................................0001.00:001(4)
```

```
   >>> (OVER)WRITE "($userid.ERAQ.PLAM,ERAQ.PROG(*UPPER-LIMIT,S))" (y/n) ?
*y
   >>> "($userid.ERAQ.PLAM,ERAQ.PROG(*UPPER-LIMIT,S))" (OVER)WRITTEN
//D ERAQ.DIR ──────────────────────────────────────────────────────────────── (10)
   >>> PROJECT DIRECTORY FILE CREATED ──────────────────────────────────────── (11)
//C ($PASSUP-XT,BS2000CALLS.SPEC),*D ──────────────────────────────────────── (12)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C ($PASSUP-XT,DMSIO.SPEC),*D
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
```

(10) Nachdem alle Quellen eingegeben wurden, muß nun vor der ersten Übersetzung die Projektdatei eingestellt werden. Die ausführliche Schreibweise dieser Anweisung lautet:

```
DEFINE-PROJECT-FILE DIRECTORY = ERAQ.DIR
```

(11) Da die Projektdatei eraq.dir noch nicht existiert, wird sie durch die vorangegangene DEFINE-PROJECT-FILE-Anweisung eingerichtet, zur Bearbeitung eröffnet und diese Meldung ausgegeben. Existiert die Projektdatei bereits, so wird sie nur eröffnet und die Ausgabe einer Meldung unterbleibt.

(12) Bevor eine Spezifikation bei der Übersetzung einer anderen Spezifikation, einer Paketimplementierung oder eines Hauptprogramms benutzt werden kann, muß diese Spezifikation in die Projektdatei eingetragen werden. Dies geschieht durch Übersetzung dieser Spezifikation. Hier werden zuerst die Spezifikation der vordefinierten Pakete BS2000CALLS und DMSIO übersetzt. Da die Listings dieser beiden Übersetzungen nicht interessieren, wird das Listing in die BS2000-Datei *DUMMY geschrieben. Die ausführliche Schreibweise dieser Anweisung lautet:

```
COMPILE-UNIT SOURCE = *LIBRARY (SOURCE-LIBRARY = $PASSUP-XT,
                                SOURCE-ELEMENT = DMSIO.SPEC),
             LISTING = *DUMMY
```

```
//MC (ERAQ.PLAM),(*STD,*STD),*STD,CH=ON,IN=ON,OP=ON ———————————————— (13)
//C (,FSTAT.SPEC) ——————————————————————————————————————————————— (14)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,FSTAT.BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,DIALOG.SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,DIALOG.BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,ERAQ.PROG)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//R ——————————————————————————————————————————————————————————— (15)
0000003: C:$userid.AAAA
0000003: C:$userid.AAAA2
0000003: C:$userid.ERAQ.DIR
0000048: C:$userid.ERAQ.PLAM
Soll die Datei 'C:$userid.AAAA' geloescht werden?(Y/N/E): ———————— (16)
n
Soll die Datei 'C:$userid.AAAA2' geloescht werden?(Y/N/E):
e
//E DIALOG(B) ——————————————————————————————————————————————————— (17)
```

(13) Mit der MODIFY-COMPILE-Anweisung werden die Standardwerte der Operanden für die folgenden Übersetzungen geändert: Quellbibliothek = ERAQ.PLAM, Listingbibliothek = Quellbibliothek, Listingelement = P-Element mit selbem Namen wie Quellelement, Modulbibliothek = Quellbibliothek und die Optionen Check, Initialize und Optimize auf ON. Die ausführliche Schreibweise dieser Anweisung lautet:

```
MODIFY-COMPILE SOURCE = *LIBRARY (SOURCE-LIBRARY = ERAQ.PLAM),
               LISTING = *LIBRARY (LIST-LIBRARY = *STD,
                                   LIST-ELEMENT = *STD),
               MODULE = *STD,
               CHECK = ON, INITIALIZE = ON, OPTIMIZE = ON
```

Durch diese Einstellungen benötigt man bei der Programmentwicklung nur eine PLAM-Bibliothek. In dieser Bibliothek werden die Quellen, die Listings und die Objektmodule verwaltet.

(14) Zur Übersetzung muß nur noch der Elementname in der COMPILE-Anweisung angegeben werden. Die ausführliche Schreibweise dieser Anweisung lautet:

```
COMPILE-UNIT SOURCE = *LIBRARY (SOURCE-ELEMENT = FSTAT.SPEC)
```

(15) Mit der RUN-Anweisung wird nun das zuletzt übersetzte Programm aus der Bibliothek gestartet, in die es hineingeschrieben wurde. Daraus folgt, daß das Hauptprogramm immer zuletzt übersetzt werden muß. Der DLL lädt die benötigten Pakete aus derselben Bibliothek nach (Autolink-Mechanismus), aus der das Hauptprogramm geladen wurde. Es ist nicht möglich, ein Programm mit Paketen aus dem *-Bereich heraus zu starten, da dort der Autolink-Mechanismus des DLL nicht funktioniert. Hier wird also das Programm eraq aus eraq.plam gestartet und die Externreferenzen auf die Pakete fstat und dialog werden aus der Bibliothek eraq.plam befriedigt.

(16) Das Programm ist fehlerhaft. Es löscht das erste Zeichen des Dateinamens (hier ein Blank).

(17) Nach der ersten fehlerfreien Übersetzung sind alle Übersetzungseinheiten in die Projektdatei eingetragen. Man kann nun beim Editieren und Übersetzen mit den Paket- oder Programm-Namen arbeiten. Nun wird die Übersetzungseinheit DIALOG mit Kind BODY editiert. Die ausführliche Schreibweise dieser Anweisung lautet:

```
EDIT-UNIT UNIT = DIALOG (KIND = BODY)
```

```
   1.00 with BS2000CALLS, DMSIO;
   2.00 package body dialog (input, output, tmp);
   3.00 var
   4.00    tmp  : text;
   5.00
   6.00 procedure makedialog (filename: fntype);
   7.00 var
   8.00    c    : char;
   9.00    b    : boolean;
  10.00    zeile: string;
  11.00 begin
  12.00    assignfile (tmp, filename);
  13.00    reset (tmp);
  14.00    while not eoln (tmp) do
  15.00       begin
  16.00       readln (tmp,  zeile);
  17.00       delete (zeile, 1, 27);                                          - (18)
  18.00       writeln ('Soll die Datei ''', zeile,
  19.00                ''' geloescht werden?(Y/N/E):');
  20.00       readln;
  21.00       read (c);
  22.00       if c in ['Y','y','j','J'] then
                 ($userid.ERAQ.PLAM,DIALOG.BODY(*UPPER-LIMIT,S))
.....................................................................0001.00:001(0)
```

```
   1.00 with BS2000CALLS, DMSIO;
   2.00 package body dialog (input, output, tmp);
   3.00 var
   4.00    tmp  : text;
   5.00
   6.00 procedure makedialog (filename: fntype);
   7.00 var
   8.00    c    : char;
   9.00    b    : boolean;
  10.00    zeile: string;
  11.00 begin
  12.00    assignfile (tmp, filename);
  13.00    reset (tmp);
  14.00    while not eoln (tmp) do
  15.00       begin
  16.00       readln (tmp, zeile);
  17.00       delete (zeile, 1, 26);                                          - (19)
  18.00       writeln ('Soll die Datei ''', zeile,
  19.00                ''' geloescht werden?(Y/N/E):');
  20.00       readln;
  21.00       read (c);
  22.00       if c in ['Y','y','j','J'] then
  23.00          begin
h.....................................................................0001.00:001(0)
```

```
   >>> (OVER)WRITE PACKAGE BODY "DIALOG" (y/n) ? ——————————————————————— (20)
*y
   >>> PACKAGE BODY "DIALOG" (OVER)WRITTEN
```

(18) Der Programmfehler liegt in Zeile 17. Die Längenangabe in der Stringprozedur Delete ist um eins zu groß.

(19) Fehlerkorrektur.

(20) Für die Rückfrage, ob das Paket überschrieben werden soll, wird jetzt auch der Unitname verwendet und nicht mehr der Bibliotheks- und Elementname.

```
//S R ------------------------------------------------------------------------ (21)
COMPILE DIALOG (BODY) ------------------------------------------------------- (22)
//S P ------------------------------------------------------------------------ (23)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (ERAQ.DIR)

PACKAGE NAME    KIND  LOCATION
BS2000CALLS    (SPEC) ($PASSUP-XT,BS2000CALLS.SPEC(*UPPER-LIMIT,S))
DIALOG         (SPEC) ($userid.ERAQ.PLAM,DIALOG.SPEC(*UPPER-LIMIT,S))
DIALOG        !(BODY) ($userid.ERAQ.PLAM,DIALOG.BODY(*UPPER-LIMIT,S)) ------- (24)
DMSIO          (SPEC) ($PASSUP-XT,DMSIO.SPEC(*UPPER-LIMIT,S))
ERAQ           (PROG) ($userid.ERAQ.PLAM,ERAQ.PROG(*UPPER-LIMIT,S))
FSTAT          (SPEC) ($userid.ERAQ.PLAM,FSTAT.SPEC(*UPPER-LIMIT,S))
FSTAT          (BODY) ($userid.ERAQ.PLAM,FSTAT.BODY(*UPPER-LIMIT,S))

//S P(K=BODY,REF=ALL) --------------------------------------------------------- (25)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (ERAQ.DIR)

PACKAGE NAME    KIND  LOCATION

DIALOG        !(BODY) ($userid.ERAQ.PLAM,DIALOG.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   - undefined - ------------------------------------------------ (26)
      indirect: - undefined -

FSTAT          (BODY) ($userid.ERAQ.PLAM,FSTAT.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   BS2000CALLS, FSTAT (OWN SPEC) ------------------------------- (27)
      indirect: - -
```

(21) Mit der SHOW-Anweisung werden jetzt die erforderlichen Neuübersetzungen ausgegeben. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = RECOMPILATIONS (PACKAGES = *NECESSARY,
                                       KIND = ALL)
```

(22) Als Ausgabe von SHOW RECOMPILATIONS erhält man COMPILE-Anweisungen für die gewünschten Übersetzungseinheiten.

(23) Mit der SHOW-Anweisung wird der Inhalt der Projektdatei ausgegeben. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = PROJECT-FILE
```

(24) In der Ausgabe sind die nochmals zu übersetzenden Pakete mit einem Ausrufungszeichen markiert.

(25) Mit der SHOW-Anweisung werden die Spezifikationen ausgegeben, die von den Paketimplementierungen importiert werden. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = PROJECT-FILE (KIND = BODY,
                                     REFERENCES = ALL)
```

(26) Die Implementierung von Dialog ist als ungültig gekennzeichnet, deshalb können für dieses Paket keine Beziehungen angegeben werden. Es könnte ja die WITH-Klausel des Pakets verändert worden sein.

(27) Die Implementierung von fstat verwendet die Spezifikation von BS2000CALLS und natürlich die eigene Spezifikation. In diesem Beispiel gibt es keine indirekt importierten Pakete, da die importierten Spezifikationen ihrerseits keine weitere Pakete importieren.

```
//S P (USE=ALL) ——————————————————————————————————————————————————— (28)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (ERAQ.DIR)

PACKAGE NAME    KIND  LOCATION

BS2000CALLS    (SPEC) ($PASSUP-XT,BS2000CALLS.SPEC(*UPPER-LIMIT,S))
    USED BY
      direct:   DIALOG (BODY), FSTAT (BODY) ——————————————————————— (29)
      indirect: - -

DIALOG         (SPEC) ($userid.ERAQ.PLAM,DIALOG.SPEC(*UPPER-LIMIT,S))
    USED BY
      direct:   DIALOG (OWN BODY), ERAQ (PROG)
      indirect: - -

DIALOG        !(BODY) ($userid.ERAQ.PLAM,DIALOG.BODY(*UPPER-LIMIT,S)) ——— (30)

DMSIO          (SPEC) ($PASSUP-XT,DMSIO.SPEC(*UPPER-LIMIT,S))
    USED BY
      direct:   DIALOG (BODY)
      indirect: - -

ERAQ           (PROG) ($userid.ERAQ.PLAM,ERAQ.PROG(*UPPER-LIMIT,S))

FSTAT          (SPEC) ($userid.ERAQ.PLAM,FSTAT.SPEC(*UPPER-LIMIT,S))
    USED BY
      direct:   ERAQ (PROG), FSTAT (OWN BODY)
      indirect: - -

FSTAT          (BODY) ($userid.ERAQ.PLAM,FSTAT.BODY(*UPPER-LIMIT,S))
```

(28) Mit der SHOW-Anweisung wird nun die umgekehrte Relation ausgegeben, d.h. von welchen Paketen ein Paket importiert wird. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = PROJECT-FILE (USED-BY = ALL)
```

(29) Die Spezifikation von BS2000CALLS wird von den Implementierungen der Pakete DIALOG und FSTAT verwendet.

(30) Paketimplementierungen können von anderen Paketen nicht importiert werden.

```
//S R,*E(8) ——————————————————————————————————————————————— (31)
//CA *E(8) ———————————————————————————————————————————————— (32)
(%STMT) COMPILE DIALOG (BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//S R ————————————————————————————————————————————————————— (33)
   >>> NO RECOMPILATIONS NECESSARY
//R (ERAQ.PLAM,ERAQ) ————————————————————————————————————— (34)
0000003 :C:$userid.AAAA
0000003 :C:$userid.AAAA2
0000003 :C:$userid.ERAQ.DIR
0000048 :C:$userid.ERAQ.PLAM
Soll die Datei ':C:$userid.AAAA' geloescht werden?(Y/N/E):
y
Datei ':C:$userid.AAAA' geloescht!
Soll die Datei ':C:$userid.AAAA2' geloescht werden?(Y/N/E):
y
Datei ':C:$userid.AAAA2' geloescht!
Soll die Datei ':C:$userid.ERAQ.DIR' geloescht werden?(Y/N/E):
n
Soll die Datei ':C:$userid.ERAQ.PLAM' geloescht werden?(Y/N/E):
e
//END
   END OF THE PASCAL SESSION  -  USED TIME = 31.091 SECONDS
```

(31) Mit der SHOW-Anweisung werden nochmals die erforderlichen Neuübersetzungen ausgegeben. Die Ausgabe erfolgt diesmal in den Arbeitsbereich 8 des EDT, damit die erzeugten COMPILE-Anweisungen mit einer CALL-STATEMENT-FILE-Anweisung ausgeführt werden können. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = RECOMPILATIONS (PACKAGES = *NECESSARY,
                                       KIND = ALL),
                OUTFILE = *EDT (WORKFILE = 8)
```

(32) Die Anweisungen im Arbeitsbereich 8 des EDT werden mit der CALL-STATEMENT-FILE-Anweisung ausgeführt. Die mit der MODIFY-COMPILE-Anweisung gemachten Voreinstellungen (siehe (13)) sind immer noch wirksam. Die ausführliche Schreibweise dieser Anweisung lautet:

```
CALL-STATEMENT-FILE STMT-FILE = *EDT (WORKFILE = 8)
```

(33) Die SHOW-Anweisung gibt aus, daß keine Neuübersetzungen mehr erforderlich sind.

(34) Da die zuletzt übersetzte Einheit kein Hauptprogramm war, muß bei der RUN-Anweisung die Bibliothek und der Name des zu startenden Programms angegeben werden, sonst würde die RUN-Anweisung mit Meldung

```
>>> LAST COMPILATION UNIT IS NOT EXECUTABLE
```

abgewiesen. Die ausführliche Schreibweise dieser Anweisung lautet:

```
RUN-PROGRAM PROGRAM = (LIBRARY = ERAQ.PLAM, ELEMENT = ERAQ)
```

Die Projektdatei

Die Projektdatei unterstützt das Paketkonzept, das für die Sprache Pascal-XT von zentraler Bedeutung ist. Das Paketkonzept erlaubt das Aufteilen großer, komplexer Programme in kleinere Einheiten (Pakete), die getrennt bearbeitet, übersetzt und verwaltet werden können. Jedes Paket besteht aus einer Spezifikation und einer Implementierung.

Übersetzungseinheiten sind Paket-Spezifikationen, Paket-Implementierungen und Hauptprogramme. Der Name einer Übersetzungseinheit ist immer der Pascal-Bezeichner, der hinter dem Schlüsselwort "program", "package" oder "body" angegeben ist.

Der Programmname eines Hauptprogramms muß von den Paketnamen aller zum Programm gehörigen Pakete verschieden sein.

Aufgaben der Projektdatei

Programmnamen zu Dateinamen zuordnen

In der Projektdatei werden die Paketnamen und der Programmname des Hauptprogramms den Namen der Dateien zugeordnet, in denen die Übersetzungseinheiten des Programms stehen. Diese Zuordnung wird für mehrere Funktionen benötigt.

- Um die Schnittstellen zwischen Paketen zu überprüfen, liest der Pascal-XT-Compiler nochmals die Spezifikationen der importierten Pakete. Da die WITH-Klauseln einer Übersetzungseinheit nur die Namen der importierten Pakete enthalten, muß der Compiler die Namen der Dateien mitgeteilt bekommen, in denen die Spezifikationen abgespeichert sind. Diese Information entnimmt der Compiler der Zuordnung der Paketnamen von Spezifikationen zu den Dateinamen in der Projektdatei.
- Bei den häufig verwendeten Anweisungen an das Programmiersystem EDIT und COMPILE kann man statt eines Dateinamens auch den Namen eines Hauptprogramms oder Pakets angeben, das bearbeitet werden soll. Auch hierfür wird auf die Zuordnungen in der Projektdatei zugegriffen.

Die Zuordnung von Paket- und Programmnamen zu den Dateinamen trägt der Compiler in die Projektdatei ein (siehe 3.2).

Beziehungen zwischen Übersetzungseinheiten festhalten

Ein Pascal-XT Programm besteht i.a. aus einem Hauptprogramm und einer Menge von Paketen. Die Beziehungen zwischen den Übersetzungseinheiten sind in den WITH-Klauseln der Übersetzungseinheiten statisch festgelegt. Ein Programm kann damit durch einen Graphen dargestellt werden, dessen Knoten die Übersetzungseinheiten sind und bei dem das Hauptprogramm eine Wurzel darstellt. Welche Beziehungen zwischen den Übersetzungseinheiten bestehen, ist insbesondere dann von Interesse, wenn eine Übersetzungseinheit geändert werden soll. Anhand der Beziehungen kann leicht festgestellt werden, ob sich die Änderungen auch auf andere Übersetzungseinheiten auswirken, die dann neu übersetzt werden müssen.

Aus der Projektdatei lassen sich die Beziehungen zwischen den Übersetzungseinheiten ableiten und mit der SHOW-Anweisung (siehe 2.6.12 und 3.2) ausgeben.

Statusinformationen abspeichern

Für jede Übersetzungseinheit werden in der Projektdatei einige Statusinformationen abgespeichert. Dazu gehören Datum und Uhrzeit der letzten Übersetzung und eine Änderungsmarkierung, die nach einer Modifikation mit der EDIT-Anweisung gesetzt wird. Die SHOW-Anweisung greift auf diese Statusinformationen zu, um festzustellen, welche Übersetzungseinheiten neu übersetzt werden müssen.

Einstellen und Bearbeiten der Projektdatei

Einstellen der Projektdatei

Die Projektdatei wird mit der DEFINE-Anweisung (siehe 2.6.5) eingestellt. Sie wird eingerichtet, wenn sie noch nicht existiert und bleibt solange geöffnet, bis eine andere Projektdatei eingestellt oder das Programmiersystem verlassen wird.

Modifizieren der Projektdatei

Die Zuordnungen von Paket- bzw. Programm-Namen zu den Dateinamen, in denen die Quellprogramme enthalten sind, werden vom Compiler nur nach einer fehlerfreien Übersetzung in die Projektdatei eingetragen, sofern eine eingestellt ist. Bereits existierende Zuordnungen werden überschrieben. Nach der fehlerfreien Übersetzung einer Paket-Spezifikation wird vom Programmiersystem eine Fehlermeldung ausgegeben, wenn die Projektdatei nicht aktualisiert werden kann.

Nach einer Änderung wird eine Übersetzungseinheit als modifiziert markiert, wenn die Änderung unter der Kontrolle des Programmiersystems erfolgt (siehe EDIT-Anweisung). Anhand der Änderungsanzeige können notwendige Rekompilierungen festgestellt werden.

Nach jeder Übersetzung einer Übersetzungseinheit wird das Übersetzungsdatum in der Projektdatei vermerkt, um notwendige Rekompilierungen erkennen zu können. So muß das Übersetzungsdatum einer Übersetzungseinheit jünger sein als das der Spezifikationen aller importierten Pakete. Eine Übersetzung einer Spezifikation, auch wenn keine Änderungen durchgeführt wurden, hat immer die Rekompilierung der abhängigen Pakete zur Folge.

Löschen von Einträgen in der Projektdatei

Die Informationen für ein nicht mehr benötigtes Hauptprogramm oder Paket können mit der Anweisung REMOVE-DIRECTORY-ENTRY aus der aktuell eingestellten Projektdatei gelöscht werden. Es werden alle Einträge für den beim Aufruf der Anweisung angegebenen Namen gelöscht. Mit der Löschung der Einträge bleiben aber weiterhin die Dateien der Quellprogramme bestehen.

Übersetzungseinheiten, die ein gelöschtes Paket importieren, werden als ungültig markiert. Sie müssen neu übersetzt werden.

Statusinformationen ausgeben

Mit der SHOW-Anweisung können durch Angabe der Operanden PROJECT oder RECOMPILATIONS verschiedene Informationen der Projektdatei ausgegeben werden. Standardmäßig werden alle Paket- bzw. Hauptprogramm-Namen und die Dateinamen ausgegeben, in denen die Quellprogramme enthalten sind. Zusätzlich können folgende Paketbeziehungen mit aufgelistet werden:

(a) Für eine Übersetzungseinheit können alle Pakete ausgegeben werden, die von X direkt und indirekt importiert werden. Die direkt importierten Pakete sind in der WITH-Klausel von X angegeben. Diese können wiederum Pakete importieren, die damit von X indirekt importiert werden.
Bei der Ausgabe dieser Beziehungen wird zusätzlich unterschieden, welche Pakete von der Spezifikation und welche von der Implementierung importiert werden.

(b) Für jedes Paket X können alle Übersetzungseinheiten bestimmt werden, die von diesem Paket X abhängen, d.h. die das Paket X direkt oder indirekt importieren. Diese Beziehungen werden zur Bestimmung der notwendigen Rekompilierungen benötigt. Nach einer Änderung der Spezifikation des Pakets X müssen alle Übersetzungseinheiten, die von X abhängen, nochmals übersetzt werden. Damit wird die Konsistenz eines Programms gewährleistet und die Anzahl der Übersetzungen auf die von den Änderungen betroffenen Übersetzungseinheiten beschränkt.

Nach Änderungen oder Neuübersetzungen von Übersetzungseinheiten können alle Pakete bestimmt werden, die nochmals übersetzt werden müssen. Die Anweisung SHOW RECOMPILATIONS erzeugt COMPILE-Anweisungen, so daß die Übersetzungen in der richtigen Reihenfolge ausgeführt werden.

Hinweise zum Arbeiten mit der Projektdatei

Die Projektdatei ist mehrfach benutzbar, d.h. es können mehrere Benutzer gleichzeitig darauf lesend und schreibend zugreifen. Wenn mehrere Benutzer gleichzeitig auf denselben Satz der Projektdatei schreibend zugreifen, werden diese Schreibzugriffe nacheinander ausgeführt.

Bei der Projektentwicklung hat sich folgendes Vorgehen als nützlich erwiesen:

- Für jedes Projekt eine eigene Projektdatei anlegen, in der alle zum Projekt gehörigen Übersetzungseinheiten eingetragen sind.
- Paketspezifikationen getrennt von den Paketimplementierungen verwalten.
- Paketspezifikationen gegen unbefugtes Verändern schützen: nur Lesezugriffe erlauben oder Schreib-Paßwort setzen.
- Schnittstellen (Paketspezifikationen) nach dem Design "einfrieren". Notwendige Änderungen nur zu bestimmten Zeitpunkten durchführen und alle Betroffenen davon unterrichten.
- Nach Änderungen an Spezifikationen alle Übersetzungseinheiten neu übersetzen, die die geänderten Spezifikationen direkt oder indirekt importieren.
- Alle Bindemodule von ausgetesteten Paketimplementierungen in Bindemodulbibliotheken zugänglich machen.
- Zum Modifizieren einer Paketimplementierung sollte ein Entwickler auf einer eigenen Kopie der Quelle arbeiten. Zum Austesten werden die Objektmodule des modifizierten Pakets mit den anderen, zum Programm gehörigen Objektmodulen, aus den zentralen Bindemodulbibliotheken zusammengebunden.

Beispiel: Arbeiten mit der Projektdatei

In diesem Beispiel wird der Umgang mit der Projektdatei anhand der vier Pakete A, B, C und D gezeigt. Die Spezifikation von B importiert das Paket A, die Spezifikation von C importiert das Paket B und die Implementierung von C importiert das Paket D.

Im Bild 3-1 sind die Beziehungen zwischen den Paketen durch Pfeile angegeben. Ein Pfeil zeigt dabei immer in Richtung der importierten Spezifikation. Zur Unterscheidung der Übersetzungseinheiten werden die Paketnamen mit dem Suffix ".SPEC" (für Spezifikation) und mit ".BODY" (für Implementierung) gekennzeichnet.

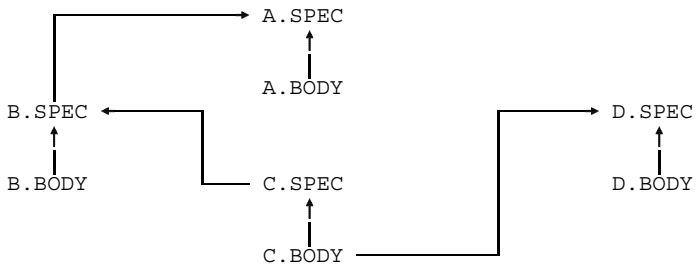


Bild 3-1 Beziehungen zwischen Paketen

Die Quelldateien der Paketspezifikationen seien in der PLAM-Bibliothek PLAM.SPEC, die Paketimplementierungen in PLAM.BODY abgespeichert.

```
/EXEC $USERID.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ... LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//D TEST.DIRECTORY ------------------------------------------------------------ (01)
   >>> PROJECT DIRECTORY FILE CREATED
//MC (PLAM.SPEC,),*D,CH=ON,IN=ON,OP=ON
//C (,A.SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,B.SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,C.SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,D.SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//MC (PLAM.BODY)
//C (,A.BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,B.BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,C.BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//C (,D.BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
```

(01) Um eine Projektdatei für dieses Programmsystem zu erzeugen, sind zwei Schritte notwendig:
- zuerst mit der DEFINE-Anweisung eine Projektdatei einstellen,
- anschließend mit der COMPILE-Anweisung alle Pakete fehlerfrei übersetzen.

```
//S P ------------------------------------------------------------------ (02)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (TEST.DIRECTORY)

PACKAGE NAME    KIND  LOCATION
A              (SPEC) ($USERID.PLAM.SPEC,A.SPEC(*UPPER-LIMIT,S))
A              (BODY) ($USERID.PLAM.BODY,A.BODY(*UPPER-LIMIT,S))
B              (SPEC) ($USERID.PLAM.SPEC,B.SPEC(*UPPER-LIMIT,S))
B              (BODY) ($USERID.PLAM.BODY,B.BODY(*UPPER-LIMIT,S))
C              (SPEC) ($USERID.PLAM.SPEC,C.SPEC(*UPPER-LIMIT,S))
C              (BODY) ($USERID.PLAM.BODY,C.BODY(*UPPER-LIMIT,S))
D              (SPEC) ($USERID.PLAM.SPEC,D.SPEC(*UPPER-LIMIT,S))
D              (BODY) ($USERID.PLAM.BODY,D.BODY(*UPPER-LIMIT,S))
//S P (,,A,A) ---------------------------------------------------------- (03)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (TEST.DIRECTORY)

PACKAGE NAME    KIND  LOCATION

A              (SPEC) ($USERID.PLAM.SPEC,A.SPEC(*UPPER-LIMIT,S))
    REFERENCES
      direct:   - -
      indirect: - -
    USED BY
      direct:   A (OWN BODY), B (SPEC)
      indirect: B (BODY), C (SPEC), C (BODY)

A              (BODY) ($USERID.PLAM.BODY,A.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   A (OWN SPEC)
      indirect: - -

B              (SPEC) ($USERID.PLAM.SPEC,B.SPEC(*UPPER-LIMIT,S))
    REFERENCES
      direct:   A
      indirect: - -
    USED BY
      direct:   B (OWN BODY), C (SPEC)
      indirect: C (BODY)

B              (BODY) ($USERID.PLAM.BODY,B.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   B (OWN SPEC)
      indirect: A
```

```
C               (SPEC) ($USERID.PLAM.SPEC,C.SPEC(*UPPER-LIMIT,S))
    REFERENCES
      direct:   B
      indirect: A
    USED BY
      direct:   C (OWN BODY)
      indirect: - -

C               (BODY) ($USERID.PLAM.BODY,C.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   C (OWN SPEC), D
      indirect: A, B

D               (SPEC) ($USERID.PLAM.SPEC,D.SPEC(*UPPER-LIMIT,S))
    REFERENCES
      direct:   - -
      indirect: - -
    USED BY
      direct:   C (BODY), D (OWN BODY)
      indirect: - -

D               (BODY) ($USERID.PLAM.BODY,D.BODY(*UPPER-LIMIT,S))
    REFERENCES
      direct:   D (OWN SPEC)
      indirect: - -
```

(02) Nun kann mit der SHOW-Anweisung der Inhalt der Projektdatei ausgegeben werden. Die Standardausgabe besteht aus dem Paketnamen, der Angabe, ob es sich um eine Paketspezifikation (SPEC), eine Paketimplementierung (BODY) oder um ein Hauptprogramm (PROG) handelt, und aus dem Dateinamen, oder bei Bibliothekselementen aus dem Bibliotheksnamen, dem Namen des Elements, der Version und dem Typ des Elements. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = PROJECT-FILE
```

(03) Mit der SHOW-Anweisung werden nun zusätzlich zur Standardausgabe auch die Beziehungen zwischen den Übersetzungseinheiten ausgegeben. Unter der Überschrift REFERENCES werden die importierten Pakete aufgeführt. Als direkt importiert werden jene Pakete betrachtet, die direkt in der WITH-Klausel angegeben wurden, bei Paketimplementierungen außerdem noch die eigene Spezifikation. Indirekt importiert werden all jene Pakete, die von den Spezifikationen der direkt importierten Pakete importiert werden (auch über mehrere Stufen hinweg). Unter der USED-BY-Relation werden diejenigen Übersetzungseinheiten aufgelistet, die dieses Paket importieren. Für direkt und indirekt gelten sinngemäß die obigen Definitionen. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = PROJECT-FILE (USED-BY = ALL,
                                     REFERENCES = ALL)
```

```
//E A(S) ------------------------------------------------------------------ (04)
   >>> (OVER)WRITE PACKAGE SPEC "A" (y/n) ?
*y
   >>> PACKAGE SPEC "A" (OVER)WRITTEN
//S P ------------------------------------------------------------------- (05)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (TEST.DIRECTORY)

PACKAGE NAME    KIND  LOCATION
A              !(SPEC) ($USERID.PLAM.SPEC,A.SPEC(*UPPER-LIMIT,S))
A              !(BODY) ($USERID.PLAM.BODY,A.BODY(*UPPER-LIMIT,S))
B              !(SPEC) ($USERID.PLAM.SPEC,B.SPEC(*UPPER-LIMIT,S))
B              !(BODY) ($USERID.PLAM.BODY,B.BODY(*UPPER-LIMIT,S))
C              !(SPEC) ($USERID.PLAM.SPEC,C.SPEC(*UPPER-LIMIT,S))
C              !(BODY) ($USERID.PLAM.BODY,C.BODY(*UPPER-LIMIT,S))
D               (SPEC) ($USERID.PLAM.SPEC,D.SPEC(*UPPER-LIMIT,S))
D               (BODY) ($USERID.PLAM.BODY,D.BODY(*UPPER-LIMIT,S))

//S R ------------------------------------------------------------------- (06)
COMPILE A (SPEC)
COMPILE A (BODY)
COMPILE B (SPEC)
COMPILE B (BODY)
COMPILE C (SPEC)
COMPILE C (BODY)
```

(04) Mit dem Editor wird die Spezifikation A verändert und zurückgeschrieben. In der Projektdatei wird die Spezifikation des Pakets A als geändert markiert.

(05) Nun sind in der Ausgabe der Projektdatei alle Übersetzungseinheiten mit einem Ausrufungszeichen markiert, die wegen der Änderung der Spezifikation von A neu übersetzt werden müssen. Es sind also alle Übersetzungseinheiten betroffen, die das Paket A direkt oder indirekt importieren.

(06) Für diese markierten Übersetzungseinheiten werden mit Hilfe der SHOW-Anweisung die erforderlichen COMPILE-Anweisungen erzeugt. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = RECOMPILATIONS
```

```
//S R,*E(8) ——————————————————————————————————————————————— (07)
//CA *E(8) ———————————————————————————————————————————————— (08)
(%STMT) COMPILE A (SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
(%STMT) COMPILE A (BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
(%STMT) COMPILE B (SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
(%STMT) COMPILE B (BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
(%STMT) COMPILE C (SPEC)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
(%STMT) COMPILE C (BODY)
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;  NOTES:  0)
//S P ————————————————————————————————————————————————— (09)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (TEST.DIRECTORY)

PACKAGE NAME    KIND  LOCATION
A              (SPEC) ($USERID.PLAM.SPEC,A.SPEC(*UPPER-LIMIT,S))
A              (BODY) ($USERID.PLAM.BODY,A.BODY(*UPPER-LIMIT,S))
B              (SPEC) ($USERID.PLAM.SPEC,B.SPEC(*UPPER-LIMIT,S))
B              (BODY) ($USERID.PLAM.BODY,B.BODY(*UPPER-LIMIT,S))
C              (SPEC) ($USERID.PLAM.SPEC,C.SPEC(*UPPER-LIMIT,S))
C              (BODY) ($USERID.PLAM.BODY,C.BODY(*UPPER-LIMIT,S))
D              (SPEC) ($USERID.PLAM.SPEC,D.SPEC(*UPPER-LIMIT,S))
D              (BODY) ($USERID.PLAM.BODY,D.BODY(*UPPER-LIMIT,S))

//REM B ————————————————————————————————————————————————— (10)
```

(07) Damit die erzeugten COMPILE-Anweisungen mit Hilfe der CALL-STATEMENT-FILE-Anweisung ausgeführt werden können, werden sie in den Arbeitsbereich 8 des EDT ausgegeben. Die ausführliche Schreibweise dieser Anweisung lautet:

```
SHOW-ATTRIBUTES WHAT = RECOMPILATIONS,
                OUTFILE = *EDT (WORKFILE = 8)
```

(08) Die Anweisungen im Arbeitsbereich 8 des EDT werden mit der CALL-STATEMENT-FILE-Anweisung ausgeführt. Die ausführliche Schreibweise dieser Anweisung lautet:

```
CALL-STATEMENT-FILE STMT-FILE = *EDT (WORKFILE = 8)
```

(09) Es sind keine Übersetzungen mehr erforderlich, da keine Übersetzungseinheiten mehr markiert sind.

(10) Mit der REMOVE-Anweisung werden alle Einträge für den Paketnamen B gelöscht, also sowohl der Eintrag für die Spezifikation als auch der Eintrag für die Implementierung. Die ausführliche Schreibweise dieser Anweisung lautet:

```
REMOVE-DIRECTORY-ENTRY UNIT = B
```

```
//S P ———————————————————————————————————————————————————————————— (11)

     CONTENTS OF THE PROJECT DIRECTORY FILE   (TEST.DIRECTORY)

PACKAGE NAME    KIND  LOCATION
A              (SPEC) ($USERID.PLAM.SPEC,A.SPEC(*UPPER-LIMIT,S))
A              (BODY) ($USERID.PLAM.BODY,A.BODY(*UPPER-LIMIT,S))
C             !(SPEC) ($USERID.PLAM.SPEC,C.SPEC(*UPPER-LIMIT,S))
C             !(BODY) ($USERID.PLAM.BODY,C.BODY(*UPPER-LIMIT,S))
D              (SPEC) ($USERID.PLAM.SPEC,D.SPEC(*UPPER-LIMIT,S))
D              (BODY) ($USERID.PLAM.BODY,D.BODY(*UPPER-LIMIT,S))

//END
   END OF THE PASCAL SESSION  -  USED TIME = 2.965 SECONDS
%  EXC0732 ABNORMAL PROGRAM TERMINATION. ERROR CODE 'NRT0101': /HELP NRT0101,INF=D
```

(11) Die Spezifikation und die Implementierung des Pakets C müssen geändert und neu übersetzt werden, da sie direkt bzw. indirekt die nicht mehr existierende Spezifikation B importiert haben.

Der Pascal-XT Compiler

Benutzung der Projektdatei

Der Compiler benötigt die Projektdatei zum Auffinden der Spezifikationen importierter Pakete, um die Schnittstellen überprüfen zu können und zum Abspeichern der Paket- und Datei-Namen für Spezifikationen. Weitergehende Funktionen der Projektdatei (siehe Kapitel 3) erleichtern lediglich das Arbeiten mit Paketen.

In welchem Fall muß eine Projektdatei eingestellt werden?

Wenn eine Übersetzungseinheit übersetzt werden soll, die Pakete importiert, muß der Benutzer eine Projektdatei einstellen. Der Compiler benötigt die Projektdatei zum Auffinden der Spezifikationen der importierten Pakete. Wenn keine Projektdatei eingestellt ist, wird in der Übersetzungsliste bei den Paketbezeichnern in der WITH-Klausel folgender Fehler gemeldet:

```
PACKAGE SPECIFICATION NOT FOUND
```

Welche Einträge werden in der Projektdatei erzeugt?

Wenn eine Übersetzungseinheit fehlerfrei übersetzt worden ist, speichert der Compiler den Paket- bzw. Programm-Namen, den Namen der Quelldatei und weitere Statusinformationen in der Projektdatei ab. Wenn eine Paketspezifikation übersetzt werden soll und keine Projektdatei eingestellt ist, wird folgende Meldung ausgegeben:

```
NO PROJECT DIRECTORY IS DEFINED
```

Wird die Übersetzungseinheit aus einem EDT-Arbeitsbereich gelesen, wird in der Projektdatei kein Eintrag erzeugt.

In welcher Reihenfolge werden die Übersetzungseinheiten übersetzt?

Paket-Spezifikationen müssen fehlerfrei übersetzt worden sein, bevor sie von anderen Übersetzungseinheiten verwendet werden dürfen (analog zu der Pascal-Regel, daß Bezeichner erst nach ihrer Deklaration verwendet werden dürfen). Die WITH-Klauseln stellen zwischen den Paketen eine statische Beziehung her, durch die automatisch festgelegt ist, in welcher Reihenfolge Paket-Spezifikationen übersetzt werden müssen. Es ist zu beachten, daß die Spezifikationen der importierten Pakete immer zuerst übersetzt werden müssen. Für Paket-Implementierungen ist keine Übersetzungsreihenfolge vorgegeben.

Implementierungsdefinierte Eigenschaften

Hier werden die in der Sprachbeschreibung ([1], Anhang A.6) aufgeführten implementierungs-definierten Eigenschaften definiert und die ab V2.2A geltenden neuen hochgenauen mathematischen Routinen erläutert . Die Numerierung ist dieselbe wie in der Sprachbeschreibung.

1) Werte der vordefinierten Real-Konstanten:

```
Long_Minreal  =   5.397605346934027E-79
Long_Maxreal  =   7.237005577332262E+75
Short_Minreal =   5.397605E-79
Short_Maxreal =   7.237005E+75
```

2) Voreinstellungen der vordefinierten Konstanten:

```
Maxint  = Long_Maxint   ( = 2^31 - 1)
Minint  = Long_Minint   ( = - 2^31)
Minreal = Long_Minreal
Maxreal = Long_Maxreal
```

3) Vordefinierte Typen:

```
Integer  =  Long_Integer
Real     =  Long_Real
```

4) Werte des Typs Short_Real werden als 32-Bit Gleitpunktzahlen, Werte des Typs Long_Real werden als 64-Bit Gleitpunktzahlen dargestellt, wie im Manual Assemblerbefehle [12] beschrieben. Der absolute Wertebereich W für beide Real-Typen ist:

```
-Maxreal <= W <= Maxreal
```

5) Die Genauigkeit für Real-Operationen und Real-Funktionen ist durch die Repräsentation [12] und die Rundung bei Zwischenergebnissen vorgegeben. Die Anzahl der signifikanten Dezimalstellen ist für Werte vom Typ

```
Long_Real:   ca. 16
Short_Real:  ca. 6
```

Exponentenunterlauf wird nicht erkannt, Exponentenüberlauf wird bei arithmetischen Operationen als Numeric_Error und beim Einlesen einer Real-Zahl als Read_Error gemeldet.

Hochgenaue mathematische Routinen:

Pascal-XT stellt folgende arithmetische Funktionen zur Verfügung (siehe Sprachbeschreibung [1], Kap. 15.4):

```
Arctan(x)                Ln(x)
Cos(x)                   Sin(x)
Exp(x)                   Sqrt(x)
```

Ab Version V2.2A verwenden sie hochgenaue mathematische Routinen, die ihre Berechnungen mit sog. ulp-genauen (ulp = *u*nit of *l*ast *p*lace) Algorithmen durch-

führen. Bei diesen Routinen wird eine maximale Genauigkeit garantiert, d.h. zwischen dem errechneten Funktionsergebnis und dem exakten Ergebnis existiert keine darstellbare Gleitpunktzahl. Trotz dieser erheblich gesteigerten Genauigkeit bieten die neuen Routinen eine mindestens ebenso gute Performance wie die alten.

Da diese neuen Routinen andere Funktionswerte als früher liefern können, kann es zu unerwartetem Programmverhalten kommen - beispielsweise wenn früher erstellte Programme die Funktionswerte in Vergleichen verwenden.

6) Die Werte des Typs Char sind die Elemente des Zeichensatzes EBCDIC SN77315 (siehe [5]). In der Tabelle sind auch die Zuordnungen der Ordinalzahlen zu den Zeichenwerten angegeben.

7) Der Typ String ohne Typ-Parameter besitzt die Maximallänge 254.

8) Die Größe einer Speichereinheit ist ein Byte (8 Bit).

9) Die Einschränkungen für die Abstand- und Bitbereichsangaben in einem Feld-Bezeichner sind im Abschnitt 4.3 beschrieben.

10) Der Basistyp einer Menge kann maximal zwischen 2041 und 2048 Elemente enthalten (siehe 4.3).

11) In einem nicht qualifizierten Mengenbildner müssen die Ordinalwerte der Elemente im Bereich 0 .. 2047 liegen.

12) Ab Pascal-XT V2.2A wird auch die Direktive "C" unterstützt. Es wird jedoch empfohlen, für fremdsprachige Unterprogramme nur die Direktive "EXTERNAL" zu verwenden (siehe 7.2).

13) Für Unterprogramme mit einer Direktive sind nicht alle Pascal-XT-Parametertypen zulässig. Die geltenden Einschränkungen sind in den Abschnitten 7.2 bis 7.4 beschrieben.

14) Die Datei-Operationen, die bei Rewrite, Put, Reset und Get ausgeführt werden, sind in Abschnitt 5.4 beschrieben.

15) Die Beschreibung einer externen Datei in der vordefinierten Prozedur Assignfile und die Wirkung dieser Prozedur sind in Abschnitt 5.3.3 angegeben.

16) Die Anwendung von Reset oder Rewrite auf die vordefinierten Textdateien Input oder Output führt zu einem Open_Error mit dem Systemfehlercode 1607.

17) Die Standardausgabelänge ist für

```
Integer-Werte  : 12
Real-Werte     : 22
Boolesche Werte:  5.
```

Der boolesche Wert TRUE wird rechtsbündig in dem Ausgabefeld ausgegeben.

18) Bei Ausgabe von Real-Werten in Gleitpunktdarstellung

   – wird als Exponentenzeichen ’E’ ausgegeben und
   – für den Exponenten werden ein Vorzeichen und zwei Dezimalziffern ausgegeben.

19) Boolesche Werte werden in Großbuchstaben ausgegeben (TRUE und FALSE).

20) Die vordefinierte Prozedur Page erzeugt nur dann ein Seitenvorschubsteuerzeichen (Ausgabe des Zeichens ’A’ in der ersten Spalte) in einer Textdatei, wenn bei der Dateizuordnung mit der vordefinierten Prozedur Assignfile das Attribut SPACE=E angegeben wurde (siehe 5.3.3). Für die BS2000-Systemdatei SYSLST wird standardmäßig wie bei SPACE=E verfahren. Mehrere Aufrufe von Page unmittelbar hintereinander haben nur einen Seitenvorschub zur Folge. Ein Aufruf von Page hat keine Wirkung, wenn nach dem Page kein weiterer Text ausgegeben wird. Page hat keine Wirkung, wenn der Pascal-Datei die Systemdatei SYSOUT zugewiesen wurde.

21) Die Einschränkungen für Entry-Unterprogramme sind im Abschnitt 7.3 beschrieben.

22) Die Zuordnung von externen Dateien zu Programmparametern ist in Abschnitt 5.3 beschrieben.

23) Die Voreinstellungen der Compiler-Optionen sind im Abschnitt 4.5 beschrieben. Beim Aufruf der COMPILE-Anweisung kann zusätzlich noch die LINES-PER-PAGE-Option (siehe 2.6.4) angegeben werden.

Compilerinterne Beschränkungen und Systemabhängigkeiten

- Die Zeilenlänge eines Quellprogramms ist auf 254 Zeichen beschränkt. Bei längeren Zeilen wird ein Fehler gemeldet.
- Bezeichner können maximal 254 Zeichen lang sein.
- Es können maximal 50 Records ineinander geschachtelt werden.
- Es können maximal 50 WITH-Anweisungen geschachtelt werden.

Darstellung von Objekten im Hauptspeicher

Der Speicherbedarf und die Ausrichtung ist für Objekte einfacher Typen und Zeigertypen in der Tabelle 4-1 angegeben. Für strukturierte Typen ist die Angabe dieser Größen etwas komplizierter und wird daher nur kurz beschrieben.

Die vordefinierten Funktionen SIZEOF und ALIGNOF liefern Angaben über den Speicherbedarf bzw. die Ausrichtung für Werte eines Typs.

Datentyp T	Wertebereich	Speicherbedarf SIZEOF (T) (in Byte)	Ausrichtung ALIGNOF (T) (in Byte)
BOOLEAN	FALSE..TRUE	1	1
CHAR	CHR(0)..CHR(255)	1	1
SHORT_INTEGER	SHORT_MININT..SHORT_MAXINT	2	2
LONG_INTEGER	LONG_MININT..LONG_MAXINT	4	4
INTEGER	LONG_MININT..LONG_MAXINT	4	4
Aufzählung	bis 256 Elemente	1	1
Aufzählung	bis 32768 Elemente	2	2
Aufzählung	darüber	4	4
Teilbereich	0 .. 255	1	1
Teilbereich	SHORT_MININT..SHORT_MAXINT	2	2
Teilbereich	LONG_MININT..LONG_MAXINT	4	4
LONG_REAL		8	8
SHORT_REAL		4	4
REAL		8	8
Zeiger		4	4

Tabelle 4-1 Speicherbedarf und Ausrichtung einfacher Typen und Zeigertypen

Gepackte Darstellung strukturierter Typen

Bei Objekten strukturierter Typen wird i.a. nicht zwischen gepackter und nicht gepackter Darstellung unterschieden. Eine Ausnahme stellen Record-Typen dar (siehe unten).

Mengen-Typen

Die maximale Anzahl von Elementen einer Menge ist auf 2048 beschränkt. Ist die Untergrenze UG des Basis-Typs der Menge ungleich Null, dann kann die Maximalanzahl im ungünstigsten Fall auf 2041 beschränkt sein.
Der Speicherbedarf einer Menge (in Bytes) berechnet sich in etwa aus der Anzahl der Elemente geteilt durch 8.
Die Ausrichtung erfolgt an einer Bytegrenze.

String-Typen

Der Speicherbedarf ergibt sich aus der maximalen Anzahl der Zeichen und einem Halbwort für das Längenfeld.
Die Ausrichtung erfolgt immer an einer Halbwortgrenze, auch innerhalb gepackter Recordtypen.
Der Speicherbedarf für die Standard-Stringlänge ist 256 Byte.

Array-Typen

Der Speicherbedarf berechnet sich aus der Summe der Elementgrößen und evtl. auftretender Lücken.
Die Ausrichtung ist gleich der Ausrichtung des Elementtyps.

Record-Typen

Der Speicherbedarf ergibt sich aus dem Speicherbedarf der Felder des Festteils, der Felder der größten Variante des Variantteils und evtl. auftretender Lücken (bedingt durch Ausrichtungsanforderungen). Das Feld mit der größten Ausrichtung bestimmt die Ausrichtung des gesamten RECORD-Typs.

Bei gepackten Record-Typen werden Felder von skalaren Typen und Zeiger-Typen auf Byte-Grenzen ausgerichtet. Felder strukturierter Typen werden gemäß den Ausrichtungsanforderungen dieser Typen ausgerichtet (die durch die Angabe packed wiederum beeinflußt werden kann).

Zur Speicherplatzoptimierung belegen Varianten eines Variantteils denselben Speicherplatz. Sofern eine Variante ein Feld eines FILE-Typs enthält, werden alle Varianten hintereinander allokiert. Damit ist ein verkürztes Anlegen auf der Halde mit der Prozedur New nicht mehr möglich.

Enthält ein RECORD-Typ Felder eines FILE-Typs, dann dürfen in dem RECORD-Typ keine Angaben zur Speicherdarstellung gemacht werden.

Werden für Felder keine Angaben zur Speicherdarstellung angegeben, dann kann der Compiler zur besseren Speicherausnutzung die Felder in einer anderen Reihenfolge, als im Text angegeben, allokieren.

Für die Abstandsangaben in einem Feld gelten zusätzliche Einschränkungen:

- Der Abstand eines Feldes von einem String-Typ muß immer ein Vielfaches von 2 sein (ein String-Typ ist ein gepackter Typ).
- Der Abstand eines Feldes vom Typ Long_Real (und damit auch Real) muß immer ein Vielfaches von 8 sein.
- Der Abstand eines Feldes vom Typ Short_Real muß immer ein Vielfaches von 4 sein.
- In einem ungepackten RECORD-Typ muß der Abstand eines Feldes ein ganzzahliges Vielfaches der Ausrichtung seines Typs sein.
- In einem gepackten RECORD-Typ kann für Felder von einem Ordinaltyp oder Zeigertyp ein beliebiger Abstand angegeben werden. Der Abstand eines Felds eines strukturierten Typs muß wiederum ein Vielfaches der Ausrichtung dieses Typs sein.
- Speicherbereiche für Felder in verschiedenen Varianten eines Variantteils dürfen sich überlappen.

Für die Bitbereichsangaben in einem Feld gelten zusätzliche Einschränkungen:

- Bitbereiche können nur für Felder von einem Ordinaltyp angegeben werden.
- Die Werte der Bitbereichsgrenzen müssen im Bereich 0..31 liegen.

Hinweis

Werden Variable eines RECORD-Typs als Schnittstellen zu externen Programmen benutzt, dann sollten für alle Komponenten Speicherdarstellungen angegeben werden. Der Compiler garantiert dann die gewünschte Darstellung, sofern sie den o.g. Forderungen nicht widerspricht.

File-Typen

Der Speicherbedarf berechnet sich aus der Größe des Komponententyps und der Größe eines festen Verwaltungsblocks. Bei Textdateien wird der Zeilenpuffer auf der Halde angelegt.
Die Ausrichtung erfolgt an einer Doppelwortgrenze.

Erzeugte Objektmodule

Der Pascal-XT Compiler erzeugt für eine Übersetzungseinheit Objektmodule (Bindemodule), wenn folgende Bedingungen erfüllt sind:

- Die Übersetzungseinheit ist ein Hauptprogramm (program) oder eine Paketimplementierung (package body).
- Die Übersetzung war fehlerfrei.
- Die Compileroption Generate war eingeschaltet.

Die erzeugten Module werden entsprechend der Angabe im MODULE-Operanden der COMPILE-Anweisung in die temporäre Bindemoduldatei *OMF oder in einer PLAM-Bibliothek abgelegt. Folgende Module werden erzeugt:

Code-Modul
: Das Code-Modul enthält die Konstanten und den Objektcode der Übersetzungseinheit. Er ist ablaufinvariant und mehrfach benutzbar.

Daten-Modul
: Das Daten-Modul enthält die Variablen der Übersetzungseinheit. Dieses Modul ist nicht mehrfach benutzbar.

Testtabellen-Modul
: Das enthält Informationen für die Testhilfe PATH. Das Modul ist ablaufinvariant und mehrfach benutzbar.

Starter-Modul
: Dieses Modul wird nur für ein Pascal-Hauptprogramm erzeugt. Er enthält die Startadresse des Programms und ist nicht mehrfach benutzbar.

XS - Fähigkeit

Die vom Compiler erzeugten Objektmodule und die Module des Laufzeitsystems sind XS-fähig. Für jedes Objektmodul gilt AMODE=ANY und RMODE=ANY. Ein Pascal-XT Programm kann damit überall im 31-Bit Adressraum ablaufen (siehe auch Kap. 6).

Namensgebung für die Objektmodule

Die Namen der erzeugten Objektmodule werden nach folgendem Verfahren generiert:

1.-7. Zeichen — Werden aus dem Programm- bzw. Paketnamen übernommen. Kürzere Namen werden beim Starter-Modul mit Leerzeichen (Blank), bei den Code-, Daten- und Testtabellen-Modulen mit Nummernzeichen ("#") aufgefüllt. Unterstreichungszeichen ("_") werden in Nummernzeichen ("#") umgewandelt.

8. Zeichen — Wird zur Unterscheidung der Module benutzt. Es steht ein:

"C" für Code-Modul,
"D" für Daten-Modul und
"T" für Testtabellen-Modul.
" " für Starter-Modul.

Namensgebung für Prozeduren

Die Namen von externen Unterprogrammen und Entry-Prozeduren sind nach außen sichtbar. Sie sind maximal 8 Zeichen lang. Längere Namen werden abgeschnitten, kürzere Namen mit Leerzeichen (Blank) aufgefüllt. Unterstreichungszeichen ("_") werden in Nummernzeichen ("#") umgewandelt.

Hinweis

In anderen Sprachen sind Unterstriche in Namen im allgemeinen nicht zugelassen. Daher sollten Namen von externen und Entry-Prozeduren in den ersten 8 Zeichen keine Unterstriche enthalten.

Compileroptionen

Für alle Pascal-XT-Implementierungen gelten die Compileroptionen (Steueranweisungen an den Compiler), die in der Sprachbeschreibung von Pascal-XT [1] definiert sind. Sie können alle im Quellprogramm und mit Ausnahme von PAGE und TITLE auch in der COMPILE-Anweisung als Operand angegeben werden. PAGE und TITLE können nur im Quellprogramm angegeben werden.

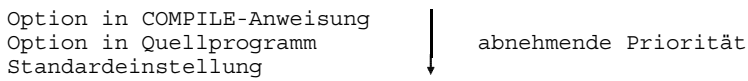
In einem Pascal-XT-Quellprogramm werden Compileroptionen wie in folgendem Beispiel als Pseudokommentare angegeben.

Beispiel

```
{$Check = On, Initialize = On, List = Off}
```

Zusätzlich gibt es noch die Optionen LINES-PER-PAGE und MESSAGE-LEVEL, die nur in der COMPILE-Anweisung angegeben werden können (siehe 2.6.4). LINES-PER-PAGE legt die Anzahl der Zeilen pro Seite in der Übersetzungsliste fest, MESSAGE-LEVEL beeinflußt Art und Umfang der Compilermeldungen.

Wenn eine Option als Operand in der COMPILE-Anweisung angegeben wird, gilt ihre Einstellung für die gesamte Übersetzungseinheit und setzt die Standardeinstellung oder die im Quellprogramm angegebene Einstellung außer Kraft. Welche Einstellung einer Option gültig ist, hängt also davon ab, wo sie angegeben wurde. Für die Eingabe von Optionen gelten folgende Prioritäten.



In der folgenden Tabelle sind nochmals alle Optionen mit den zulässigen Werten alphabetisch aufgelistet. Die Standardwerte sind durch Unterstreichung gekennzeichnet.

Option	zulässige Werte	Bedeutung	Angabe Quelle	mögl. in COMPILE-Anw.
Assembler	On Off	Ausgabe eines Objektcode-Protokolls in Assemblerformat	x	x
Check	On Off	Code für Laufzeitüberprüfungen erzeugen	x	x
Debug	On Restricted Off	Erzeugen von Testinformationen für die Pascal-Testhilfe PATH	x	x
Generate	On Off	Erzeugen von Objektcode	x	x
Initialize	On Off	Speicherbereiche für Variable werden mit dem Sedezimalwert 88 besetzt.	x	x
Lines-per-page	11 bis 2147483639 63	Festlegen der Zeilenanzahl pro Seite in der Übersetzungsliste		x
List	On Off	Ein-/Ausschalten des Übersetzungsprotokolls	x	x
Map	On Off	Erzeugen von Adreßtabellen	x	x
Message-level	Errors Warnings Notes	Festlegen von Art und Umfang der Compilermeldungen in der Übersetzungsliste		x
Optimize	On Off	Erzeugen von optimiertem Objektcode	x	x
Page		Erzeugen eines Seitenvorschubs	x	
Standard	On L0 Off	Bei On wird Standard Pascal Level 1, bei L0 wird Standard Pascal Level 0 und bei Off wird Pascal-XT akzeptiert	x x	x x x
Title	Zeichen-kette ''	Angabe eines Titels in der Kopfzeile des Übersetzungsprotokolls	x	
Xref	On Off	Erzeugen einer Querverweisliste	x	x

Tabelle 4-2 Optionen des Pascal-XT-Compilers im BS2000

Vom Compiler erzeugte Listen

Der Pascal-XT Compiler erzeugt während der Übersetzung eine Reihe von Listen mit Informationen über die Struktur des Quellprogramms, der erzeugten Objektmodule und den Ablauf der Übersetzung. Beispiele für die verschiedenen Listen sind im Anhang A.2 angegeben.

Alle Listen besitzen eine einheitliche Überschrift, die aus

- Bezeichnung der gewünschten Liste
- Angabe des Betriebssystems und des Compilers
- Version und Datum des Pascal-XT Compilers
- Datum und Uhrzeit des Beginns der Übersetzung
- Fortlaufende Seitennumerierung

besteht. Sie wird am Anfang jeder Seite ausgegeben. Ab Version V2.2A werden bei der Datumsangabe in der Kopfzeile der Übersetzungsliste Jahreszahlen vierstellig ausgegeben.

Steuerung der Listenausgabe

Alle vom Compiler erzeugten Listen werden auf eine Ausgabedatei ausgegeben. Diese Datei wird durch den Operanden LISTING in der COMPILE-Anweisung festgelegt (siehe 2.6.4).

Das Erzeugen der verschiedenen Listen wird durch die Compileroptionen Assembler, List, Map und Xref gesteuert. Die Bedeutung der Optionen ist im Abschnitt 4.5 beschrieben.

Die Anzahl der Zeilen pro Seite in der Übersetzungsliste ist standardmäßig 63 und kann mit der Lines-Per-Page Option verändert werden.

Übersetzungsliste

Die Übersetzungsliste ist in die Bereiche Prolog, Quellprogrammliste und Compilation Summary eingeteilt. Die Ausgabe dieser Liste wird durch die Option List gesteuert. Standardmäßig wird eine Übersetzungsliste erzeugt (List=On). Bei List=Off werden nur der Prolog und das Compilation Summary ausgegeben.

Prolog

Im Prolog werden die globalen Informationen der Übersetzungseinheit zusammengefaßt.

GLOBAL OPTIONS FOR THIS COMPILATION
: Gibt die Werte der Optionen an, die global für die gesamte Übersetzungseinheit gelten. Die Debug-Option schaltet die Optimize-Option aus, wenn entweder beide in der Quelle oder beide in der COMPILE-Anweisung angegeben wurden. Hinter den Optionen wird aufgeführt, wo sie angegeben wurden:

BY COMMAND	in der COMPILE-Anweisung
IN SOURCE	innerhalb des Quellprogramms
BY DEFAULT	Standardwert (siehe 4.5)
BY DEBUG OPTION	die Optimize-Option wurde durch die Debug-Option ausgeschaltet
BY OPTIMIZE OPTION	die Debug-Option wurde durch die Optimize-Option ausgeschaltet (durch Angabe in der COMPILE-Anweisung)

LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)
: Es werden die Dateien angegeben, aus denen die direkt und indirekt importierten Paketspezifikationen gelesen werden. Indirekt importierte Spezifikationen tauchen nicht in der Kontext-Spezifikation der Übersetzungseinheit auf, werden aber durch mindestens eine der dort angegebenen Spezifikationen importiert. Diese Liste wird nur ausgegeben, wenn in der Übersetzungseinheit Paketspezifikationen importiert werden.

CURRENT COMPILATION UNIT (SOURCE FILE)
: gibt die Datei an, aus der die Übersetzungseinheit gelesen wird.

Quellprogrammliste

Im Anschluß an den Prolog wird das Quellprogramm mit einer bei 1 beginnenden fortlaufenden Zeilennumerierung ausgegeben. Die Zeilennummern in der Übersetzungsliste stimmen mit der Zeilennumerierung des Quellprogramms überein, auch wenn innerhalb des Quellprogramms die Listenausgabe zeitweilig ausgeschaltet wird.
Erstreckt sich ein Kommentar im Quellprogramm über mehrere Zeilen, dann werden alle Folgezeilen des Kommentars durch das Zeichen "C" unmittelbar hinter der Zeilennummer gekennzeichnet.

Bei Fehlern, Warnungen und Hinweisen (Notes) enthält die Quellprogrammliste zusätzlich Meldungszeilen, die keinen Einfluß auf die Zeilennumerierung haben. Der Aufbau der Meldungszeilen ist im Abschnitt 4.6.3 beschrieben.

Compilation Summary

Im Compilation Summary werden die Ergebnisse der Übersetzung zusammengefaßt. Es wird auch die Anzahl der Meldungen ausgegeben, die durch die Compileroption MESSAGE-LEVEL unterdrückt wurden.

ERRORS DETECTED
: Anzahl der Syntax- und Semantik-Fehler.

WARNINGS
: Anzahl der Warnungen.

NOTES
: Anzahl der Notes (Hinweise).

SIZE OF CODE MODULE
: Speicherbedarf des erzeugten Code-Moduls in Byte. Es wird der Wert 0 ausgegeben, wenn Generate=Off gilt, wenn Übersetzungsfehler gemeldet wurden oder wenn die Übersetzungseinheit eine Paket-Spezifikation war.

SIZE OF DATA MODULE
: Speicherbedarf des erzeugten Daten-Moduls in Byte. Es wird der Wert 0 ausgegeben, wenn Generate=Off gilt, wenn Übersetzungsfehler gemeldet wurden oder wenn die Übersetzungseinheit eine Paket-Spezifikation war.

COMPILATION TIME
: Gibt die benötigte CPU-Zeit für die Übersetzung und das Erzeugen der verschiedenen Listen an. Werden bei der Übersetzung Spezifikationen referiert, dann geht die Zeit für das nochmalige Lesen dieser Spezifikationen ebenfalls in die Übersetzungszeit ein.

Fehler, Warnungen und Hinweise

Während des Übersetzungsvorgangs kann der Compiler Fehler, Warnungen und Hinweise ausgeben, die in der Übersetzungsliste erscheinen.

Fehler (Errors) beziehen sich auf Syntax- und Semantikfehler der Übersetzungseinheit.

Warnungen (Warnings) werden an solchen Stellen ausgegeben, an denen der Compiler bereits erkennt, daß zur Laufzeit des Programms an diesen Stellen ein Fehler auftreten wird. Zum Beispiel erkennt der Compiler in der Anweisung

```
x := long_maxint + 1
```

daß bei der Addition ein Überlauf auftreten wird und gibt daher an dieser Stelle eine WARNING aus. Bei Auftreten von WARNINGS wird trotzdem ein ablauffähiges Objekt erzeugt.

Mit Hinweisen (Notes) werden Fehler in Pseudokommentaren und deklarierte, aber nicht verwendete Bezeichner gemeldet. Bei Auftreten von NOTES wird trotzdem ein ablauffähiges Objekt erzeugt. Fehler in Pseudokommentaren führen dazu, daß Compileroptionen nicht die gewünschte Wirkung zeigen. Deklarierte und nicht verwendete Bezeichner werden nach folgenden Regeln gemeldet:

- Alle im Programmtext deklarierten, aber nicht verwendeten Bezeichner werden markiert. Dies gilt für Bezeichner von Konstanten, Typen, Variablen, Funktionen, Prozeduren und Formalparameter. Ausgenommen davon sind Feldbezeichner (Komponenten von Record-Typen) und die Konstantenbezeichner von Aufzählungstypen.
- Die Bezeichner in den WITH- und USE-Klauseln der Hauptübersetzungseinheit oder der zugehörigen Paketspezifikation werden gemeldet, wenn sie im Programmtext nicht verwendet werden. Dies gilt nicht, wenn die Hauptübersetzungseinheit eine Paket-Spezifikation ist.
- Funktionsbezeichner werden markiert, wenn kein Aufruf der Funktion im Programmtext enthalten ist. Die Zuweisung an den Funktionsbezeichner innerhalb des Funktions-Blocks gilt nicht als Verwendung des Funktionsbezeichners.
- Der Bezeichner eines Aufzählungstyps gilt bereits dann als verwendet, wenn eine der Aufzählungskonstanten dieses Typs verwendet wird.
- Für die in einer Paket-Spezifikation deklarierten Bezeichner und die Formalparameter externer Unterprogramme werden keine NOTES ausgegeben.

Die Gesamtzahl der Fehler, Warnungen und Hinweise sowie die Anzahl der unterdrückten Meldungen (Compileroprtion MESSAGE-LEVEL, siehe 2.6.4) wird im COMPILATION SUMMARY angegeben und auf die Systemdatei SYSOUT ausgegeben.

Bei ausgeschalteter List-Option (List=Off) werden nur der Prolog, die Meldungszeilen mit den zugehörigen Quellprogrammzeilen und das Compilation Summary ausgegeben.

Die Spalten der Quellprogrammzeile, auf die sich die Meldungen beziehen, werden in einer eigenen Zeile durch unterschiedliche Ziffern markiert und unmittelbar hinter der Quellprogrammzeile ausgegeben. Zur Hervorhebung in der Quellprogrammliste wird diese Zeile mit dem Zeichen "-" aufgefüllt. Für jede Ziffer werden anschließend ein oder mehrere selbsterklärende Fehlermeldungstexte ausgegeben.

Assemblerliste

In der Assemblerliste wird der generierte Code in Assemblerschreibweise ausgegeben. Die Ausgabe der Liste wird durch die Option Assembler gesteuert und erfolgt im Anschluß an die Übersetzungsliste.
Die Assemblerliste kann für die gesamte Übersetzungseinheit oder für einzelne Unterprogramme erzeugt werden. Im letzteren Fall muß innerhalb des Unterprogrammblocks die Option eingeschaltet und in einer der folgenden Zeilen wieder ausgeschaltet werden.
Für "inline" deklarierte Prozeduren und Funktionen wird keine Assemblerliste erzeugt, da für sie nur an der Aufrufstelle Code erzeugt wird.

Das Format der Assemblerliste entspricht im wesentlichen dem Format, wie es der Assembler erzeugt.

Feld	**Bedeutung**
LOCTN	Relativadresse im Modul (sedezimal in Bytes)
OBJECT CODE	Objektcode in sedezimaler Form
ADDR1 ADDR2	Adressen, die im Befehl verwendet werden. Die Adressen werden aus dem Inhalt der Basisregister und der Distanzen berechnet. Sie geben die Relativadresse im Modul an, entsprechen also den Einträgen in der Spalte LOCTN
STMNT	Fortlaufende Nummerierung in der Liste
ASSEMBLY CODE	Erzeugter Assemblercode

In den Assemblercode werden zusätzlich Kommentarzeilen eingefügt. In diesen wird die Nummer der Quellzeile angegeben, für die die nachfolgenden Befehle erzeugt werden.

Querverweisliste (Cross-Reference-Listing)

Die Querverweisliste enthält alle benutzerdefinierten und verwendeten vordefinierten Bezeichner der Übersetzungseinheit in alphabetischer Reihenfolge. Bezeichner mit gleicher Schreibweise aus verschiedenen Gültigkeitsbereichen treten in der Querverweisliste für jeden Definitionspunkt einmal auf. Zu jedem Bezeichner gehört eine Beschreibungszeile, gefolgt von einer Liste von Zeilennummern, in denen dieser Bezeichner auftritt.

Das Erzeugen der Querverweisliste wird durch die Option Xref gesteuert. Die Liste wird im Anschluß an die Übersetzungsliste ausgegeben.

Die Beschreibungszeile besteht aus vier Feldern. Die Werte der Felder 2 und 3 sind in Abhängigkeit vom Feld 1 zu interpretieren. Die Felder bedeuten:

Feld 1	Klasse des Bezeichners
Feld 2	Typ des Bezeichners
Feld 3	Klassenspezifisches Attribut
Feld 4	Definitionspunkt des Bezeichners

Klassen von Bezeichnern

CONSTANT	Konstantenbezeichner
FIELD	Feldbezeichner in einem Recordtyp
FUNCTION	Funktionsbezeichner
FUNC-PARAM	Bezeichner einer formalen Funktion
PACKAGE	Paketbezeichner
PROCEDURE	Prozedurbezeichner
PROC-PARAM	Bezeichner einer formalen Prozedur
TYPE	Typbezeichner
VAL-PARAM	Bezeichner eines formalen Werteparameters
VAR-PARAM	Bezeichner eines formalen Variablenparameters
VARIABLE	Variablenbezeichner

Typ des Bezeichners

In dieser Spalte wird der Typ des Bezeichners angegeben. Für Bezeichner der Klassen PACKAGE und PROCEDURE entfällt diese Angabe. Für Bezeichner der Klasse FUNCTION und FUNC-PARAM bezeichnet die Typangabe den Typ des Funktionsergebnisses.

Klassenspezifisches Attribut

In diesem Feld wird ein Wert ausgegeben, der in Abhängigkeit der Klasse des Bezeichners zu interpretieren ist.

Klasse	Bedeutung des Feldes 3
CONSTANT	Für Ordinalkonstanten wird deren Wert ausgegeben, ansonsten entfällt die Angabe
FIELD	Gibt den relativen Abstand zum Recordanfang in Bytes an
FUNCTION PROCEDURE	} Gibt die Zeilennummer der ersten Anweisung im Block der } Funktion bzw. Prozedur an
TYPE	Speicherbedarf des Typs in Bytes
sonstige	entfällt

Definitionspunkt des Bezeichners

In dieser Spalte wird die Nummer der Zeile angegeben, in der der Bezeichner vereinbart wurde. Ist der Bezeichner in einem fremden Paket (oder der eigenen Paketspezifikation) vereinbart worden, dann wird zusätzlich zur Zeilennummer der Name des Paketes angegeben. Handelt es sich um einen vordefinierten Bezeichner, dann wird "PREDEFINED" ausgegeben.

Adreßliste (Map-Listing)

Die Adreßliste liefert Informationen über den Inhalt der erzeugten Objektmodule der Übersetzungseinheit. Die Ausgabe dieser Liste wird durch die Option Map gesteuert.

Adressen von Prozeduren und Funktionen

Für die Prozeduren und Funktionen der Übersetzungseinheit wird ausgegeben (von links nach rechts):

Einsprungadresse	Enthält die Anfangsadresse der Prozedur (Funktion)
Anfangsadresse	des Prozedurcodes relativ zum Anfang des Code-Moduls
Name	der Prozedur bzw. Funktion

Globale Konstanten

Alle strukturierten Konstanten der Übersetzungseinheit werden ausgegeben. Die Angaben bedeuten (von links nach rechts):

Relativadresse	der Konstante im Code-Modul. Die Angabe erfolgt in dezimaler und sedezimaler Form
Typ	der Konstanten
Bezeichner	Für deklarierte Konstanten wird der Konstantenbezeichner, für nicht deklarierte Konstanten wird der Leerstring angegeben
Wert	der Konstanten. Bei Mengen werden nur die Werte des kleinsten und größten Elements angegeben

Globale Variable

Es werden die in der Übersetzungseinheit global vereinbarten Variablen ausgegeben. Die angegebenen Adressen (in dezimaler und sedezimaler Form) sind Relativadressen im erzeugten Daten-Modul.

Dateien

Pascal-Dateien

Externe Pascal-Dateien

Externe Pascal-Dateien sind Dateivariable, die durch Angabe ihres Variablennamens in der Programmparameterliste als externe, vom Programm unabhängig existierende Dateien angemeldet werden. Diesen Dateivariablen müssen beim Ablauf des Programms BS2000-Dateien zugeordnet werden.

Eine Ausnahme bilden die vordefinierten Textdateien Input und Output, denen standardmäßig die Systemdateien SYSDTA bzw. SYSOUT zugeordnet sind.

Im Abschnitt 5.2 werden die BS2000-Dateien beschrieben, die den externen Dateien zugeordnet werden können. Im Abschnitt 5.3 wird beschrieben, wie die Zuordnungen hergestellt werden können.

Lokale Pascal-Dateien

Lokale Pascal-Dateien sind Dateivariable oder Komponenten von Variablen, die nicht als Programmparameter auftreten. Die Lebensdauer einer lokalen Datei entspricht der Lebensdauer des Blocks, in dem die Datei (Filevariable) vereinbart wurde. Nach Verlassen eines Blocks kann auf dessen lokale Dateien nicht mehr zugegriffen werden. Lokale Dateien in dynamischen Variablen existieren solange, wie Zeiger auf diese Variablen existieren oder die dynamische Variable mit der vordefinierten Prozedur Dispose vernichtet wird. Beim rekursiven Aufruf einer Prozedur (Funktion) mit einer lokalen Datei wird für jede Inkarnation des Aufrufs eine neue lokale Datei angelegt.

Unterstützte BS2000-Dateien und Bibliotheken

Das Pascal-XT System unterstützt Zugriffe auf folgende BS2000-Dateien und Bibliotheken:

- Standard-Dateien
- Sequentielle (SAM) und indexsequentielle (ISAM) Dateien
- PLAM-Bibliothekselemente
- temporäre Dateien

Im Abschnitt 5.3 werden die verschiedenen Möglichkeiten der Zuordnung von BS2000-Dateien an Pascal-Dateien beschrieben.

Eröffnungs-Modi (Open-Modi) für Dateien

Zu den vordefinierten Unterprogrammen Reset und Rewrite werden im vordefinierten Paket DMSIO (A.5) noch die Prozeduren Extend und Replace angeboten. In den Abschnitten 5.2.1 bis 5.2.4 werden die jeweils zulässigen Prozeduren (Eröffnungsmodi) angegeben.

Reset	Die Datei bzw. das Bibliothekselement wird zum Lesen eröffnet. Existiert die Datei, das Element oder die Bibliothek nicht, dann wird ein Laufzeitfehler (OPEN_ERROR) erzeugt.
Rewrite	Die Datei bzw. das Bibliothekselement wird zum Schreiben eröffnet. Ein bereits existierendes Element wird überschrieben. Existiert das Element nicht, dann wird es angelegt.
Extend	Diese Prozedur wird im vordefinierten Paket DMSIO (A.5) angeboten. Sie hat die Wirkung wie Rewrite, nur daß der Inhalt einer bereits existierenden Datei nicht verloren geht.
Replace	Diese Prozedur wird im vordefinierten Paket DMSIO (A.5) angeboten und eröffnet eine Datei zum Lesen und Schreiben.

Zulässige Eröffnungs-Modi für die verschiedenen Dateien

Die beiden Tabellen 5-1 und 5-2 geben die möglichen Kombinationen von Dateien und Eröffnungs-Modi an. Eine gültige Kombination ist mit "ok" gekennzeichnet, für ungültige Kombinationen ist der gemeldete Fehler mit dem zugehörigen Systemfehler-Code (system_code) angegeben.

Ist die Pascal-Dateivariable f vom Typ "Text", dann gilt:

	Reset(f)	Rewrite(f)	Replace(f)	Extend(f)
*SYSDTA	ok	Open_Error (1605)	Open_Error (1605)	Open_Error (1605)
*SYSLST *SYSOUT	Open_Error (1605)	ok	Open_Error (1605)	ok
*DUMMY *EDT SAM-Datei ISAM-Datei	ok	ok	Open_Error (1605)	ok
PLAM-Elem. EAM-Datei	ok	ok	Open_Error (1605)	Open_Error (1605)

Tabelle 5-1: Erlaubte Eröffnungs-Modi für Text-Dateien

Ist die Pascal-Dateivariable f vom Typ "FILE OF t", dann gilt:

	Reset(f)	Rewrite(f)	Replace(f)	Extend(f)
*DUMMY SAM-Datei ISAM-Datei	ok	ok	ok	ok
PLAM-Elem. EAM-Datei	ok	ok	Open_Error (1605)	Open_Error (1605)

Tabelle 5-2: Erlaubte Eröffnungs-Modi für Nicht-Text-Dateien

Standard-Dateien

Zu den Standard-Dateien gehören Systemdateien des BS2000 und die Arbeitsbereiche des Editors EDT. Zur Unterscheidung von zulässigen Dateinamen des Datenverwaltungssystems müssen sie mit dem Präfix "*" versehen werden. Eine Standard-Datei kann nur einer Pascal-Text-Datei zugeordnet werden.

*SYSDTA	entspricht der Systemdatei SYSDTA
*SYSOUT	entspricht der Systemdatei SYSOUT
*SYSLST	entspricht der Systemdatei SYSLST
*DUMMY	entspricht der Systemdatei *DUMMY. Beim Schreiben auf *DUMMY gehen die Daten verloren, beim Lesen aus *DUMMY wird sofort EOF geliefert
*EDT(zahl)	bezeichnet einen Arbeitsbereich des Editors EDT. zahl gibt die Nummer des Arbeitsbereichs an und muß im Bereich 0..9 liegen. Die Angabe von "(zahl)" ist optional. Bei fehlender Angabe wird der aktuell gültige Arbeitsbereich verwendet.

Eröffnungsmodi

- Reset
- Rewrite
- Extend

SAM- und ISAM-Dateien

Das Pascal-XT System kann alle im BS2000 unterstützten SAM-Dateien bearbeiten. Sequentielle Dateien können sowohl Textdateien als auch Nicht-Textdateien sein.

Beim Eröffnen einer Pascal-Datei zum Schreiben (Rewrite) wird die zugeordnete BS2000-Datei standardmäßig als eine sequentielle Datei mit variabler Satzlänge angelegt (FCBTYPE=SAM, RECFORM=V). Für alle weiteren Dateiattribute werden die Standardwerte des BS2000 verwendet (siehe [6]). Davon abweichende Dateiattribute müssen mit dem FILE-Kommando festgelegt werden.

Beim Eröffnen einer Pascal-Datei zum Lesen (Reset) und für die Eröffnungsmodi Extend und Replace (siehe A.5) werden die im Katalogeintrag angegebenen Dateiattribute vom Pascal-XT Laufzeitsystem übernommen. Inkompatibilitäten zwischen den Angaben im Katalogeintrag und den vom Programm vorausgesetzten Werten werden vom Pascal-XT Laufzeitsystem gemeldet (siehe Tabellen in 5.3.2).

Die Sprache Pascal-XT kennt selbst keine indexsequentiellen (ISAM-) Dateien, jedoch gibt es ein vordefiniertes Paket DMSIO (siehe A.5), das das Arbeiten mit ISAM-Dateien ermöglicht. Die Eigenschaften einer ISAM-Datei müssen vor der Generierung mit dem FILE-Kommando (siehe 5.3.2) festgelegt werden. Ist einer Pascal-Text-Datei eine ISAM-Datei zugeordnet, dann werden beim Schreiben keine Schlüssel erzeugt und beim Lesen vorhandene Schlüssel ignoriert. Der Schlüssel eines Satzes kann mit der Prozedur "movekey" des Pakets DMSIO gewonnen werden (siehe A.5).

Keybehaftetes und keyloses Dateiformat

Ab BS2000 V10 wird die sog. keylose Plattenperipherie unterstützt. Von dieser Version an unterscheidet man zwischen

- dem bisherigen keybehafteten Format (auch K-Format genannt)
- dem neu eingeführten keylosen Format (auch NK-Format genannt).

Das Dateiformat wird durch den Operanden BLKCTRL des FILE-Kommandos gesteuert. Die im K-Format enthaltenen PAM-Keys werden im NK-Format in den Datenbereich verlagert und verkürzen damit den für die Datensätze zur Verfügung stehenden Bereich. Dies hat eine Veränderung der minimalen und optimalen Blockgröße bei gegebener Record-Größe zur Folge. Der Benutzer muß das berücksichtigen, wenn er die Blockgröße in einem FILE-Kommando angibt. Gibt er sie nicht an, dann bestimmt das Laufzeitsystem bei der Neuerstellung einer Datei die minimale Blockgröße abhängig vom PAM-Key Format.

Eröffnungsmodi

- Reset
- Rewrite
- Extend
- Replace

PLAM-Bibliothekselemente

Das Pascal-XT System erlaubt die Bearbeitung von Programmbibliotheken, die mit der Bibliotheks-Zugriffsmethode PLAM bearbeitet werden. In diesen Bibliotheken, kurz PLAM-Bibliotheken genannt, können Elemente verschiedenen Typs abgespeichert werden. Für jedes Element eines Typs können mehrere Versionen existieren. Die detaillierte Beschreibung von Programmbibliotheken ist dem LMS-Handbuch [3] zu entnehmen.

PLAM-Bibliothekselemente können an Pascal-Dateien nur über die Standardprozedur Assignfile (siehe 5.3.3) zugewiesen werden.

Eröffnungsmodi

- Reset
- Rewrite

Elementbezeichnung

Die Angabe eines Bibliothekselements erfolgt analog wie in den Anweisungen des Programmiersystems. Die Syntax lautet:

plam-element = (bibliotheks-name , element)
element = element-name [([version][,typ])]

bibliotheks-name
Muß ein gültiger Dateiname des Datenverwaltungssystems sein.

element-name Bezeichnet den Namen des Elements in der Bibliothek. Der Name kann bis zu 64 Zeichen lang sein.

version Bezeichnet die Elementversion. Sie kann bis zu 24 Zeichen lang sein. Eine Sonderstellung besitzen die Versionsbezeichnungen *INCREMENT, *UPPER-LIMIT und *HIGHEST-EXISTING.

*INCREMENT bewirkt beim Schreiben eine automatische Fortschaltung der Versionsbezeichnung des Bibliothekselements. Diese Angabe ist ab PLAM-Version V2.0 möglich.

*UPPER-LIMIT steht beim Lesen und Schreiben für die höchstmögliche Version. Diese Angabe entspricht der bisher bei LMS als @-Version bezeichneten Version.

*HIGHEST-EXISTING steht für die höchste vorhandene Version. Beim Lesen wird diese Version gelesen. Wenn das Element vorhanden ist, wird beim Schreiben die höchste Version überschrieben, ansonsten wird eine Standardversion angelegt.

Weiterhin ist in Pascal-XT V2.2A die Versionsangabe *STD möglich. Sie steht beim Lesen für die (lexikographisch gesehen) höchste vorhandene Version, entspricht also der Angabe *HIGHEST-EXISTING. Beim Schreiben steht sie für die höchstmögliche Version, entspricht also der Angabe *UPPER-LIMIT.

Standardwert: *HIGHEST-EXISTING

typ — Bezeichnet den Elementtyp. Er kann bis zu 8 Zeichen lang sein. Elemente vom Typ C (Binärdateien) werden nicht unterstützt.

Standardwert: S (für Source)

Weitergehende Konventionen für die Elementbezeichnung werden von PLAM nicht festgelegt. Es empfiehlt sich jedoch, die in LMS [3] getroffenen Konventionen einzuhalten.

Temporäre Dateien

Temporäre Dateien werden durch EAM-Dateien realisiert. Nach Beendigung des Programms werden alle EAM-Dateien gelöscht.

Eröffnungsmodi

- Reset
- Rewrite

Zuordnung von BS2000-Dateien zu Pascal-Dateien

Während des Ablaufs eines Programms müssen den im Programm deklarierten Pascal-Dateien (FILE-Variablen) Dateien des Betriebssystems zugeordnet werden. Die Kapitel 5.3.1 bis 5.3.4 beschreiben die verschiedenen Möglichkeiten solcher Zuordnungen.

Standardmäßige Zuordnungen

Input und Output

Den vordefinierten Textdateien Input und Output sind standardmäßig die Systemdateien SYSDTA bzw. SYSOUT zugeordnet. Mit der vordefinierten Prozedur Assignfile (siehe 5.3.3) kann ihnen aber auch jede der in Abschnitt 5.2 beschriebenen Dateien zugeordnet werden (aber: SYSOUT kann nicht Input und SYSDTA nicht Output zugeordnet werden).

Externe Pascal-Dateien

Einer externen Pascal-Datei muß beim Ablauf des Programms vor dem Eröffnen der Datei (Reset, Rewrite, Extend, Replace) eine BS2000-Datei zugeordnet werden. Standardmäßig verwendet das Laufzeitsystem den Pascalnamen der externen Pascal-Datei als Linknamen und holt sich aus der Task File Table den für den Linknamen angegebenen Dateinamen. Es wird ein OPEN_ERROR gemeldet, wenn in der Tabelle kein Eintrag vorhanden ist.

Lokale Pascal-Dateien

Lokale Pascal-Dateien werden standardmäßig als temporäre EAM-Dateien eingerichtet.

Zuordnung mit dem FILE-Kommando

Mit dem FILE-Kommando des BS2000 können Eigenschaften einer Datei definiert und ein Eintrag im Dateikatalog und der Task File Table (TFT) erstellt werden. Über die TFT kann das Pascal-XT Laufzeitsystem eine Zuordnung zu einer externen Pascal-Datei herstellen. Dazu muß im FILE-Kommando ein Linkname angegeben werden, der mit dem aus dem Variablennamen der externen Datei gebildeten Linknamen (siehe unten) übereinstimmen muß.

Konvention für die Bildung von Linknamen

Die Bildung des Linknamens aus dem Pascalnamen einer externen Datei erfolgt nach folgenden Regeln:

- Ist der Pascalname länger als 8 Zeichen, dann wird er auf 8 Zeichen gekürzt.
- Unterstriche ('_') im Pascalnamen werden im Linknamen durch Nummernzeichen ('#') ersetzt.

Eindeutigkeit von Programmparametern

Linknamen können maximal 8 Zeichen lang sein. Aus diesem Grund müssen die Variablennamen aller zu einem Programm gehörenden externen Dateien in den ersten 8 Zeichen verschieden sein. Eine Verletzung dieser Bedingung kann vom Compiler nicht überprüft werden und kann beim Ablauf des Programms zu einem undefinierten Verhalten führen.

Welche Zuordnungen sind möglich?

Das Pascal-XT Laufzeitsystem stellt die Zuordnung zu einer BS2000-Datei standardmäßig über den Link-Mechanismus her. Damit können nur SAM- und ISAM-Dateien an externe Pascal-Dateien (siehe 5.1.1) zugeordnet werden. Die Zuordnung erfolgt über den Linknamen, der aus dem Bezeichner der Pascal-Datei abgeleitet wird. Die Zuordnung von SYSDTA bzw. SYSOUT an die vordefinierten Textdateien Input bzw. Output kann mit dem FILE-Kommando nicht verändert werden (siehe aber Assignfile, 5.3.3).

Wann muß das FILE-Kommando ausgeführt werden?

Das FILE-Kommando muß vor dem Eröffnen der Pascal-Datei (Reset, Rewrite, Extend, Replace) ausgeführt werden. Der Aufruf kann vor der Ausführung eines Programms abgesetzt oder im Programm über die Prozedur CMD des vordefinierten Pakets BS2000CALLS (siehe A.4) aufgerufen werden.

Beim Eröffnen einer Pascal-Datei zum Schreiben wird die zugeordnete BS2000-Datei standardmäßig als SAM-Datei mit den vom BS2000 vorgegebenen Dateiattributen eingerichtet.

Eine Änderung der Dateiattribute kann nur mit dem FILE-Kommando erreicht werden. Insbesondere bei der Verarbeitung von ISAM-Dateien müssen die Attribute über das FILE-Kommando eingestellt werden (siehe vordefiniertes Paket DMSIO in A.5). In den Tabellen 5-3 und 5-4 sind die Auswirkungen beim Eröffnen von Dateien beschrieben, abhängig davon, ob ein FILE-Kommando angegeben wurde.

Zusammenwirken mit der Prozedur Assignfile

Dateizuordnungen können über die vordefinierte Prozedur Assignfile und das FILE-Kommando gemeinsam hergestellt werden. Das ist insbesonders dann notwendig, wenn Dateiattribute verändert werden müssen, was mit der Prozedur Assignfile nicht möglich ist.

Verträglichkeit von Katalogeintrag und FILE-Kommando

Beim Eröffnen einer existierenden Datei mit Reset oder Replace verlangt das Laufzeitsystem die Verträglichkeit der Attribute des Katalogeintrages mit denen des FILE-Kommandos.

- Die Überprüfung wird für die Attribute RECFORM, FCBTYPE, KEYPOS und KEYLEN durchgeführt.
- Für das Attribut RECSIZE wird zusätzlich noch die Größe des Laufzeitsystem-internen Puffers mit betrachtet. Dieser ist für Nicht-Text-Dateien gleich der Größe des Elementtyps der FILE-Variablen und für Text-Dateien der Wert von MAXLINELENGTH (der Standardwert ist 254, siehe auch Kap. 5.3.3). Diese Werte erhöhen sich bei RECFORM=V um 4 und bei SPACE=E um 1.
- Wird kein FILE-Kommando bzw. der RECSIZE-Operand nicht angegeben, dann vergleicht das Laufzeitsystem die interne Puffergröße mit der RECSIZE-Angabe aus dem Katalog und löst bei unverträglichen Kombinationen die Exception Open_Error aus.
- Wird der RECSIZE-Operand im FILE-Kommando angegeben, dann vergleicht das Laufzeitsystem die interne Puffergröße mit dieser Angabe und löst bei unverträglichen Kombinationen die Exception Open_Error aus.
- Den Vergleich der RECSIZE aus dem Katalog mit der aus dem FILE-Kommando überläßt das Laufzeitsystem dem Betriebssystem.
- Das Betriebssystem (DVS) betrachtet RECSIZE=0 mit anderen RECSIZE-Werten als verträglich und bearbeitet die Datei korrekt.

Die verträglichen Kombinationen von RECSIZE-Angaben sind in DVS Manual [14] beschrieben.

Eröffnen mit	Aktuelle Datei existiert	Aktuelle Datei existiert nicht
Reset	Die Datei wird mit ihren aktuellen Attributen eröffnet. Sind die Dateiattribute nicht mit den Attributen der Pascal-Datei verträglich, dann tritt beim Lesen ein Laufzeitfehler auf (READ_ERROR).	OPEN_ERROR (System_Code 1604)
Rewrite Extend	Die Datei wird als SAM-Datei mit variabler Satzlänge eingerichtet.	
Replace	Datei existiert: Verhalten wie bei Reset Datei existiert nicht: Verhalten wie bei Rewrite	

Tabelle 5-3 Eröffnen einer Datei ohne Angabe eines FILE-Kommandos

Eröffnen mit	Aktuelle Datei existiert	Aktuelle Datei existiert nicht
Reset	Es wird ein OPEN_ERROR gemeldet, wenn die Dateiattribute im Katalogeintrag nicht mit denen im FILE-Kommando verträglich sind. Sind im FILE-Kommando die Dateiattribute nicht angegeben, dann tritt bei unverträglichen Attributen erst beim Lesen ein Laufzeitfehler auf.	OPEN_ERROR (System_Code 1604)
Rewrite Extend	Die Datei wird mit den im FILE-Kommando angegebenen Attributen eingerichtet.	
Replace	Datei existiert: Verhalten wie bei Reset Datei existiert nicht: Verhalten wie bei Rewrite	

Tabelle 5-4 Eröffnen einer Datei mit Angabe eines FILE-Kommandos

Zuordnung mit der vordefinierten Prozedur Assignfile

Mit der vordefinierten Prozedur Assignfile kann innerhalb eines Programms einer externen oder lokalen Pascal-Datei jede der in Abschnitt 5.2 beschriebenen Dateien zugeordnet werden. Für ISAM-Dateien sind allerdings einige Einschränkungen zu beachten (siehe unten). Die Prozedur besitzt zwei Parameter:

Assignfile (f, beschreibung)

"f" ist eine Variable eines beliebigen Pascal-Filetyps. "beschreibung" ist ein String (Zeichenkettenausdruck), mit dem die BS2000-Datei beschrieben wird, die der Pascal-Datei zugeordnet werden soll. Die Beschreibung kann eine Dateibeschreibung, eine Attributbeschreibung oder beides zusammen enthalten (siehe Definitionen weiter unten). Der String darf Klein- und Großbuchstaben enthalten, Leerzeichen (Blanks) werden ignoriert.

Der Aufruf von Assignfile bewirkt lediglich die Überprüfung des Parameters "beschreibung" auf syntaktische Korrektheit und die Abspeicherung der Angaben im Laufzeitsystem. Die semantischen Eigenschaften (Gültigkeit der Attribute etc.) werden erst beim Eröffnen der Datei überprüft (siehe auch 5.3.2).

Außer für Input und Output, die implizit eröffnet werden, wird die Zuordnung einer BS2000-Datei zu einer Pascal-Datei erst beim Eröffnen der Datei (Reset, Rewrite, Extend, Replace) durchgeführt. Existiert noch eine früher getroffene Zuordnung, dann wird die bisher zugeordnete BS2000-Datei geschlossen. Assignfile auf die vordefinierte Textdatei Input bzw. Output bewirkt das Schließen und Wiedereröffnen der Textdatei, d.h. die im Parameter "beschreibung" definierten Zuordnungen werden sofort wirksam.

Die Zeichenkette innerhalb des Parameters "beschreibung" muß der folgenden Syntax für datei genügen. Leerzeichen zwischen den lexikalischen Einheiten werden ignoriert.

```
datei                =  dateibeschreibung              |
                        [","] attributbeschreibung     |
                        dateibeschreibung "," attributbeschreibung.

dateibeschreibung    =  standard-datei |
                        datei-name     |
                        plam-element   |
                        temp-datei.

attributbeschreibung = attribut { "," attribut }.

attribut              = "SPACE" "=" "E"           |
                        "LINK" "=" link-name      |
                        "MAXLINELENGTH" "="  Ganzzahl.
```

Hinweis

In den folgenden Beispielen kann die Dateivariable "f" für eine beliebige externe oder lokale Pascal-Datei stehen. Für den Parameter "beschreibung" werden nur Zeichenkettenliterale angegeben. Selbstverständlich können beliebige Zeichenkettenausdrücke auftreten.

- **Dateibeschreibung**

In der Dateibeschreibung wird der Name der gewünschten BS2000-Datei angegeben. Früher getroffene Zuordnungen und Attributbeschreibungen werden gelöscht.

standard-datei

Die zulässigen Standarddateien sind im Abschnitt 5.2.1 beschrieben. Die Pascal-Datei muß vom Typ Text sein.

Beispiele

```
Assignfile (f, '*DUMMY');
Assignfile (f, '*SYSOUT');
Assignfile (f, '*EDT');
Assignfile (f, '*EDT(5)');
```

datei-name

"datei-name" steht für den Namen einer BS2000-Datei. Beim Eröffnen der Pascal-Datei zum Schreiben (Rewrite) wird unter diesem Namen eine SAM-Datei mit den vom BS2000 vorgegebenen Standardattributen eingerichtet. Soll die zu erzeugende Datei eine ISAM-Datei sein, dann müssen zuvor mit dem FILE-Kommando die Dateiparameter definiert werden (siehe 5.2.2 und 5.3.2). Beim Eröffnen der Pascal-Datei zum Lesen kann "datei-name" für eine SAM- oder ISAM-Datei stehen. Die Behandlung von Schlüssseln in ISAM-Textdateien ist in 5.2.2 beschrieben.

Beispiel

```
Assignfile (f, '$USERID.BEISPIEL.PROG')
```

Der Pascal-Datei f wird die Datei $USERID.BEISPIEL.PROG zugeordnet.

plam-element

Ein PLAM-Element wird durch Bibliotheksname, Elementname, Typ und Version definiert. Die genaue Beschreibung ist in Abschnitt 5.2.3 angegeben. Bei fehlender Angabe von Typ und Version verwendet das Laufzeitsystem standardmäßig "S" (für Source) und beim Lesen die höchste existierende, beim Schreiben die höchstmögliche Version. Die Bibliothek kann auch über einen Linknamen angesprochen werden (siehe LINK-Attribut).

Beispiele

```
Assignfile (f, '(TOOLS, LMS.PROG (10A,S))')
```

Der Pascal-Datei f wird das Element LMS.PROG mit der Version 10A und dem Typ S aus der PLAM-Bibliothek TOOLS zugeordnet.

```
Assignfile (f, '(TOOLS, LMS.PROG), LINK=F')
```

Der Pascal-Datei f wird das Bibliothekselement LMS.PROG mit der höchsten Version (*STD) und dem Typ S zugeordnet. Die Bibliothek wird über den Linknamen F in der Task-File-Table gesucht. Wurde der Linkname F nicht gefunden, dann wird die Bibliothek TOOLS verwendet.

```
Assignfile (f, '(,LMS.PROG(,P)), LINK=LIB')
```

Der Pascal-Datei f wird das Bibliothekselement LMS.PROG mit der höchsten Version (*STD) und dem Typ P zugeordnet. Die Bibliothek wird über den Linknamen LIB angesprochen. Vor dem Eröffnen des Bibliothekselementes muß ein FILE-Kommando mit dem Dateinamen der PLAM-Bibliothek und dem Linknamen LIB abgesetzt werden, sonst tritt ein Open_Error mit dem Systemfehlercode 1603 auf.

temp-datei

Jeder Pascal-Datei f kann eine temporäre Datei (EAM-Datei) zugeordnet werden, indem ein Leerstring angegeben wird. Auf temporäre Dateien kann nach Beendigung des Programms nicht mehr zugegriffen werden.

Beispiel

```
Assignfile (f, '')
```

- **Attributbeschreibung**

In der Attributbeschreibung können zusätzliche Eigenschaften einer Datei spezifiziert werden, wie z.B. Ändern der Puffergrößen für Textdateien oder Ansprechen der Datei über Linknamen. Die Attributbeschreibungen können zusammen mit einem Dateinamen im selben Aufruf von Assignfile oder allein in einem folgenden Aufruf angegeben werden. Werden die Attribute in einem eigenen Aufruf von Assignfile angegeben, dann hat das optionale Komma vor der Attributliste eine wesentliche Funktion.

(a) Wird das Komma nicht angegeben, dann werden die Attribute der vorhandenen Beschreibung hinzugefügt, sofern eine vorhanden ist.

(b) Beginnt die Attributliste mit einem Komma, dann wird die bisherige Beschreibung gelöscht, sofern eine existiert hat.

Lokale Dateien haben implizit eine Beschreibung, in der die Zuordnung zu den temporären EAM-Dateien angegeben wird. Soll einer lokalen Datei eine Nicht-EAM-Datei zugeordnet werden, dann muß dies entweder durch Angabe eines Dateinamens oder eines Link-Attributes mit einem vorausgehenden Komma geschehen.

SPACE = E

Dieses Attribut bewirkt in einer Textdatei die Reservierung der ersten Spalte für Seitenvorschubsteuerzeichen. Der Anwender kann auf dieses Zeichen nicht explizit zugreifen. Damit die vordefinierte Prozedur Page die geforderte Wirkung erzielt, muß bei der Zuordnung einer aktuellen Datei dieses Attribut angegeben werden. Bei der Ein-/Ausgabe ergeben sich folgende Auswirkungen:

Ausgabe: Die Spalte 1 jeder Zeile wird für Steuerzeichen reserviert und mit einem Leerzeichen (Blank) besetzt, d.h. die von einem Programm ausgegebene Zeile wird um ein Zeichen verlängert.
Bei Aufruf der vordefinierten Prozedur Page wird die aktuelle, evtl. noch nicht mit Writeln abgeschlossene Zeile ausgegeben, und in der Spalte 1 der folgenden Zeile ein Seitenvorschub-Steuerzeichen ("A") ausgegeben.
Der Aufruf von Page hat keine Wirkung, wenn das Attribut SPACE nicht angegeben wurde.
Für die Systemdateien SYSLST und SYSOUT ist standardmäßig SPACE=E eingestellt.

Eingabe: Die Spalte 1 jeder Zeile wird überlesen und nicht an das verarbeitende Programm weitergeleitet.

Beispiel

```
Assignfile (listing, 'LST.MAIN, SPACE=E')
```

Der Pascal-Datei listing wird die Datei LST.MAIN zugordnet. Die erste Spalte in der Datei ist für das Vorschubsteuerzeichen reserviert.

LINK = link-name

Mit diesem Attribut wird die Zuordnung einer BS2000-Datei an eine Pascal-Datei über den Link-Mechanismus des BS2000 hergestellt. Vor dem Eröffnen (Reset, Rewrite,...) der Pascal-Datei muß ein FILE-Kommando mit Angabe des gewünschten Dateinamens und dem in Assignfile angegebenen Linknamen gegeben werden. Das Laufzeitsystem beschafft sich dann beim Eröffnen der Pascal-Datei über die Task File Table den Dateinamen und übernimmt weitere Parameter des FILE-Kommandos, sofern welche angegeben wurden. Wird der Pascal-Datei in Assignfile eine Datei zugeordnet und zusätzlich das LINK-Attribut angegeben (in verschiedenen Aufrufen oder demselben Aufruf von Assignfile), dann dient der in Assignfile angegebene Dateiname als Standardwert. Er wird nur dann verwendet, wenn das Laufzeitsystem in der Task File Table keinen Dateinamen zu dem verwendeten Linknamen findet.
Beim Eröffnen der Pascal-Datei wird ein Open_Error mit dem Systemfehlercode 1603 gemeldet, wenn weder ein Dateiname in Assignfile noch ein FILE-Kommando für den Linknamen (mit Angabe eines Dateinamens) angegeben wurde.
Als "link-name" (maximal 8 Zeichen) kann ein beliebiger Name oder der aus dem Bezeichner der Pascal-Datei abgeleitete Name (siehe 5.3.2) verwendet werden. Linknamen können für lokale und externe Pascal-Dateien angegeben werden.

Linknamen für externe Pascal-Dateien

Über den Link-Mechanismus können Dateiattribute der zugeordneten BS2000-Datei durch die entsprechenden Parameter im FILE-Kommando vorgegeben werden. Bei existierenden Dateien müssen diese Parameter mit den Einträgen im Katalog übereinstimmen (siehe Kapitel 5 und A.5).

Linknamen für lokale Pascal-Dateien

Für lokale Pascal-Dateien (d.h. nicht in Programm-Parameterlisten aufgeführte Datei-Variable) *muß* vor der Attributliste das führende Komma angegeben werden.

Beispiele

```
Assignfile (f, 'TOOLS.SPEC, LINK=F')
```

Der Pascal-Datei f wird über den Linknamen F eine Datei zugeordnet. Wurde kein FILE-Kommando für diesen Linknamen abgesetzt, dann wird die Datei TOOLS.SPEC verwendet.

```
Assignfile (f, 'LINK=F');
BS2000CALLS.cmd ('/FILE TEST, LINK=F, KEYPOS=3, KEYLEN=10', error);
Reset (f);
...
Assignfile (f, 'LST.BSP');
```

Der Pascal-Datei f wird über den Linknamen F die ISAM-Datei TEST mit den angegebenen Attributen zugeordnet. Beim zweiten Aufruf von Assignfile wird an die Pascal-Datei f eine Datei zugeordnet und damit die bisherige Zuordnung mit den Attributen gelöscht.

MAXLINELENGTH = Ganzzahl

Bei Textdateien ist der Laufzeitsystem-interne Zeilenpuffer standardmäßig 254 Zeichen lang. Längere Zeilen beim Lesen von einer Datei bzw. Schreiben in eine Datei führen zu einem FILE_ERROR. Mit dem Attribut MAXLINELENGTH kann nun die Größe des Zeilenpuffers verändert werden (z.B. um mit einem einzigen Aufruf von Writeln einen Bildschirminhalt auszugeben). "Ganzzahl" muß eine positive Integerzahl sein und gibt die gewünschte Anzahl von Zeichen pro Zeile an. Bei Werten kleiner oder gleich 0 wird ein FILE_ERROR gemeldet. Bei der Längenangabe muß ein durch SPACE=E bedingtes zusätzliches Zeichen nicht berücksichtigt werden. Wird das Attribut bei einem Aufruf von Assignfile nicht angegeben, dann gilt die Standardlänge, unabhängig von einem zuvor eingestellten Wert.

Das Attribut wird erst beim Eröffnen der Pascal-Datei wirksam. Eine Ausnahme bilden die vordefinierten Textdateien Input und Output, die der Anwender nicht explizit eröffnen kann. Sie werden durch den Aufruf von Assignfile implizit geschlossen und umgehend wieder mit den neuen Attributen eröffnet.

Beispiele

```
Assignfile (output, '*SYSOUT, MAXLINELENGTH=4000')
```

Nach dem Aufruf von Assignfile können auf die vordefinierte Textdatei Output Zeilen mit einer maximalen Länge von 4000 Zeichen ausgegeben werden. Die Zuordnung von Output zum Terminal oder einer evtl. zuvor eingestellten Datei wird nicht verändert.

```
Assignfile (f, 'MAXLINELENGTH=512, SPACE=E')
```

Nach dem Eröffnen (Reset, Rewrite) der Datei f können Zeilen mit einer maximalen Länge von 512 Zeichen ausgegeben werden. Das zusätzlich benötigte Zeichen für die Vorschubsteuerung (SPACE=E) muß nicht berücksichtigt werden.

Beispiel

Mit dem Programm LIST kann der Inhalt einer Datei auf Terminal ausgegeben werden. Im Aufruf von Assignfile wird sowohl der im Dialog angeforderte Dateiname als auch ein Linkname angegeben. Der im Dialog eingegebene Name wird nur verwendet, wenn kein entsprechendes FILE-Kommando abgesetzt wurde.

```
/EXEC $PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ... LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//sy er *
//c (plam.manual,list.prog),*dummy
   >>> COMPILATION SUCCESSFUL (WARNINGS:  0;   NOTES:  0)
//run
Name der Datei eingeben
(plam.manual, list.prog)  ------------------------------------------------------ (01)
(*$TITLE = 'Ausgabe einer Datei nach SYSOUT'*)

program LIST (input, output, source);

var
      source    : text;
      str       : string;
      dateiname : string;

begin
   writeln (output, 'Name der Datei eingeben');
   readln  (input);
   read    (input, dateiname);
   assignfile (source, concat (dateiname, ',LINK=SOURCE'));
   reset   (source);
   while not eof (source) do begin
      readln  (source, str);
      writeln (output, str);
      end;
end (* LIST *).
//sy file testdatei,link=source  ------------------------------------------------ (02)
//run
Name der Datei eingeben
(plam.manual, list.prog)  ------------------------------------------------------ (03)
   ***   Diese Datei "testdatei" enthaelt nur diese Zeile   ***
//
```

(01) Das Programm LIST fordert den Namen der auszugebenden Datei an. Es wird der Name des Bibliothekselements eingegeben, in dem das Quellprogramm von LIST enthalten ist. Der Link-Mechanismus hat keine Wirkung, da noch kein FILE-Kommando angegeben wurde.

(02) Für den Linknamen SOURCE wird ein FILE-Kommando mit Angabe des Dateinamens "testdatei" abgesetzt.

(03) Die Eingabe des Dateinamens hat keine Wirkung mehr. Es wird der im FILE-Kommando angegebene Dateiname verwendet.

Zuordnung in der RUN-Anweisung

Beim Starten eines Programms in der Programmierumgebung mit der RUN-Anweisung (siehe 2.6.11) können im Operanden PARAMETER Dateizuordnungen in einem String folgendermaßen eingegeben werden:

```
PARAMETER = 'pascal-datei = BS2000-dateiname'
```

Mehrere Zuordnungen müssen durch Kommata getrennt werden. Als "pascal-datei" ist nur der Bezeichner einer externen Pascal-Datei zulässig. Für "BS2000-dateiname" können nur SAM- und ISAM-Dateien angegeben werden. Besonderheiten im Zusammenhang mit Dateiattributen sind in 5.3.2 beschrieben. Die Bindung der BS2000-Datei an die Pascal-Datei erfolgt über das FILE-Kommando mit dem Namen der Pascal-Datei als 1Linknamen, wie es in Abschnitt 5.3.2 beschrieben ist. Nach der Ausführung des Programms wird der Linkname wieder freigegeben. Die Programmierumgebung merkt sich allerdings die Zuordnung und stellt sie bei einem erneuten Aufruf der RUN-Anweisung wieder ein. Den vordefinierten Textdateien Input und Output können keine BS2000-Dateien zugewiesen werden. Sie sind fest den Systemdateien SYSDTA bzw. SYSOUT zugeordnet.

Beispiel 1

Einem Programm mit der externen Datei SOURCE soll die BS2000-Datei "testdatei" zugeordnet werden. Die beiden Anweisungen zeigen die verschiedenen Eingabemöglichkeiten für die RUN-Anweisung.

```
//run parameter = 'source=testdatei'
//r ,'source=testdatei'
```

Beispiel 2

Einem Programm mit den externen Dateien SOURCE und DESTINATION sollen bei der Ausführung die BS2000-Dateien "datei1" und "datei2" zugeordnet werden.

```
//r ,'source = datei1, destination = datei2'
```

Verwendete Dateioperationen

Ist die in den vordefinierten Unterprogrammen Rewrite, Reset, Get, Put, Readln und Writeln als erster Parameter angegebene Pascal-Datei einer BS2000-Datei (SAM oder ISAM), einer Bibliothek oder einer Systemdatei zugeordnet, so haben die Unterprogramme eine durch das Dateiverwaltungssystem DVS bzw. das Bibliothekszugriffssystem PLAM bestimmte Wirkung. Im einzelnen gilt:

Rewrite: Die BS2000-Datei wird gegebenenfalls zuerst mit der DVS-CLOSE-Operation geschlossen und neu mit der DVS-OPEN-Operation im Open-Modus OUTPUT eröffnet.
Bei PLAM-Bibliothekselementen wird zuerst die Bibliothek mit der PLAM-PMATCH-Operation im Attach-Modus INOUT und dann das Element mit der PLAM-PMOPEN Operation im Open-Modus WRITE eröffnet. Kann die Datei bzw. die PLAM-Bibliothek oder das Bibliothekselement nicht eröffnet werden, so tritt ein Open_Error ein.
Systemdateien werden nicht geöffnet, da sie als permanent offen gelten.
Für eine Textdatei wird ein interner Zeilenpuffer vom Typ String [n] angelegt, wobei n die im letzten Aufruf von Assignfile eingestellte maximale Zeilenlänge bzw. die Standardlänge 254 ist.

Reset: Analog zu Rewrite wird die BS2000-Datei zuerst geschlossen. Bei mit Rewrite oder Replace eröffneten Textdateien wird vor dem Schließen noch der interne Zeilenpuffer in die Datei übertragen, falls dieser nicht leer ist (implizites Writeln). Die Übertragung des Zeilenpuffers bei Close ist von der nachfolgenden Open-Operation Rewrite/Reset unabhängig. Anschließend wird die BS2000-Datei mit der DVS-OPEN-Operation im Open-Modus IN eröffnet.
Bei PLAM-Bibliothekselementen wird zuerst die Bibliothek mit der PLAM-PMATCH-Operation im Attach-Modus INPUT und dann das Element mit der PLAM-PMOPEN-Operation im OPEN-Modus INPUT eröffnet. Kann die Datei oder die Bibliothek nicht eröffnet werden, so tritt ein Open_Error auf.
Systemdateien werden nicht geöffnet, da sie als permanent offen gelten.
Bei Nicht-Textdateien wird der erste Satz der BS2000-Datei mit der DVS-GET-Operation bzw. der PLAM-PMGETA-Operation gelesen und in die Puffervariable der Pascal-Datei übertragen.
Für eine Textdatei wird ein Zeilenpuffer vom Typ String[n] angelegt, wobei n die im letzten Aufruf von Assignfile eingestellte maximale Zeilenlänge bzw. die Standardlänge 254 ist.
Der erste Satz der BS2000-Datei wird mit der DVS-GET-Operation bzw. der PLAM-PMGETA-Operation bzw. der DVS-RDATA-Operation gelesen und in diesen Zeilenpuffer übertragen, das erste Zeichen des Zeilenpuffers in die Puffervariable der Textdatei kopiert und Eoln auf False gesetzt. Ein Vorauslesen des ersten Satzes mittels RDATA ist nur für SYSDTA gültig, da bei Stan-

dardinput kein Vorauslesen des ersten Satzes erfolgen darf. Ist der erste Satz leer, so wird stattdessen ein Leerzeichen in die Puffervariable eingetragen und Eoln auf True gesetzt. Konnte bei SAM- und ISAM-Dateien mit der DVS-GET-Operation kein Satz gelesen werden, da die Datei leer ist, so wird stattdessen Eof auf True gesetzt. Analoges gilt, wenn bei PLAM-Bibliothekselementen mit der PLAM-PMGETA-Operation kein Satz der Satzart 1 mehr gelesen werden konnte.

Bei der Systemdatei SYSDTA wird die Kommandoeingabe /EOF als Dateiende interpretiert.

Get: Bei Nicht-Textdateien wird der nächste Satz der BS2000-Datei mit der DVS-GET-Operation bzw. der PLAM-PMGETA-Operation gelesen und in die Puffervariable der Pascal-Datei übertragen. Bei mit Replace eröffneten ISAM-Dateien oder bei ISAM-Dateien mit SHARED-UPDATE=YES wird der gelesene Satz gesperrt (LOCK). Die Aussage über LOCK gilt auch für Textdateien. Ist bei Textdateien unmittelbar vor Aufruf von Get Eoln True, so wird der nächste Satz der BS2000-Datei mit der DVS-GET-Operation bzw. der PLAM-PMGETA-Operation bzw. der DVS-RDATA-Operation gelesen und in den internen Zeilenpuffer übertragen, das erste Zeichen des Zeilenpuffers in die Puffervariable der Textdatei eingetragen und Eoln auf False gesetzt. Ist der gelesene Satz leer, so wird stattdessen ein Leerzeichen in die Puffervariable eingetragen und Eoln auf True gesetzt. Sonst wird das nächste Zeichen des Zeilenpuffers in die Puffervariable kopiert. Falls es keine weiteren Zeichen im Zeilenpuffer mehr gibt, so wird stattdessen ein Leerzeichen in die Puffervariable eingetragen und Eoln auf True gesetzt. Konnte bei SAM- und ISAM-Dateien mit der DVS-GET-Operation kein Satz mehr gelesen werden, da das Dateiende bereits erreicht wurde, so wird stattdessen Eof auf True gesetzt. Analoges gilt, wenn bei PLAM-Bibliothekselementen mit der PLAM-PMGETA-Operation kein Satz der Satzart 1 mehr gelesen werden konnte.
Bei der Systemdatei SYSDTA wird die Kommandoeingabe /EOF als Dateiende interpretiert.

Put: Bei Nicht-Textdateien wird ein nächster Satz der BS2000-Datei mit dem Inhalt der Puffervariablen der Pascaldatei mittels einer DVS-Operation geschrieben. Hierbei wird, für mit Rewrite eröffnete ISAM-Dateien die DVS-PUT-Operation, für mit Replace eröffnete ISAM-Dateien die DVS-STORE-Operation verwendet. Bei SAM-Dateien wird die Puffervariable, da hier im Locate Mode gearbeitet wird, nur in den internen Systempuffer der BS2000-Datei gestellt. Paßt die Puffervariable nicht mehr in den Systempuffer, wird der Systempuffer mit der DVS-RELESE-Operation entleert.
Bei Bibliotheken mit Nicht-Textelementen wird die Puffervariable der Pascaldatei mit der PLAM-PMPUTA-Operation in die Bibliothek übertragen.
Bei Textdateien wird der Inhalt der Puffervariablen der Textdatei als näch-

stes Zeichen in den internen Zeilenpuffer geschrieben. Bei Überlauf des internen Zeilenpuffers wird ein Fehler gemeldet.

Readln: Nur für Textdateien zulässig. Der nächste Satz der BS2000-Datei wird mit der DVS-GET-Operation bzw. der PLAM-PMGETA-Operation gelesen und in den internen Zeilenpuffer übertragen. Konnte bei SAM- und ISAM-Dateien mit der DVS-GET-Operation kein Satz mehr gelesen werden, da das Dateiende bereits erreicht wurde, so wird stattdessen Eof auf True gesetzt. Analoges gilt, wenn bei PLAM-Bibliothekselementen vom Typ S mit der PLAM-PMGETA-Operation kein Satz der Satzart 1 mehr gelesen werden konnte. Sonst wird das erste Zeichen dieses Satzes in die Puffervariable der Textdatei kopiert. Wurde ein leerer Satz gelesen, so wird stattdessen Eoln auf True gesetzt und ein Leerzeichen in die Puffervariable eingetragen.

Writeln: Nur für Textdateien zulässig. Ein nächster Satz der BS2000-Datei wird mit dem Inhalt des internen Zeilenpuffers mittels einer DVS-Operation (analog Put bei Nicht-Textdateien) bzw. der PLAM-PMPUTA-Operation geschrieben. Bei SYSLST wird der Zeilenpuffer mittel DVS-WRLST-Operation und bei SYSOUT wird der Zeilenpuffer mittels DVS-WROUT-Operation in die entsprechende Systemdatei übertragen. Es ist zu beachten, daß das Schreiben einer Leerzeile auf SYSOUT DVS-bedingt keinen Zeilenumbruch verursacht.

Close: Nur wirksam, wenn die BS2000-Datei überhaupt offen ist. Die BS2000-Datei wird mit der DVS-CLOSE-Operation geschlossen. Bei PLAM-Bibliothekselementen wird zuerst das Element mit der PLAM-PMCLOS-Operation und dann die Bibliothek mit der PLAM-PMDTCH-Operation geschlossen. Bei mit Rewrite oder Replace eröffneten Textdateien wird vor dem Schließen noch der interne Zeilenpuffer in die Datei übertragen, falls dieser nicht leer ist (implizites Writeln). Close wird beim Verlassen des Blocks aufgerufen, in dem eine Pascaldatei deklariert ist.

Assignfile:

Die eigentlichen Aktionen von Assignfile werden erst bei einem nachfolgenden Reset, Rewrite oder Replace ausgeführt. Nur beim Wechsel der Dateizuordnung von temporärer Datei auf BS2000-Datei, Bibliothek oder Systemdatei wird die temporäre Datei sofort gelöscht. Für Standardinput wird zusätzlich ein implizites Reset, für Standardoutput ein implizites Rewrite durchgeführt.

Binden und Ausführen von Objektprogrammen

Allgemeines

Um ein ablauffähiges Programm zu erhalten, sind folgende Schritte erforderlich:

- Übersetzen der Quellen des Programms (Hauptprogramm, Spezifikationen und Implementierungen der importierten Pakete) in der richtigen Reihenfolge (siehe 4.1).
- Binden der vom Compiler erzeugten Objektmodule und der Module des Laufzeitsystems zu einem Programm.
- Laden und Starten des Programms.

Für das Hauptprogramm und die Paketimplementierungen erzeugt der Pascal-XT-Compiler nach der korrekten Übersetzung jeweils folgende Objektmodule:

- Code-Modul
- Daten-Modul
- Testtabellen-Modul

und für das Hauptprogramm zuätzlich ein

- Starter-Modul.

Für Paketspezifikationen werden keine Objektmodule erzeugt.

Der Name eines Objektmoduls wird generiert aus dem Namen der Übersetzungseinheit (Programmname bzw. Paketname) und einem zusätzlichen Zeichen, das die Art des erzeugten Moduls angibt (genaue Beschreibung siehe 4.4).

Standardmäßig werden diese Module nach der Übersetzung in die temporäre EAM-Bindemoduldatei (*OMF) geschrieben, sie können aber auch in einer PLAM-Bibliothek (Programm-Bibliothek) abgelegt werden (siehe 2.6.4, COMPILE-UNIT-Anweisung, Operand MODULE-LIBRARY).

Ein ablauffähiges Pascal-Programm besteht aus einem Hauptprogramm, einer Anzahl von Paketen und dem Pascal-XT Laufzeitsystem. Pakete ohne ein Hauptprogramm sind nicht ablauffähig. Das Binden der Objektmodule zu einem Programm kann statisch mit dem Binder TSOSLNK oder dynamisch mit dem Bindelader DLL erfolgen. Die Binder TSOSLNK und DLL sind in [4] ausführlich beschrieben.

Das Binden eines Programms mit der Testhilfe PATH ist im Abschnitt 9.4 beschrieben.

Referenzen zwischen den Objektmodulen

Jedes Code-Modul eines Pakets oder Hauptprogramms enthält Externreferenzen auf die Code-Module der Pakete, die direkt und indirekt importiert werden und auf die Code-Module der benötigten Laufzeitsystemmodule. Gleichlaufend dazu hat jeder Daten-Modul Externreferenzen auf die Daten-Module der importierten Pakete und Laufzeitsystemmodule. Enthält ein Paket eine Entry-Prozedur, dann enthält das Daten-Modul eine Externreferenz auf das eigene Code-Modul. Externreferenzen von Code- auf Daten-Module existieren grundsätzlich nicht. Das Starter-Modul eines Hauptprogramms enthält Externreferenzen auf das Code- und den Daten-Modul des Hauptprogramms. Die Externreferenzen auf externe (fremdsprachige) Unterprogramme sind in den Daten-Modulen enthalten.

Beispiel 1

Im Bild 6-1 sind die Externreferenzen in einem Pascal-XT Programm ohne Entry-Prozeduren und fremde Unterprogramme dargestellt. Das Hauptprogramm BEISPIEL benötigt das Paket AUSGABE, das seinerseits das Paket TOOLS benutzt. Das Hauptprogramm hat auf TOOLS eine Referenz, da es indirekt importiert wird. Die Externreferenzen auf das Laufzeitsystem sind nicht angegeben.

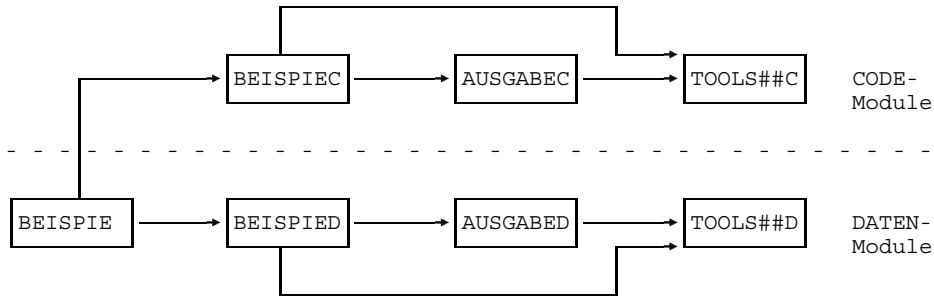


Bild 6-1 Externreferenzen zwischen den Objektmodulen

Beispiel 2

Das Paket TEST benutzt das Paket AUSGABE, das wiederum das externe (fremdsprachige) Modul EXTUP aufruft. In TEST ist eine Entry-Prozedur definiert, folglich enthält das Daten-Modul eine Externreferenz auf das eigene Code-Modul. Die Externreferenz auf das fremde Unterprogramm EXTUP ist im Daten-Modul von AUSGABE enthalten. Die Externreferenzen auf das Laufzeitsystem sind nicht angegeben.

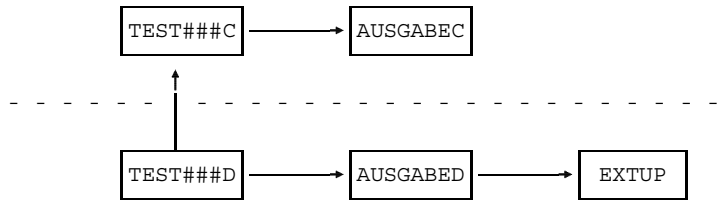


Bild 6-2 Externreferenzen bei Entry-Prozeduren und externen Unterprogrammen

Mehrfach benutzbare Programme

Durch die strikte Trennung von Code- und Daten-Modulen sind Pascal-XT Programme (Hauptprogramm, Pakete und Laufzeitsystem) mehrfach benutzbar. Insbesondere können damit die Code-Module von Programmen oder des Pascal-XT-Laufzeitsystems in den Shared Code oder Common Memory Pool geladen werden (siehe 6.2.2).

Pascal-XT-Laufzeitsystem und ILCS (Inter-Language Communication Services)

In den folgenden Kapiteln wird angenommen, daß das Pascal-XT Laufzeitsystem in der Bindemodulbibliothek $PASLIB-XT vorliegt. In dieser Bindemodulbibliothek befinden sich auch die benötigten Module von ILCS, die beim Binden des Pascal-XT-Laufzeitsystems automatisch miteingebunden werden (siehe 7.1 und das Handbuch ASSEMBH [15]).

Kompatibilität der Versionen von Laufzeitsystem und Compiler

Pascal-XT-Objekte, die von einer älteren Compilerversion erzeugt wurden, können mit dem neuesten Pascal-XT-Laufzeitsystem gebunden werden. Aber Objekte, die mit dem neuesten Compiler erzeugt wurden, dürfen nicht mit einem älteren Laufzeitsystem gebunden werden.
Ein ILCS-fähiges Pascal-XT-Programm setzt das Laufzeitsystem der Version V2.2A voraus.

Statisches Binden

Mit dem statischen Binder TSOSLNK können die Objektmodule eines Programms zu einer ladbaren Phase (Lademodul), zu Großmodulen oder segmentiert gebunden werden.

Beim Binden eines Programms müssen alle benötigten Pakete eingebunden werden, auch wenn auf sie beim Ablauf des Programms nicht zugegriffen wird. Nach dem Start eines Programms werden alle Pakete initialisiert. Fehlt eines der Pakete, dann führt dies zu einem Programmabbruch.

Binden zu einer Phase

Die Objektmodule des Hauptprogramms und der importierten Pakete werden mit dem Laufzeitsystem zu einer Phase (Lademodul) zusammengebunden. Mit der INCLUDE-Anweisung muß das Starter-Modul des Hauptprogramms eingelesen werden, der die Startadresse des Programms enthält. Zum Auflösen der Externreferenzen auf die importierten Pakete und die Module des Laufzeitsystems müssen RESOLVE-Anweisungen auf die Bindemodulbibliotheken angegeben werden, die diese Module enthalten. Der Binder darf keine unaufgelösten Externreferenzen melden (siehe oben). Die erzeugte Phase kann auf allen BS2000 Betriebssystemen ab Version 7.5 ablaufen.

Statisches Binden auf XS-Rechnern

Auf XS-Rechnern kann der Ladepunkt durch Angabe von LOADPT=*XS in den "oberen" Adressraum (größer 16 MB) gelegt werden. Dabei ist allerdings zu beachten, daß externe Unterprogramme ebenfalls XS-fähig sind (siehe Abschnitt 7.2).

Die einfachste Form des statischen Bindens geschieht durch folgende Bindekommandos:

```
/EXEC $TSOSLNK
COMMENT   *** Binden eines Programms zu einem Lademodul ***
PROGRAM   prog                                                    (01)
INCLUDE   progname , modlib                                       (02)
RESOLVE            , modlib                                       (03)
RESOLVE            , $PASLIB-XT                                   (04)
END
```

(01) Die PROGRAM-Anweisung legt den Namen des gebundenen Programms (Lademodul) fest. Unter diesem Namen kann das Programm später gestartet werden.

(02) "progname" ist der Name des Hauptprogramms (Name des Starter-Moduls), das aus der Bindemodulbibliothek "modlib" gelesen wird. Der Binder ermittelt die Startadresse des Programms aus diesem Modul.

(03) Die RESOLVE-Anweisung gibt die Bindemodulbibliothek an, aus welcher der TSOSLNK die Externreferenzen befriedigen soll. Wenn nicht alle benötigten Module in der Bibliothek angegeben sind, dann müssen weitere Bibliotheken angegeben werden.

(04) Die Externreferenzen auf Module des Laufzeitsystems werden aus der Bindemodulbibliothek $PASLIB-XT aufgelöst.

Beispiel 1

Die Objektmodule des Beispielprogramms aus Anhang A.2 seien in der Bindemodulbibliothek LIB.BEISPIEL enthalten. Das Programm wird zuerst zu einem Lademodul mit dem Namen BSP gebunden.

```
/EXEC $TSOSLNK
%  P500 TSOSLNK/190/85-07-10 LOADED
PROGRAM BSP
INCLUDE BEISPIE, LIB.BEISPIEL
RESOLVE        , LIB.BEISPIEL
RESOLVE        , $PASLIB-XT
END
%  T500 PROG BOUND
%  T503 PROG FILE WRITTEN: BSP
%  T504 NUMBER PAM PAGES USED:     41
```

Dann wird das Programm unter dem Namen des Lademoduls BSP gestartet und ausgeführt.

```
/EXEC BSP
%  P500 BSP/000/88-01-21 LOADED
32767
Hexadezimalwert = #00007FFF
0
Hexadezimalwert = #00000000
/
```

Beispiel 2

Dasselbe Programm wird nun so gebunden, daß es im oberen Adressraum (>16 MB) abläuft. In der PROGRAM-Anweisung muß dazu LOADPT=*XS angegeben werden. Die erzeugte Phase kann nur noch auf XS-Rechnern ablaufen.

```
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0C08' OF '87-09-28' LOADED.
PROGRAM BSP,LOADPT=*XS
INCLUDE BEISPIE, LIB.BEISPIEL
RESOLVE        , LIB.BEISPIEL
RESOLVE        , $PASLIB-XT
END
%  LNK0500 PROG BOUND
%  LNK0062 PHASE CAN BE LOADED ON XS SYSTEM ONLY
%  LNK0503 PROG FILE WRITTEN: BSP
%  LNK0504 NUMBER PAM PAGES USED:     41

/EXEC BSP
%  BLS0500 PROGRAM 'BSP', VERSION ' ' OF '88-01-21' LOADED.
777
Hexadezimalwert = #00000309
0
Hexadezimalwert = #00000000
/
```

Vorbinden zu Großmodulen

Der TSOSLNK ermöglicht das Vorbinden mehrerer Objektmodule zu Großmodulen. Diese können vom TSOSLNK in weiteren Bindevorgängen verarbeitet werden oder, wenn sie ein ausführbares Programm repräsentieren, vom DLL geladen und ausgeführt werden.

Das Vorbinden wird z.B. dazu benutzt, um ein Programm zu einem Modul (keine Phase) vorzubinden und in einer Bindemodulbibliothek abzulegen. Es können auch die Code- und Daten-Module eines Programms getrennt zu Großmodulen vorgebunden werden. Die verschiedenen Möglichkeiten werden in den folgenden Abschnitten beschrieben.

Beim Vorbinden müssen die Externreferenzen und Entries sichtbar bleiben. Dies wird über die LINK-SYMBOLS Anweisung des TSOSLNK gesteuert (siehe [4]). Das genaue Vorgehen ist in den Beispielen weiter unten beschrieben.

In ein Großmodul soll das Pascal-XT-Laufzeitsystem nicht eingebunden werden. Beim statischen Binden hat ein Verstoß gegen diese Regel zwar keine negativen Auswirkungen. Denn wenn mehrere Großmodule mit je einem Laufzeitsystem statisch zusammengebunden werden, verhält sich das Programm so, als ob nur ein einziges Laufzeitsystem eingebunden wäre. Beim dynamischen Bindeladen (DLL) oder Nachladen (LINK-Makro) eines solchen Großmoduls mit eigenem eingebundenem Laufzeitsystem würde das Laden des Großmoduls vom DLL mit einer Fehlermeldung abgebrochen.

Vorbinden zu einem einzigen Großmodul

Die Code- und Daten-Module eines Programms werden ohne das Laufzeitsystem zu einem Großmodul vorgebunden. Die Externreferenzen auf das Laufzeitsystem werden erst beim Laden des Programms aufgelöst. Die Laufzeitsystemmodule können im Shared Code oder im Common Memory Pool liegen, oder sie werden aus einer Bibliothek (z.B. $PASLIB-XT) nachgeladen.

```
/EXEC $TSOSLNK      "ab Version 18"
COMMENT   * Binden eines Hauptprogramms ohne Laufzeitsystem *
MODULE    prog  ------------------------------------------------------------ (01)
INCLUDE   progname , modlib  ------------------------------------------------ (02)
RESOLVE           , modlib  ------------------------------------------------ (03)
BIND  ---------------------------------------------------------------------- (04)
```

(01) Die eingelesenen Module werden zu dem Großmodul "prog" zusammengebunden und in die temporäre Bindemoduldatei (*OMF) gespeichert.

(02) Das Starter-Modul "progname" des Hauptprogramms wird aus der Bindemodulbibliothek "modlib" gelesen. Der Binder ermittelt die Startadresse des Programms aus diesem Modul.

(03) Die RESOLVE-Anweisung gibt die Bindemodulbibliothek an, aus welcher der TSOSLNK die Externreferenzen befriedigen soll. Wenn nicht alle benötigten Module in der Bibliothek angegeben sind, dann müssen weitere Bibliotheken angegeben werden.

(04) Die Eingabe an den TSOSLNK wird mit der BIND-Anweisung abgeschlossen, damit der Bindevorgang trotz der nicht aufgelösten Externreferenzen auf das Laufzeitsystem durchgeführt wird.

Beispiel

Die Objektmodule des Beispielprogramms aus Anhang A.2 seien in der Bindemodulbibliothek LIB.BEISPIEL enthalten. Die Objektmodule des Programms werden ohne das Laufzeitsystem zu einem einzigen Großmodul vorgebunden. Die Laufzeitsystemmodule sollen bei der Ausführung des Programm aus der $PASLIB-XT dynamisch nachgebunden werden. Dazu muß die Tasklib auf die $PASLIB-XT eingestellt werden.

```
/ERASE *
/EXEC $TSOSLNK
%  P500 TSOSLNK/190/85-07-10 LOADED
MODULE  BSPMOD
INCLUDE BEISPIE, LIB.BEISPIEL
RESOLVE        , LIB.BEISPIEL
BIND
 UNRESOLVED EXTRNS:
 IP@#OP2D  IP@#ER2D  IP@#RT2D  IP@#OP2C  IP@#ER2C  IP@#RT2C  IP@#OU2C
 IP@#iN2C  IP@#TX2C  IP@#RE2C  IP@#OU2D  IP@#iN2D  IP@#TX2D  IP@#RE2D
%  T056 MODULE BOUND WITH UNRESOLVED EXTERNS
%  T505 MODULE BSPMOD WRITTEN TO EAM OMF

/EXEC $LMS
%  P500 LMS/11B/85-08-02 LOADED
LMS (BS2000) VERSION  V1.1B05
CTL=(RDR)                                                        PRT=(CON)
LIB LIB.GROSSMODUL,OUT,OML,ANY
ADDR *OMF
END
****  E N D   O F   R U N  ****  LMS (BS2000) VERSION  V1.1B05

/SYSFILE TASKLIB=$PASLIB-XT
/EXEC (BSPMOD, LIB.GROSSMODUL)
%  P001 DLL VER 761
%  P500 BSPMOD/001/88-01-21 LOADED
255
Hexadezimalwert = #000000FF
0
Hexadezimalwert = #00000000
/
```

Code- und Daten-Module getrennt vorbinden

Die Code- und Daten-Module eines Programms werden getrennt zu Großmodulen vorgebunden. Das Laufzeitsystem wird nicht mit eingebunden. Durch diese Art des Vorbindens kann beispielsweise das Code-Modul in den Shared Code geladen werden. Dabei ist zu beachten, daß auch die Code-Module des Laufzeitsystems im Shared Code liegen müssen.

Das Starter-Modul eines Hauptprogramms gehört zu den Daten-Modulen. Beim Vorbinden der Daten-Module muß es als erstes Modul mit der INCLUDE-Anweisung eingelesen werden, da er die Startadresse des Programms enthält. In Daten-Modulen können Externreferenzen auf Code-Module bestehen (im Starter-Modul und bei Entry-Prozeduren). Beim Vorbinden der Daten-Module ist darauf zu achten, daß diese Referenzen nicht aufgelöst werden. Das kann mit der EXCLUDE-Anweisung erreicht werden oder indem nur die benötigten Daten-Module mit der INCLUDE-Anweisung eingelesen werden.
Beim Vorbinden der Code-Module ist darauf zu achten, daß die benötigten Entry-Namen (CSECT-Name des Code-Moduls des Hauptprogramms und die Namen der Entry-Prozeduren) sichtbar bleiben. Das muß mit der LINK-SYMBOLS Anweisung des TSOSLNK (siehe [4]) sichergestellt werden.

```
/EXEC $TSOSLNK    "ab Version 18"
COMMENT   *** Binden der Code-Module ***
MODULE    code                                                    (01)
INCLUDE   progcode , modlib                                       (02)
RESOLVE            , modlib                                       (03)
LINK-SYMBOLS *KEEP                                                (04)
BIND                                                              (05)

/EXEC $TSOSLNK    "ab Version 18"
COMMENT   *** Binden des Starter-Moduls und der Daten-Module ***
MODULE    prog                                                    (06)
INCLUDE   progname      , modlib                                  (07)
RESOLVE                 , modlib                                  (08)
EXCLUDE   (progcode,...), modlib                                  (09)
BIND                                                              (05)
```

(01) Die eingelesenen Code-Module werden zu dem Großmodul "code" gebunden und in die temporäre Bindemoduldatei (*OMF) gespeichert.

(02) "progcode" ist der Name des Hauptprogramm-Code-Moduls, der aus der Bindemodulbibliothek "modlib" gelesen wird.

(03) Die Externreferenzen auf die übrigen Code-Module sollen aus der Bindemodulbibliothek "modlib" aufgelöst werden.

(04) Die Externnamen bleiben sichtbar.

(05) Der Bindevorgang wird trotz der nicht aufgelösten Externreferenzen durchgeführt.

(06) Das Starter-Modul des Hauptprogramms und die Daten-Module werden zu dem Großmodul "prog" gebunden und in die temporäre Bindemoduldatei (*OMF) gespeichert.

(07) Das Starter-Modul "progname" wird aus der Bindemodulbibliothek "modlib" gelesen. Der Binder ermittelt die Startadresse des Programms aus diesem Modul.

(08) Alle Externreferenzen auf die übrigen Daten-Module werden aus der Bindemodulbibliothek "modlib" aufgelöst.

(09) Die Externreferenz auf das Hauptprogramm-Code-Modul "progcode" darf nicht aufgelöst werden. Enthält das Anwenderprogramm Entry-Prozeduren, dann sind die Namen der Code-Module, die diese Prozeduren enthalten, ebenfalls in dieser Liste anzugeben. Die EXCLUDE-Anweisung kann entfallen, wenn die Code-Module in einer eigenen Bindemodulbibliothek aufbewahrt werden.

Beispiel

Die Objektmodule des Beispielprogramms aus Anhang A.2 seien in der Bindemodulbibliothek LIB.BEISPIEL enthalten. Die Code- und Daten-Module des Programms werden getrennt ohne die Module des Laufzeitsystems vorgebunden. Die erzeugten Großmodule BSPDAT und BSPCOD werden in der Bindemodulbibliothek LIB.GROSSMODUL abgelegt. Mit dem EXEC-Kommando wird das Programm ausgeführt. Als Parameter sind das Daten-Modul BSPDAT und die Bindemodulbibliothek anzugeben. Das Code-Modul BSPCOD wird vom DLL automatisch nachgeladen. Zur Auflösung der Externreferenzen auf die Laufzeitsystemmodule muß vor der Programmausführung die Tasklib auf die Bibliothek $PASLIB-XT eingestellt werden.

```
/ERASE *
/EXEC $TSOSLNK
%  P500 TSOSLNK/190/85-07-10 LOADED
MODULE BSPCODE
INCLUDE BEISPIEC, LIB.BEISPIEL
RESOLVE         , LIB.BEISPIEL
LINK-SYMBOLS *KEEP
BIND
 UNRESOLVED EXTRNS:
 IP@#OP2C  IP@#ER2C  IP@#RT2C  IP@#OU2C  IP@#iN2C  IP@#TX2C  IP@#RE2C
%  T056 MODULE BOUND WITH UNRESOLVED EXTERNS
%  T505 MODULE BSPCODE WRITTEN TO EAM OMF

/EXEC $TSOSLNK
%  P500 TSOSLNK/190/85-07-10 LOADED
MODULE BSPDATA
INCLUDE BEISPIE , LIB.BEISPIEL
RESOLVE         , LIB.BEISPIEL
EXCLUDE BEISPIEC, LIB.BEISPIEL
BIND
 UNRESOLVED EXTRNS:
 IP@#OP2D  IP@#ER2D  IP@#RT2D  IP@#OU2D  IP@#iN2D  IP@#TX2D  IP@#RE2D
 IP@#RT2C  BEISPIEC
%  T056 MODULE BOUND WITH UNRESOLVED EXTERNS
%  T505 MODULE BSPDATA WRITTEN TO EAM OMF

/EXEC $LMS
%  P500 LMS/11B/85-08-02 LOADED
LMS (BS2000) VERSION  V1.1B05
LIB LIB.GROSSMODUL,OUT,OML,ANY
ADDR *OMF
END
****  E N D   O F   R U N  ****  LMS (BS2000) VERSION  V1.1B05

/SYSFILE TASKLIB=$PASLIB-XT
/EXEC (BSPDATA, LIB.GROSSMODUL)
%  P001 DLL VER 761
%  P500 BSPDATA/001/88-01-21 LOADED
16383
Hexadezimalwert = #00003FFF
0
Hexadezimalwert = #00000000
/
```

Laufzeitsystem vorbinden

Zur Einsparung von Hauptspeicher ist es sinnvoll, wenn für mehrere Anwendungen das Pascal-XT Laufzeitsystem nur ein einziges Mal vorhanden ist. Das kann dadurch erreicht werden, daß die Code-Module des Laufzeitsystems in den Shared Code oder in den Common Memory Pool (siehe [6]) geladen werden. Die Daten-Module sind für jeden Anwender privat und müssen zu seinem Programm gebunden werden.

Zum Vorbinden der Code-Module werden alle Module aus der Laufzeitsystembibliothek PASLIB-XT eingelesen, deren Namen mit "IP@" beginnen und deren achtes Zeichen ein "C" (für Code) ist. Entsprechend werden alle Daten-Module eingelesen, die mit "IP@" beginnen und deren achtes Zeichen ein "D" (für Daten) ist. Die Modulnamen müssen sichtbar bleiben (siehe LINK-SYMBOLS-Anweisung des TSOSLNK).

Die Objektmodule der vordefinierten Pakete (siehe Anhang A.3) sind in der PASLIB-XT enthalten. Sie können bei Bedarf zu den Großmodulen des Laufzeitsystems gebunden werden.

Das Modul #TEST darf nicht mit eingebunden werden. Er besitzt eine Schalterfunktion und hätte die Ausführung eines Programms unter Kontrolle der Testhilfe zur Folge (siehe Kap. 9). Ebenfalls sollten die Module #PATH##C und #PATH##D der Testhilfe Path nur bei Bedarf mit eingebunden werden.

Segmentiertes Binden

Ein Pascalprogramm, bestehend aus einem Hauptprogramm und mehreren Paketen, kann segmentiert (siehe [4]) gebunden werden. Das Binden mit Überlagerungsstruktur ist nur unter bestimmten Bedingungen möglich:

(a) Ins Grundsegment (Wurzel) müssen die Module des Hauptprogramms und aller gemeinsam verwendeten Pakete eingebunden werden. Dazu gehören auch die Module des Laufzeitsystems. Es darf keine Referenzen auf Pakete in anderen Segmenten geben.

(b) Pakete in einem vom Wurzelsegment verschiedenen Segment dürfen nur Pakete aus dem eigenen Segment und/oder dem Wurzelsegment importieren.

(c) Eine Prozedur in einem Segment kann nur über den "External-Entry Mechanismus" aus dem Grundsegment heraus aufgerufen werden. Im Grundsegment muß also eine Prozedur mit der Direktive External definiert werden, die denselben Namen (Prozedur-Bezeichner) und dieselbe Formalparameterliste besitzt wie die zu rufende Entry-Prozedur in dem Segment.
Bedingt durch diesen Aufrufmechanismus sind Einschränkungen bei der Parameterübergabe zu beachten (siehe Kap. 7).

(d) Beim ersten Aufruf einer Entry-Prozedur in einem Segment und beim ersten Aufruf nach einem Segmentwechsel werden alle zu dem Segment gehörigen Pakete initialisiert.

Hinweise

Segmentiert gebundene Programme können nicht mit der Testhilfe PATH getestet werden.
Auf XS-Rechnern können Programme nur im unteren Adressraum (kleiner 16 MB) segmentiert gebunden werden.

Beispiel

In einem einfachen Beispiel wird eine Überlagerungsstruktur mit zwei Segmenten vorgestellt. Zur Vereinfachung besteht das Grundsegment nur aus dem Hauptprogramm TEST und jedes Segment aus einem Paket (OVL1 bzw. OVL2). Jedes Segment kann aber auch aus beliebig vielen Paketen bestehen.
Im Hauptprogramm und den Paketen werden Meldungen ausgegeben, damit der Ablauf des Programms leichter verfolgt werden kann.

```
/EXEC $USERID.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ... LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//D DIRECTORY
//MC (PLAM.EXAMPLE,), *SYSOUT, *STD
//C (, OVL1.SPEC)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG       =    OFF          BY DEFAULT
GENERATE    =    ON           BY DEFAULT
MAP         =    OFF          BY DEFAULT
STANDARD    =    OFF          BY DEFAULT
XREF        =    OFF          BY DEFAULT

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,OVL1.SPEC(*STD,S))

     1      package OVL1;
     2
     3         entry procedure proz1;
     4         entry procedure proz11 ( var i : integer );
     5
     6      end { package OVL1 }.
```

```
************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :         0                     *
*  WARNINGS                 :         0                     *
*  NOTES                    :         0                     *
*  SIZE OF CODE MODULE      :         0 BYTES               *
*  SIZE OF DATA MODULE      :         0 BYTES               *
*  COMPILATION TIME         :     0.228 SEC                 *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C (, OVL2.SPEC)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG       =    OFF          BY DEFAULT
GENERATE    =    ON           BY DEFAULT
MAP         =    OFF          BY DEFAULT
STANDARD    =    OFF          BY DEFAULT
XREF        =    OFF          BY DEFAULT

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,OVL2.SPEC(*STD,S))

     1      package OVL2;
     2
     3         entry procedure proz2 ;
     4
     5      end { package OVL1 }.


************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :         0                     *
*  WARNINGS                 :         0                     *
*  NOTES                    :         0                     *
*  SIZE OF CODE MODULE      :         0 BYTES               *
*  SIZE OF DATA MODULE      :         0 BYTES               *
*  COMPILATION TIME         :     0.068 SEC                 *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C (, OVL1.BODY)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG       =    OFF          BY DEFAULT
GENERATE    =    ON           BY DEFAULT
MAP         =    OFF          BY DEFAULT
STANDARD    =    OFF          BY DEFAULT
XREF        =    OFF          BY DEFAULT
```

```
LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)

               ($USERID.PLAM.EXAMPLE,OVL1.SPEC(*STD,S))

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,OVL1.BODY(*STD,S))

     1      package body OVL1 ( output ) ;
     2
     3      procedure proz1;
     4      begin
     5        writeln ('- Ausfuehrung von OVL1.proz1');
     6      end;
     7
     8      procedure proz11 ( var i : integer ) ;
     9      begin
    10        writeln ('- Ausfuehrung von OVL1.proz11; i = ', i:1);
    11      end;
    12
    13      begin
    14        writeln ('- Initialisierung des Pakets OVL1');
    15      end { package body OVL1 }.


************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED         :        0                      *
*  WARNINGS                :        0                      *
*  NOTES                   :        0                      *
*  SIZE OF CODE MODULE     :      948 BYTES                *
*  SIZE OF DATA MODULE     :      236 BYTES                *
*  COMPILATION TIME        :    0.242 SEC                  *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C (, OVL2.BODY)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT
```

```
LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)

               ($USERID.PLAM.EXAMPLE,OVL2.SPEC(*STD,S))

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,OVL2.BODY(*STD,S))

     1      package body OVL2 ( output );
     2
     3      procedure proz2;
     4      begin
     5        writeln ('- Ausfuehrung von OVL2.proz2');
     6      end;
     7
     8      begin
     9        writeln ('- Initialisierung des Pakets OVL2');
    10      end { package body OVL2 }.


************************************************************
*                   COMPILATION SUMMARY                    *
************************************************************
*  ERRORS DETECTED         :        0                      *
*  WARNINGS                :        0                      *
*  NOTES                   :        0                      *
*  SIZE OF CODE MODULE     :      676 BYTES                *
*  SIZE OF DATA MODULE     :      168 BYTES                *
*  COMPILATION TIME        :    0.216 SEC                  *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C (, OVLTEST.PROG)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,OVLTEST.PROG(*STD,S))

     1      program OVLTEST ( input , output );
     2
     3      var k : integer;
     4
     5      procedure proz1;                        external;
     6      procedure proz11 ( var i : integer ); external;
     7      procedure proz2;                        external;
     8
```

```
 9      begin
10         writeln ('- Start des Hauptprogramms');
11         proz1;                { Aufruf von OVL1.proz1  }
12         k := 1024;
13         proz11 ( k );         { Aufruf von OVL1.proz11 }
14         proz2;                { Aufruf von OVL2.proz2  }
15         proz1;                { Aufruf von OVL1.proz1  }
16         writeln ('- Ende des Hauptprogramms');
17      end { program OVLTEST }.


************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED         :        0                      *
*  WARNINGS                :        0                      *
*  NOTES                   :        0                      *
*  SIZE OF CODE MODULE     :      740 BYTES                *
*  SIZE OF DATA MODULE     :      116 BYTES                *
*  COMPILATION TIME        :    0.229 SEC                  *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//END
   END OF THE PASCAL SESSION  -  USED TIME = 1.247 SECONDS
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0D10' OF '90-05-10' LOADED.
PROG OVLTEST, FILENAM=PH.OVL,CONTROL=Y
COMMENT ***     Wurzel    ***
INCLUDE OVLTEST , PLAM.EXAMPLE
COMMENT ***   Segment 1   ***
OVERLAY K1,O1
TRAITS  OVL1###C, RMODE=24
TRAITS  OVL1###D, RMODE=24
INCLUDE OVL1###C, PLAM.EXAMPLE
INCLUDE OVL1###D, PLAM.EXAMPLE
COMMENT ***   Segment 2   ***
OVERLAY K1,O2
TRAITS  OVL2###C, RMODE=24
TRAITS  OVL2###D, RMODE=24
INCLUDE OVL2###C, PLAM.EXAMPLE
INCLUDE OVL2###D, PLAM.EXAMPLE
COMMENT *** Grundsegment ***
RESOLVE         , $USERID.PASLIB-XT
RESOLVE         , PLAM.EXAMPLE
END
PROG BOUND
PROGRAM FILE WRITTEN : PH.OVL
NUMBER PAM PAGES USED:     49
```

```
/EXEC PH.OVL
%  BLS0500 PROGRAM 'OVLTEST', VERSION ' ' OF '90-11-27' LOADED.
- Start des Hauptprogramms
- Initialisierung des Pakets OVL1
- Ausfuehrung von OVL1.proz1
- Ausfuehrung von OVL1.proz11; i = 1024
- Initialisierung des Pakets OVL2
- Ausfuehrung von OVL2.proz2
- Initialisierung des Pakets OVL1
- Ausfuehrung von OVL1.proz1
- Ende des Hauptprogramms
```

Dynamisches Binden

Das dynamische Binden erfolgt mit dem dynamischen Bindelader DLL, der die Objektmodule in den Hauptspeicher lädt, bindet und das Programm startet.

Externreferenzen, z.B. auf importierte Pakete, werden durch den Autolink-Mechanismus des DLL (siehe [4]) befriedigt.

Hinweis

Der Autolink-Mechanismus betrachtet nicht die temporäre Bindemoduldatei (*OMF) der Task.

Dynamisches Bindeladen innerhalb des Programmiersystems

Das Bindeladen erfolgt innerhalb des Programmiersystems implizit durch die Anweisung RUN-PROGRAM (siehe 2.6.11). Mit ihr kann ein Programm aus der temporären Bindemoduldatei (*OMF) des Benutzerprozesses oder aus einer angegebenen Bindemodulbibliothek geladen und gebunden werden. Das Programm benutzt das Laufzeitsystem des Programmiersystems mit.

Dynamisches Bindeladen außerhalb des Programmiersystems

Das dynamische Bindeladen erfolgt implizit durch das EXEC- oder LOAD-Kommando. Das Laden aus der temporären Bindemoduldatei (*OMF) ist nicht möglich. Es muß aus einer Bindemodulbibliothek geladen werden.

```
/EXEC (prog,modlib)
```

Das auszuführende Programm "prog" wird aus der Bindemodulbibliothek "modlib" geladen. "prog" bezeichnet das Starter-Modul des Hauptprogramms oder den Namen des Großmoduls, zu dem das Programm vorgebunden wurde. Die Objektmodule importierter Pakete können aus dem SHARED CODE hinzugebunden werden, oder aus Bindemodulbibliotheken, die über den Autolink-Mechanismus des DLL "erreichbar" sind.

Dynamisches Bindeladen auf XS-Rechnern

Pascal-XT Programme und das Pascal-XT Laufzeitsystem sind XS-fähig, d.h. sie können überall im 31-Bit Adressraum ablaufen.

Enthält ein Pascal-XT Programm externe Unterprogramme (z.B. Assembler-Unterprogramme), dann müssen diese ebenfalls XS-fähig sein, wenn das Programm im "oberen" Adressraum (größer 16 MB) ablaufen soll. Ein Programm wird i.a. undefiniert beendet, wenn dies nicht gewährleistet ist.

Beispiel

Beim Aufruf des Programms wird dem DLL durch den Parameter PROG-MOD mitgeteilt, daß das Programm auch im oberen Adressraum ablaufen kann.

```
/SYSFILE TASKLIB=$PASLIB-XT
/EXEC (BEISPIE,LIB.BEISPIEL),PROG-MOD=ANY
%  BLS0001 DLL VER 917
%  BLS0517 MODULE 'BEISPIE' LOADED
55
Hexadezimalwert = #00000037
0
Hexadezimalwert = #00000000
/
```

Programm-Beendigungscode

Ein ausgeführtes Pascal-XT Programm wird entsprechend den Konventionen für das Programm-Beendigungsverhalten im BS2000 beendet. Die Beschreibung der Konvention ist im Abschnitt 2.4 nachzulesen.

Der Beendigungscode kann folgende Werte annehmen:

’0’ Das Programm ist fehlerfrei gelaufen und wurde normal beendet.

’2’ Das Programm ist fehlerhaft gelaufen. Der Spin-Off-Mechanismus wird aktiviert.

’3’ Das Programm ist wegen eines Fehlers im Laufzeitsystem beendet worden. Der Spin-Off-Mechanismus wird aktiviert.

Die Programminformation ist standardmäßig mit 3 Leerzeichen (Blanks) besetzt, kann im Programm jedoch im Rahmen der Konvention programmspezifisch besetzt werden. Das Setzen der Programminformation im Programm erfolgt mit der Prozedur SET_RETURN_CODE aus dem vordefinierten Paket BS2000CALLS. Es muß beim Aufruf auch der Beendigungscode mit angegeben werden.

Lizenzschutz für das Pascal-XT-Laufzeitsystem

Ab Version V2.2A wird beim Start eines in Pascal-XT implementierten Programms mit eingebundenem Laufzeitsystem überprüft, ob für das Programm eine Ablaufberechtigung besteht. Eine Ablaufberechtigung liegt dann vor, wenn auf der Kennung $TSOS ein Pascal-XT-Lizenzmodul vorhanden ist, dessen Version größer oder gleich der Version des im Programm eingebundenen Laufzeitsystems ist. Wenn das Lizenzmodul nicht die richtige Version hat oder nicht vorhanden ist, wird folgende Meldung ausgegeben und das Programm abgebrochen:

```
PASCAL-XT: MISSING LICENSE KEY FOR RUNTIME SYSTEM.
```

Das Lizenzmodul ist Bestandteil der beiden Liefereinheiten Pascal-XT Programmiersystem und Pascal-XT Laufzeitsystem. Wer in Pascal-XT implementierte Software einsetzen will, aber kein Pascal-XT Programmiersystem besitzt, braucht das Pascal-XT Laufzeitsystem.

Sprachverknüpfungen

Ein Programm besteht im allgemeinen aus mehreren Programmteilen, die in verschiedenen Programmiersprachen implementiert sein können.

Ein Pascal-XT-Hauptprogramm oder eine Pascal-XT-Entry-Prozedur mit den jeweils darin aufgerufenen Unterprogrammen wird als Pascal-XT-Programmteil bezeichnet. Der Aufruf zwischen Pascal-XT-Programmteilen und fremdsprachigen Programmteilen erfolgt über die im folgenden beschriebenen Schnittstellen.

Die Grenzen der Pascal-XT-Programmteile sind sowohl für die Behandlung von Fehlern (siehe 10.2) als auch für das Verhalten der Testhilfe PATH (siehe 9.1.4.3 und 9.1.4.4) von Bedeutung.

Pascal-XT unterstützt auf einfache Weise solche Sprachverknüpfungen:

– Aufruf eines fremdsprachigen Unterprogramms

  Das Unterprogramm muß im Pascal-XT-Programm als Funktion bzw. Prozedur mit der Direktive EXTERNAL deklariert und wie eine Pascal-XT-Prozedur bzw. -Funktion aufgerufen werden.

– Aufruf von Pascal-XT-Prozeduren durch fremdsprachige Programme

  Die Pascal-XT-Prozedur muß in einer Paketspezifikation (package) deklariert werden und sie muß mit dem Schlüsselwort ENTRY als Entry-Prozedur gekennzeichnet werden (siehe 7.3). Pascal-XT-Funktionen können *nicht* aus fremdsprachigen Programmen aufgerufen werden. Entry-Prozeduren können auch aus Pascal-XT-Programmen aufgerufen werden.

Fremdsprachige Unterprogramme werden ab Pascal-XT V2.2A gemäß den ILCS-Konventionen (siehe 7.1), in den älteren Versionen gemäß der bisher gültigen "Standard-Unterprogramm-Schnittstelle" aufgerufen. Bei den Sprachverknüpfungen sind beliebige Aufrufsequenzen ohne irgendeine Einschränkung möglich. Die Namenskonventionen (siehe 4.4) sind zu beachten.

Daneben gibt es noch die Möglichkeit, Assembler-Unterprogramme über einen schnellen Mechanismus aufzurufen (siehe 7.4). Hierfür wird die Direktive INTERNAL verwendet.

Zugriffe auf dieselbe Dateivariable über Sprachgrenzen hinweg sind nicht möglich, da Pascal-XT keine Informationen über den Zustand von Dateien mit externen Unterprogrammen austauschen kann.

Die Programm-Kommunikationsschnittstelle ILCS

ILCS (Inter-Language Communication Services) ist ein Softwareprodukt, das wesentliche Funktionen der Kommunikation zwischen den Programmen einer Ablaufeinheit und zwischen Ablaufeinheit und Betriebssystem sprachübergreifend vereinheitlicht und vereinfacht.

ILCS ist eine Kombination aus Software und Schnittstellen-Konvention:
Es beinhaltet zum einen Laufzeitroutinen, die in einer PLAM-Bibliothek zusammengefaßt sind, zum anderen garantiert ILCS auch die den "Standard-Linkage-Konventionen im BS2000" entsprechende Kommunikationsschnittstelle; d.h. jedes von einem ILCS-fähigen Compiler erzeugte Objektmodul ist entsprechend den Standard-Linkage-Konventionen für die Verknüpfung mit gleich- und verschiedensprachigen Programmen vorbereitet.

Die Bibliothek der ILCS-Laufzeitroutinen wird mit jedem ILCS-fähigen Compiler - als gleichsam zusätzliches Laufzeitsystem - ausgeliefert.

Im einzelnen bietet ILCS folgende Funktionen:

- Konvention zur Verknüpfung verschiedensprachiger Programme
- einheitliche Richtlinien zur Ereignisbehandlung
- Speicherverwaltung (Stack- und Heap-Speicher)
- Behandlung der Programmaske
- Verarbeitung nicht lokaler Sprünge

In diesem Abschnitt wird nur die vom Pascal-XT-Compiler genutzte ILCS-Funktion Programmverknüpfung mit den grundlegenden ILCS-Datenstrukturen beschrieben.

Hinweis

Programme, die mit ILCS-fähigen Compilern übersetzt wurden, werden obligatorisch mittels ILCS zu einem Programmsystem verknüpft. 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-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 Parameteradreßliste
2 - 12	Programmdaten
13	Anfangsadresse des Sicherstellungsbereiches (Save Area) des aufrufenden Programms
14	Adresse des Rückkehrpunktes ins aufrufende Programm
15	Adresse des Einsprungpunktes im aufgerufenen Programm.
PM	Programmaske: Wert aus PCD-Feld "Programmaske"

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 - 1	Returnwerte von Ganzzahl-Funktionen oder undefiniert
2 - 14	wie bei Aufruf-Versorgung
15	undefiniert
PM	Programmaske: Wert aus PCD-Feld "Programmaske"

ILCS-Datenstrukturen

Save Area (Sicherstellungsbereich)

Das aufrufende Programm übergibt die Adresse eines Sicherstellungsbereiches, in der 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.-7. Bit: reserviert 8. Bit = im allgemeinen 0 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 (geketteten) 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 allen Programmiersprachen zur Verfügung steht. Er hat eine Größe von 4096 Byte.
Der erste Teil enthält die von ILCS verwendeten Datenbereiche, u.a. auch (in Byte 148) das Feld "Programmaske", 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.

Initialisierung des Pascal-XT-Laufzeitsystems

Abweichend von der ILCS-Konvention initialisiert Pascal-XT sein Laufzeitsystem selbst. Wenn das Pascal-XT-Hauptprogramm bzw. das erste Pascal-XT-Entry-Unterprogramm ausgeführt wird, initialisiert das Pascal-XT-Laufzeitsystem sich selbst und anschließend ILCS. Jedes ILCS-konforme fremdsprachige Unterprogramm, das aus einem beliebigen Pascal-XT-Programmteil aufgerufen wird, kann also davon ausgehen, daß es eine initialisierte ILCS vorfindet.

Programmasken-Behandlung durch ILCS

Die Programmaske für den Programmablauf wird im Wege der Initialisierung auf den Wert des PCD-Feldes "Programmaske" (vorbelegt mit X'0C') gesetzt. Wird sie während des Programmablaufs verändert, muß sie vor dem nächsten Programmaufruf bzw. -rücksprung auf den Wert des PCD-Feldes "Programmaske" zurückgesetzt werden.

Parameterübergabe in ILCS-Programmsystemen

Die Semantik 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.

Compiler	Datentypen			
	Binär Wort	Gleitpunkt Wort	Gleitpunkt Doppelwort	String
COBOL85	PIC S9(i) COMP SYNCHRONIZED 5<=i<=9	COMP-1	COMP-2	USAGE DISPLAY
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)
C	long	float	double	char <var> [<size>]
Columbus-Assembler	F	E	D	C

Pascal-XT unterstützt auch weiterhin die von ILCS nicht vorgesehenen Parametertypen "short_integer" und "char". In COBOL85 heißen die entsprechenden Typen "PIC S9(4) USAGE COMP" und "PIC X".

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

Die Übergabe erfolgt "by reference", d.h. es wird die Adresse des Parameters, nicht sein Wert, übergeben. 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 "ILCS-Registerkonventionen").

Übergabe von Funktions-Returnwerten

Returnwerte von Ganzzahl-Funktionen werden in den Registern 0 und 1, Returnwerte von Gleitpunkt-Funktionen werden, beginnend mit dem Gleitpunkt, 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 in den einzelnen Programmiersprachen freigestellt.

Binden von ILCS-Programmsystemen

Grundsätzlich gilt:
Der Anwender muß so binden, daß das ILCS-Modul IT0INITS genau einmal im Programmsystem enthalten ist. Dieses Modul befindet sich, wie auch das Pascal-XT-Laufzeitsystem, in der Bibliothek $PASLIB-XT. Näheres siehe Kapitel 6.

Anschluß fremdsprachiger Unterprogramme

Prozeduren, die in anderen Sprachen geschrieben sind, müssen in der Prozedurdeklaration im Pascal-Programm (Paket) durch eine Direktive gekennzeichnet werden.

Deklaration des fremdsprachigen Unterprogramms im Pascal-Programm

Für ein fremdsprachiges Unterprogramm muß im Pascal-XT-Programm der Prozedur- bzw. Funktionskopf wie folgt deklariert werden:

```
PROCEDURE    Prozedur-Bezeichner (Formalparameterliste); direktive;
```
oder
```
FUNCTION     Funktions-Bezeichner (Formalparameterliste): typ; direktive;
```

Für "direktive" ist einer der folgenden Werte anzugeben:

`EXTERNAL`	für alle fremden Programmiersprachen, bei XS-Rechnern in jedem Fall notwendig (siehe Besonderheiten bei XS-Rechnern); Ab Pascal-XT V2.2A wird empfohlen, nur noch diese Direktive zu verwenden. Es werden jedoch weiterhin auch die nachfolgenden Direktiven unterstützt.
`FORTRAN`	für Fortran-Unterprogramme
`COBOL`	für Cobol-Unterprogramme
`C`	für C-Unterprogramme

Die Unterprogrammdeklaration ist in einer Paketspezifikation (package), einer Paketimplementierung (package body), oder einem Hauptprogramm (program) erlaubt. Zu der Unterprogrammdeklaration darf es keine Identifikation (Unterprogrammblock) geben.

Ergebnistypen von externen Funktionen

Der Ergebnistyp bei Funktionen kann ein Ordinal-Typ, oder ein Real-Typ, oder ein Zeigertyp sein. Ordinalwerte und Zeigerwerte müssen im Register R0 und Realwerte müssen im Register F0 zurückgeliefert werden. Externe Funktionen sind nur mit den Direktiven EXTERNAL und FORTRAN möglich.

Formale Parameter

Als formale Parameter sind Variablen- und Werteparameter zugelassen. Für Werteparameter werden vor dem Aufruf Kopien angelegt und deren Adressen übergeben.

Interpretation der Direktiven

Die verschiedenen Direktiven führen zu geringfügigen Interpretationsunterschieden. Die folgende Tabelle gibt an, wie das Pascal-XT-System den Inhalt von Register 0 und das höchstwertige Bit im letzten Parameter in Abhängigkeit von der Direktive besetzt.

Direktive	Register 0 enthält	"Erstes Bit"
EXTERNAL	Anzahl der Parameter	ist gelöscht "0"
FORTRAN	Anzahl der Parameter	ist gesetzt "1"
COBOL	Anzahl der Parameter	ist gesetzt "1"

Bei parameterlosen Prozeduren werden Register 0 und Register 1 mit dem Wert 0 besetzt.

Besonderheiten bei XS-Rechnern

Für XS-fähige externe Unterprogramme, die über die Standard-Unterprogramm-Schnittstelle aufgerufen werden, muß immer die Direktive EXTERNAL angegeben werden, unabhängig davon in welcher Sprache die Unterprogramme geschrieben sind.

Aktionen vor dem Aufruf eines fremdsprachigen Unterprogramms

Bei dem Aufruf eines externen Unterprogramms werden folgende Aktionen durchgeführt:

- Die Register werden besetzt wie in 7.1 beschrieben.
- Die Pascal-XT-Fehlerbehandlung bleibt angemeldet, damit im fremdsprachigen Unterprogramm auftretende Fehler an den Pascal-XT-Programmteil propagiert werden können. Wie sich die fremdsprachigen Unterprogramme verhalten, ist im jeweiligen Handbuch nachzulesen.
- Die Pascal-XT-INTR-Behandlung wird intern deaktiviert, aber nicht bei ILCS abgemeldet. Das Ereignis INTR, das durch (K2) /INTR ausgelöst wird, wird deshalb nur dann als Break_Error interpretiert und propagiert, wenn es während des Ablaufs eines Pascal-XT-Programmteils auftritt.
- Das Unterprogramm wird nach ILCS-Konvention aufgerufen wie in 7.1 beschrieben.

Aktionen im gerufenen fremdsprachigen Unterprogramm

Die im gerufenen Unterprogramm notwendigen Aktionen sind den Handbüchern der jeweiligen Sprachen zu entnehmen. Eventuell unterschiedliche Speicherdarstellungen von Parameterwerten müssen beachtet werden.
Das gerufene Unterprogramm ist für das Retten der benötigten Register in der Save Area zuständig.

Aktion nach Rückkehr aus dem externen Unterprogramm

Die Pascal-XT-INTR-Behandlung wird intern wieder aktiviert.

Bis V2.1 gilt:
Das Pascal-XT-Programm nimmt den korrekten Ablauf des fremdsprachigen Unterprogramms an, wenn es nach Ablauf des Unterprogramms die Kontrolle zurückerhält und kein Fehler an es weitergereicht wird. Soll ein Return-Code zurückgegeben werden, dann muß dies über einen zusätzlichen Parameter erfolgen.

Ab V2.2A gilt:
Ab können Fehler über Sprachgrenzen propagiert werden, d.h. ein in einem fremdsprachigen Unterprogramm auftretender Fehler muß nicht mehr zum Programmabbruch führen (siehe Kapitel 10, Laufzeitfehler und Fehlerbehandlung). Welche Möglichkeiten der Fehlerpropagierung die Programmiersprachen der Unterprogramme bieten, ist der jeweiligen Sprachbeschreibung zu entnehmen.

Hinweis

Die Namen aller in einem Programm (Hauptprogramm und Pakete) verwendeten externen Unterprogramme müssen eindeutig sein (bis BS2000 V11 gilt: Unterscheidung in den ersten 8 Zeichen). Zur Übersetzungszeit wird dies nicht überprüft.

Beispiele

1. Sprachverknüpfung zwischen Pascal-XT und Cobol

Ein Pascal-XT-Hauptprogramm ruft ein Cobol-Unterprogramm auf und übergibt als Variablenparameter eine Record-Variable mit zwei Komponenten. Diese Komponenten werden im Cobol-Unterprogramm auf dem Bildschirm ausgegeben und anschließend wird ihr Wert verändert. Nach der Rückkehr ins Hauptprogramm werden die beiden Komponenten, deren Wert nun global verändert ist, ausgegeben.

Quellcode des Pascal-XT-Hauptprogramms "PASHAUPT":

```
PROGRAM PASHAUPT (INPUT,OUTPUT);
TYPE
   RA   =   1..8;
   STR8 =   ARRAY [RA] OF CHAR;
   SATZ =   RECORD
            A  :  INTEGER;
            B  :  STR8;
            END;
CONST
   AGGR =   STR8('T','E','S','T',' ':4); ------------------------------ (01)
VAR
   RC   :   SATZ;
   I    :   INTEGER;
PROCEDURE COBUPROG (VAR PAR:SATZ); EXTERNAL; ------------------------- (02)
```

```
BEGIN   { PASHAUPT }
    RC.A := 1111;
    RC.B := AGGR;
    COBUPROG (RC);                                                    (03)
    WITH RC DO BEGIN
       WRITELN ('A:',A);
       WRITE ('B:');
       FOR I := FIRST(RA) TO LAST(RA) DO
          WRITE (B[I]);
       WRITELN;
       END;
END.
```

Quellcode des Cobol-Unterprogramms "COBUPROG":

```
       ID DIVISION.
       PROGRAM-ID. COBUPROG. ------------------------------------------ (04)
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           TERMINAL IS SCREEN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 TEILDR    PIC 9(8). ------------------------------------------ (05)
       LINKAGE SECTION.
       01  SATZ.
           02  TEILB   PIC S9(8) COMP. ------------------------------- (06)
           02  TEILA   PIC X(8). ------------------------------------- (07)
       PROCEDURE DIVISION USING SATZ.
       ANF.
           MOVE TEILB TO TEILDR. ---------------------------------------- (08)
           DISPLAY "TEILB: " TEILDR UPON SCREEN. ------------------------ (09)
           DISPLAY "TEILA: " TEILA UPON SCREEN.
           MOVE "XXXXZZZZ" TO TEILA. ------------------------------------ (10)
           ADD 1 TO TEILB. ---------------------------------------------- (11)
       BACK.
           EXIT PROGRAM.
```

Aufruf der ablauffähigen Phase "PASCOB" und Ablaufprotokoll:

```
/EXEC PASCOB ------------------------------------------------------------ (12)
%  BLS0500 PROGRAM 'PASCOB', VERSION ' ' OF '...' LOADED
TEILB: 00001111 --------------------------------------------------------- (13)
TEILA: TEST
A:        1112 ------------------------------------------------------------ (14)
B:XXXXZZZZ
```

Erläuterung:

(01) Definition der Konstante AGGR vom Typ STR8: sie wird über ein Aggregat (siehe [1],9.5) mit der Zeichenkette "TEST " als Wert besetzt.

(02) Deklaration der fremdsprachigen Prozedur COBUPROG.

(03) Aufruf des Unterprogramms COBUPROG: die Variable RC vom Typ Satz wird übergeben. Die beiden Komponenten von RC enthalten die Werte 1111 und "TEST ".

(04) Name des Cobol-Unterprogramms: COBUPROG

(05) Deklaration der Variablen TEILDR vom Typ PIC 9(8).

(06) Die übergebene Variable vom Typ SATZ hat zwei Komponenten:
Die erste ist vom Typ PIC S9(8), der in PASCAL-XT dem Typ long_integer entspricht. Ihr Wert wird der Variablen TEILB zugewiesen. TEILB enthält somit den Wert 1111.

(07) Die zweite ist vom Typ PIC X(8), der in PASCAL-XT dem Typ array [8] of char entspricht. Ihr Wert wird der Variablen TEILA zugewiesen. TEILA enthält somit als Wert die Zeichenkette "TEXT ".

(08) Der Wert der Variablen TEILB wird der Variablen TEILDR zugewiesen. TEILDR enthält somit den Wert 1111.

(09) Name und Wert der Variablen TEILB und TEILA werden ausgegeben. Siehe (13).

(10) Der Variablen TEILA wird als Wert die Zeichenkette "XXXXZZZZ" zugewiesen.

(11) Der Wert der Variablen TEILB wird um 1 erhöht.

(12) Starten der ablauffähigen Phase PASCOB.

(13) Ausgaben aus dem Cobol-Unterprogramm: Name und Wert der Variablen TEILB und TEILA. Siehe (09).

(14) Ausgaben nach der Rückkehr ins Hauptprogramm: Name und Wert von A und B.

2. Sprachverknüpfung zwischen Pascal-XT und Assembler

Ein Pascal-XT-Hauptprogramm ruft ein Assembler-Unterprogramm auf und übergibt zwei Wertparameter, die addiert werden und einen Variablenparameter, in dem das Ergebnis der Addition gespeichert wird. Das Ergebnis wird nach der Rückkehr ins Hauptprogramm auf dem Bildschirm ausgegeben.

Quellcode des Pascal-XT-Hauptprogramms "PASASS":

```
PROGRAM PASASS (OUTPUT);
PROCEDURE ASSUP (A,B : INTEGER; VAR ERG : INTEGER); EXTERNAL; ———————— (01)
VAR
   ERG   :  INTEGER;
BEGIN
   ASSUP (5,7,ERG); ———————————————————————————————————————————————— (02)
   WRITELN (ERG);
END.
```

Quellcode des Assembler-Unterprogramms "ASSUP":

```
ASSUP     START ------------------------------------------------------------ (03)
ASSUP     AMODE  ANY
ASSUP     RMODE  ANY
PCD#AREA  DSECT
          IT0PCD PCD# ------------------------------------------------------ (04)
ASSUP     CSECT
ASS#VOR   STM    14,12,12(13) ------------------------------------------------ (05)
          BALR   5,0 ------------------------------------------------------- (06)
          USING  *,5
          ST     13,SAVPSA
          L      6,SAVPCD-SAVAI(,13)
          ST     6,SAVPCD
          LA     13,SAVAI
          ST     13,PCD#ASA-PCD#EYEC(,6)
          LM     2,4,0(1) ---------------------------------------------------- (07)
          L      7,0(2) ------------------------------------------------------ (08)
          L      8,0(3) ------------------------------------------------------ (09)
          AR     7,8 --------------------------------------------------------- (10)
          ST     7,0(4) ------------------------------------------------------ (11)
ASS#NACH  L      6,SAVPCD ---------------------------------------------------- (12)
          L      13,SAVPSA
          ST     13,PCD#ASA-PCD#EYEC(,6)
          LM     14,12,12(13)
ASS#RET   BR     14 ---------------------------------------------------------- (13)
          DS     0F
SA        IT0VSA SA ------------------------------------------------------- (14)
          END
```

Aufruf der ablauffähigen Phase "PASASSEM" und Ablaufprotokoll:

```
 /EXEC PASASSEM ---------------------------------------------------------- (15)
 %  BLS0500 PROGRAM 'PASASSEM', VERSION ' ' OF '...' LOADED
           12 ---------------------------------------------------------- (16)
```

Erläuterung:

(01) Deklaration der externen Prozedur ASSUP.

(02) Aufruf der Prozedur ASSUP, wobei die Wertparameter 5 und 7 und der Variablenparameter ERG übergeben werden. In der Variablen ERG soll das Ergebnis der Addition von 5 und 7 abgespeichert werden, die das Assembler-Unterprogramm ausführt.

(03) Name des Assembler-Unterprogramms: ASSUP

(04) IT0PCD ist ein ILCS-Makro, das die Felder der PCD-Area (Prosys Common Data Area) generiert.

(05) Register und SA (Save Area) des Rufers sicherstellen.

(06) Versorgen der Basisadreßregister.

(07) Versorgen von Register 1 mit der Adresse der Parameterliste.

(08,09) Übernahme der übergebenen Parameter.

(10) Addition der übergebenen Werteparameter.

(11) Versorgen des übergebenen Variablenparameters mit dem Ergebnis der Addition.

(12) Zustand der Register wiederherstellen, in dem sie vor Aufruf des Assembler-Unterprogramms waren.

(13) Rückkehr zum Rufer, also zum Pascal-XT-Hauptprogramm.

(14) Das ILCS-Makro IT0VSA generiert die ILCS-konforme SA.

(15) Starten der ablauffähigen Phase PASASSEM.

(16) Ausgabe des Werts der Variablen ERG nach Rückkehr ins Hauptprogramm.

Aufruf durch fremdsprachige Programme

Mit dem Schlüsselwort ENTRY gekennzeichnete Prozeduren in einer Paketspezifikation (package) können von beliebigen Programmen (auch aus Pascal-XT-Programmen) heraus aufgerufen werden. Rekursive Aufrufe von Entry-Prozeduren sind zugelassen.

Anforderungen an die Pascal-Prozedur

- Die Prozedur muß in einer Paketspezifikation (package) deklariert werden. In der Paketimplementierung (package body) muß die zugehörige Prozeduridentifikation stehen. Pascal-Funktionen können nicht von externen Programmen aufgerufen werden.
- Die Prozedur muß durch das Schlüsselwort ENTRY als Entry-Prozedur gekennzeichnet werden:

  ENTRY PROCEDURE Prozedur-Bezeichner (Formalparameterliste);

  Die Prozeduridentifikation in der Paketimplementierung darf dieses Schlüssselwort nicht enthalten.
- Als formale Parameter sind nur Variablen-Parameter (VAR-Parameter) zugelassen.

Parameterübergabe

Die Anzahl der akzeptierten Parameter ist durch die Deklaration der Pascal-XT-Prozedur festgelegt. Angaben im Register 0 über Anzahl der Parameter und/oder das "Erste Bit" im letzten Parameter der Parameteradreßliste werden ignoriert.

Gleitpunktregister werden im Pascal-Programm nicht gerettet.

Aktionen im externen Programm vor dem Aufruf

Für den Aufruf eines Pascal-XT-ENTRY-Unterprogramms ist es nicht erforderlich, daß der externe Rufer ILCS unterstützt, da das Pascal-XT-Laufzeitsystem sich selbst initialisiert und die Initialisierung von ILCS veranlaßt. Es wird nur vorausgesetzt, daß die Save Area des Rufers rückwärts verkettet ist mit der Save Area seines Rufers. Ist dies nicht der Fall, muß der Anwender ein geeignetes Adaptermodul vorsehen.

Hinweis für Assembler-Programmierer:
Wenn ein Pascal-XT-Großmodul mit Pascal-XT-Entry-Prozeduren dynamisch nachgeladen wird, muß IT0ININ nur aufgerufen werden, wenn darin mathematische Funktionen verwendet werden.

Aktionen in der aufgerufenen Pascal-XT-Entry-Prozedur

- Die Register des Rufers werden in der übergebenen Save Area gesichert.
- Die Lizenzprüfung des Pascal-XT-Laufzeitsystems wird durchgeführt (siehe 6.6).
- Wenn ILCS noch nicht initialisiert ist, wird es initialisiert.
- Wenn das Pascal-XT-Laufzeitsystem noch nicht initialisiert ist, wird es initialisert. Dabei wird die Benutzer-STXIT-Routine für das Ereignis INTR bei ILCS angemeldet. Stack und Heap werden eingerichtet.
- Die Save Area der Pascal-XT-Entry-Prozedur wird nach ILCS-Konventionen angelegt und mit der Save Area des Rufers verkettet.
- Die Pascal-XT-Fehlerbehandlung wird bei ILCS angemeldet, falls dies noch nicht geschehen ist.

  Die Pascal-XT-INTR-Behandlung wird bei jedem Aufruf aktiviert. Tritt während des Ablaufs eines Pascal-XT-Programmteils das Ereignis INTR (ausgelöst durch K2 /INTR) auf, wird es als Break_Error interpretiert und propagiert.
- Die Pascal-XT-Fehlerbehandlung wird bei ILCS angemeldet.
- Das Paket, in dem sich die Entry-Prozedur befindet und alle von ihm importierten Pakete werden initialisiert, falls dies noch nicht geschehen ist. Dadurch werden auch alle von diesem Paket referierten Pakete initialisiert, soweit das noch nicht geschehen ist.
- Die Entry-Prozedur wird aufgerufen.

Aktionen nach Beendigung einer Entry-Prozedur

- Lokale Dateien in Entry-Prozeduren und in darin aufgerufenen Prozeduren und Funktionen werden automatisch geschlossen.
- Der Stack wird logisch freigegeben. Physikalisch wird aber ein Speicherbereich für Folgeaufrufe weiter behalten.
- Die Pascal-XT-INTR-Behandlung wird intern deaktiviert, aber nicht bei ILCS abgemeldet.
- Die Save Area der Pascal-XT-Entry-Prozedur wird ausgekettet und freigegeben. Die Pascal-XT-Fehlerbehandlung wird dadurch automatisch bei ILCS abgemeldet.
- Normale Rückkehr zum Rufer, wenn kein Fehler aufgetreten ist.

Hinweis

Nach der Rückkehr aus einer Entry-Prozedur existiert das Pascal-XT-Laufzeitsystem in dem beim Rücksprung gültigen Zustand weiter.

Der Heap wird nicht freigegeben. Der Anwender kann die Freigabe des Heaps aber explizit durch "Mark" und "Release" bzw. mit Funktionen aus dem vordefinierten Paket HEAPSUPPORT (siehe A.8) selbst steuern.

Die in den Paketen definierten und geöffneten externen Dateien werden nicht geschlossen. Wenn der Heap nicht freigegeben wird, werden auch die dort befindlichen Dateien nicht geschlossen.

Fehlerbehandlung

Tritt innerhalb des Pascal-XT-Programmteils ein Fehler auf, der dort nicht behandelt wird, wird der Fehler an den Rufer weiterpropagiert (siehe Kapitel 10, Fehlerbehandlung und Laufzeitfehler). In diesem Fall bleibt der Pascal-XT-Programmteil für Diagnosezwecke in dem Zustand erhalten, der beim Auftreten des Fehlers galt.

Wenn der Fehler anschließend im rufenden Programm behandelt wird, dann wird der Pascal-XT-Programmteil über ILCS durch eine sog. Unterprogramm-Termination-Routine normal beendet.

Wenn der Fehler vom Rufer nicht behandelt und das Programm abgebrochen wird, dann wird der Pascal-XT-Programmteil nicht normal beendet. Dateien werden nicht geschlossen.

Beispiel

Sprachverknüpfung zwischen Cobol und Pascal-XT

Ein Cobol-Hauptprogramm ruft eine Pascal-XT-Entry-Prozedur auf und übergibt dabei zwei Variablenparameter. Die Entry-Prozedur gibt die übergebenen Variablen auf dem Bildschirm aus und verändert anschließend global ihre Werte. Nach der Rückkehr ins Cobol-Hauptprogramm werden die nun veränderten Variablen ausgegeben.

Quellcode des Cobol-Hauptprogramms "COBPAS":

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      COBPAS. ———————————————————————————————— (01)
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS SCREEN.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  PARAMS.
    10  VAR1      PIC S9(8) COMP.
    10  VAR2      PIC X.
PROCEDURE DIVISION.
CALLPAS.
    MOVE 777 TO VAR1.
    MOVE "A" TO VAR2.
    CALL "PASUPRO" USING VAR1, VAR2. ———————————————————— (02)
```

```
ENDE.
    DISPLAY VAR1 UPON SCREEN.                                    (03)
    DISPLAY VAR2 UPON SCREEN.
    STOP RUN.
```

Quellcode der Pascal-XT-Entry-Prozedur "PASUPRO" im Paket "PAS":

Paketspezifikation:

```
PACKAGE PAS;
ENTRY PROCEDURE PASUPRO (VAR F1 : LONG_INTEGER; VAR F2 : CHAR); ———— (04)
END.
```

Paketimplementierung:

```
PACKAGE BODY PAS (OUTPUT);
PROCEDURE PASUPRO (VAR F1 : LONG_INTEGER; VAR F2 : CHAR);
BEGIN
   WRITELN ('F1: ',F1); ———————————————————————————————— (05)
   WRITELN ('F2: ',F2);
   F1 := 111; ———————————————————————————————————————— (06)
   F2 := 'Z';
END;
BEGIN
END.
```

Aufruf der ablauffähigen Phase "COBOLPAS" und Ablaufprotokoll:

```
/EXEC COBOLPAS ———————————————————————————————————————— (07)
%  BLS0500 PROGRAM 'COBOLPAS', VERSION ' ' OF '...' LOADED
F1:          777 ———————————————————————————————————— (08)
F2: A
00000111 ———————————————————————————————————————————— (09)
Z
```

Erläuterung:

(01) Name des Cobol-Hauptprogramms: COBPAS

(02) Aufruf der Pascal-XT-Entry-Prozedur PASUPRO, wobei die Variablen VAR1 mit dem Wert 777 und VAR2 mit dem Wert 'A' übergeben werden.

(03) Nach der Rückkehr aus der Pascal-XT-Entry-Prozedur PASUPRO werden die Werte der Variablen VAR1 und VAR2 ausgegeben. Siehe (09).

(04) Deklaration der Pascal-XT-Entry-Prozedur PASUPRO in der Spezifikation des Pakets PAS.

(05) Name und Wert der übergebenen Variablen F1 und F2 werden ausgegeben. Siehe (08).

(06) Der Wert der übergebenen Variablen wird global verändert.

(07) Starten der ablauffähigen Phase COBOLPAS.

(08) Ausgabe aus der Pascal-XT-Entry-Prozedur.

(09) Ausgabe aus dem Cobol-Hauptprogramm nach der Rückkehr aus der Pascal-XT-Entry-Prozedur.

Die Intern-Schnittstelle

Assembler-Unterprogramme können über die Intern-Schnittstelle aufgerufen werden, indem in der Prozedur- bzw. Funktions-Deklaration die Direktive Internal angegeben wird, z.B.:

```
PROCEDURE Prozedur-Bezeichner (Formalparameterliste); Internal;

oder

FUNCTION Funktions-Bezeichner (Formalparameterliste): Typ; Internal;
```

Die über die Intern-Schnittstelle aufgerufenen Unterprogramme sind nur zur Realisierung solcher Funktionen gedacht, die in Pascal-XT nicht formuliert werden können. Dazu gehören z.B. die Ausnutzung von speziellen Maschinenbefehlen oder der Aufruf von Betriebssystemfunktionen (SVCs). Diese Schnittstelle ist nicht zum Aufruf größerer Unterprogramme gedacht, da der Aufruf nicht wie ein Prozeduraufruf zu verstehen ist, sondern wie das Einfügen einiger Maschinenbefehle an der Stelle des Aufrufs. Dadurch kann der Aufruf sehr effizient durchgeführt werden und bietet gegenüber der Standard-Unterprogramm-Schnittstelle (Direktiven External usw., siehe 7.1) erhebliche Laufzeitvorteile.

Einschränkungen bei Parametertypen

Für die Formalparameter sind alle Typen zugelassen bis auf Konformreihungsschemata, Dateien, strukturierte Typen mit Komponenten von einem Datei-Typ, Parameterprozeduren und Parameterfunktionen.

Register-Konventionen

R1 bis R4	Diese Register sind für Parameterübergabe und die Rückgabe eines Funktionsergebnisses vorgesehen (siehe Parameterübergabe und Funktionen).
F0, F2	Diese beiden Gleitpunktregister werden zur Übergabe von Real-Werten bzw. zur Rückgabe eines Funktionsergebnisses von einem Real-Typ verwendet.
R9	Das Register kann als Arbeitsregister verwendet werden. Vor dem Rücksprung muß es allerdings wieder denselben Wert wie beim Aufruf enthalten. Es wird aber dringend empfohlen, dieses Register nicht zu verändern, da dies im Falle einer Exception zu einem Fehler führen kann.
R10, R11	Diese beiden Register dürfen auf keinen Fall verändert werden, auch nicht temporär (siehe unten).
R14	Rücksprungadresse.
R15	Adresse des aufgerufenen Unterprogramms.

Die anderen Register können als Arbeitsregister verwendet werden. Beim Rücksprung müssen lediglich die Register R9, R10 und R11 ihren ursprünglichen Wert enthalten.

Die Register R10 und R11 dürfen nicht geändert werden, da sonst auftretende Ausnahmen nicht richtig behandelt werden können und ein korrekter Rücksprung zum Rufer nicht mehr möglich ist.

Parameterübergabe

Alle Parameter werden in Registern übergeben. Für die Übergabe werden die Register R1 bis R4 und die Gleitpunktregister F0 und F2 verwendet. Die Anzahl der Parameter, die übergeben werden können, wird von der Art des Unterprogramms und den Typen der Formalparameter beeinflußt. Bei Funktionen wird implizit ein weiterer Parameter zur Übergabe des Funktionsergebnisses benötigt (siehe Funktionen). Bei der Wertparameterübergabe eines Strings (Zeichenketten variabler Länge) werden immer zwei Parameter übergeben (siehe Wertübergabe).

Die Zuordnung der Parameter zu den Registern erfolgt nach folgendem Schema:

- Die Formalparameter werden in der Reihenfolge ihrer Aufschreibung an die Register zugewiesen.
- Die Vergabe der Register beginnt bei R1 bzw. F0 für Real-Werte.
- Bei einer Funktion als Unterprogramm wird für das Funktionsergebnis ein Register benötigt. Ist das Funktionsergebnis von einem Real-Typ, dann wird in F0 das Funktionsergebnis zurückgeliefert und es kann nur noch ein Parameter von einem Real-Typ im Register F2 übergeben werden. In allen anderen Fällen wird das Funktionsergebnis im Register R1 übergeben und es stehen nur noch die Register R2 bis R4 für die Parameterübergabe zur Verfügung.
- Bei VAR-Parameter-Übergabe wird das nächste freie Register Ri (i = 1..4) verwendet (siehe unten).
- Bei Wertübergabe eines Real-Wertes wird das nächste freie Gleitpunktregister verwendet, ansonsten das nächste freie Register Ri. Für Strings variabler Länge werden die Register Ri und Ri+1 belegt (siehe unten).

Reichen die Register zur Übergabe der Parameter nicht aus, dann wird bereits zur Übersetzungszeit eine Fehlermeldung ausgegeben. Sollen mehr Parameter übergeben werden, dann müssen sie in einem Record (möglichst mit Angaben zur Speicherdarstellung) zusammengefasst werden.

Übergabe bei Variablenparameter

Für Variablenparameter wird immer die Adresse des Aktualparameters in einem allgemeinen Register Ri übergeben. Die Adresse kann, in Abhängigkeit der Ausrichtungsanforderung des Parametertyps eine Byte-, Halbwort-, Wort-, oder Doppelwort-Adresse sein (siehe Abschnitt 4.3).

Für Strings variabler Länge wird die Adresse des 2 Byte langen Längenfeldes übergeben. Unmittelbar auf das Längenfeld folgen die Zeichenwerte.

Übergabe bei Wertparameter

Bei Werteparametern wird, abhängig vom Typ des Formalparameters, der Wert oder die Adresse des Aktualparameters übergeben. Bei Übergabe der Adresse darf auf keinen Fall der Aktualparameter verändert werden, da sonst die Semantik des Programms verletzt wird.

(a) Bei Orginal-Typen (Integer-Typen, Char, Boolean, Aufzählungs- und Teilbereichstypen) wird der Wert übergeben. Die Werte stehen rechtsbündig im Register.

(b) Bei Real-Typen wird der Wert in einem Gleitpunktregister übergeben.

(c) Bei einem variablen Zeichenkettentyp werden immer zwei Parameter übergeben. Im ersten Parameter (Ri) wird die Adresse des ersten Zeichens der Zeichenkette und im zweiten Parameter (Ri+1) wird die aktuelle Länge der Zeichenkette übergeben.

(d) Bei allen anderen Typen wird die Adresse des Aktualparameters übergeben.

Darstellung der Objekte im Speicher

Zur Bearbeitung der übergebenen Parameter sind Kenntnisse über die Darstellung von Objekten im Speicher erforderlich. Dies ist im Abschnitt 4.3 beschrieben.

Funktionen

Funktionen können wie Prozeduren aufgefaßt werden, die einen zusätzlichen Parameter besitzen, in dem das Funktionsergebnis zurückgeliefert wird. Dieser Parameter wird immer als erster Parameter übergeben. Die Übergabe des Funktionsergebnisses ist abhängig vom Ergebnistyp:

Ordinal-Typen
: Das Funktionsergebnis muß im Register R1 abgelegt werden.

Real-Typen
: Das Funktionsergebnis muß im Gleitpunktregister F0 abgelegt werden.

Strukturierte Typen
: Der Rufer hat bereits den Bereich für das Funktionsergebnis allokiert und übergibt beim Aufruf im Register R1 die Adresse dieses Bereichs, in dem der Gerufene das Ergebnis ablegen muß.

Ausnahmebehandlung

Das Auftreten einer Ausnahme (eines Fehlers) während der Ausführung des gerufenen Unterprogramms wird so behandelt, als wäre sie im Rufer aufgetreten. Von dort aus wird der nächste zuständige Ausnahmebehandler gesucht. Damit die Ausnahmebehandlung korrekt funktioniert, dürfen im gerufenen Unterprogramm die Register R9, R10 und R11 keinesfalls verändert werden. Bei Änderung nur eines der Register führt dies zu einem Laufzeitfehler mit undefiniertem Verhalten.

Trennung von Code und Daten

Das über die Intern-Schnittstelle gerufene Unterprogramm unterscheidet sich von Pascal-Unterprogrammen durch Fehlen eines Arbeitsbereichs für lokale Daten. Der Arbeitsbereich wird u.a. benötigt, um Register zu retten oder eine Kopie eines strukturierten Werteparameters anzulegen. Es bieten sich mehrere Möglichkeiten zur Beschaffung des Bereichs an:

(a) Der Rufer übergibt einen weiteren Variablen-Parameter, z.B. ein Feld (Array), das der Gerufene als Arbeitsbereich verwenden kann. Damit ist das Unterprogramm mehrfach benutzbar.

(b) Im Unterprogramm wird statisch ein Datenbereich definiert. Damit kann das Unterprogramm nicht mehrfach benutzt werden.

(c) Das Unterprogramm beschafft sich dynamisch einen eigenen Datenbereich (z.B. mit REQM) und ist damit mehrfach benutzbar.

Einbinden des Unterprogramms

Die Externreferenzen auf externe Unterprogramme stehen im Daten-Modul des Pakets bzw. Hauptprogramms, in dem das Unterprogramm deklariert ist. Beim getrennten Binden von Code- und Daten-Modulen muß folgendes beachtet werden:

(a) Programm ist **mehrfach benutzbar**

Das Unterprogramm muß explizit zu den Code-Modulen eingebunden werden. Beim Binden der Daten-Module muß das Auflösen der Externreferenz auf das Unterprogramm explizit ausgeschlossen werden.
Im Unterprogramm darf kein lokaler Arbeitsbereich deklariert werden.

(b) Programm ist **nicht mehrfach benutzbar**

Das Unterprogramm wird zu den Daten-Modulen gebunden. Im Unterprogramm kann ein lokaler Arbeitsbereich definiert sein.

Beispiel 1

In diesem Beispiel wird die Funktion UPPER über die Intern-Schnittstelle aufgerufen, die in einem String alle Kleinbuchstaben durch Großbuchstaben ersetzt. Das Hauptprogramm übergibt an die Funktion einen String als Wertparameter und bekommt als Funktionsergebnis den transformierten String zurück.
Das Programm wird mit dem DLL geladen und gestartet. Zur Auflösung der Externreferenzen auf die Laufzeitsystemmodule muß die Tasklib auf die Bibliothek $PASLIB-XT eingestellt werden.

```
/EXEC $ASSEMB
%  BLS0500 PROGRAM 'ASSEMB', VERSION '...' OF '...' LOADED.
V30.0A20 OF SIEMENS BS 2000 ASSEMBLER READY

SIEMENS F-ASSEMBLER LISTING                                                    10:02:47  90-11-27   PAGE 0001
         SYMBOL   TYPE ID  ADDR  LENGTH                    EXTERNAL SYMBOL DICTIONARY

         LETTER    SD 0001 00000  00124
         UPPER     LD 0001 00000
SIEMENS F-ASSEMBLER LISTING                                                    10:02:47  90-11-27   PAGE 0002
FLAG LOCTN OBJECT CODE    ADDR1  ADDR2  STMNT M  SOURCE STATEMENT

    000000                                   1    LETTER   CSECT

    000000                                   2             ENTRY UPPER
                                             3    *
                                             4    *        Diese Funktion wird ueber die Intern-Schnittstelle aufgerufen.
                                             5    *        Es werden alle Kleinbuchstaben in Grossbuchstaben umgewandelt.
                                             6    *        Die Umwandlung erfolgt nur innerhalb der aktuellen Laenge des
                                             7    *        Strings. Der String-Typ ist fuer diese Funktion unwichtig.
                                             8    *
                                             9    *
                                            10    *        PROCEDURE upper ( s: String ): String;  Internal;
                                            11    *
                                            12    *        Parameter:
                                            13    *           R1:  Adresse des Funktionsergebnisses
                                            14    *           R2:  Adresse des 1. Zeichen des Strings
                                            15    *           R3:  Aktuelle Laenge des Strings
                                            16    *
                         000001             17    R1       EQU   1
                         000002             18    R2       EQU   2
                         000003             19    R3       EQU   3
                         00000E             20    RR       EQU   14
                         00000F             21    RX       EQU   15
                                            22    *
    000000                                  23             USING UPPER,RX
    000000 40 30 1000                       24    UPPER    STH   R3,0(0,R1)          Aktuelle Laenge kopieren
    000004 5B 30 F120     000120            25             S     R3,=F'1'            und um 1 erniedrigen
    000008 07 4E                            26             BRM   RR                  Ruecksprung bei negativer Laenge
    00000A 44 30 F014     000014            27             EX    R3,MOVE             String kopieren
    00000E 44 30 F01A     00001A            28             EX    R3,TRANS            String transformieren
    000012 07 FE                            29             BR    RR

    000014 D2 00 10022000                   30    MOVE     MVC   2(0,R1),0(R2)
    00001A DC 00 1002F020        000020     31    TRANS    TR    2(0,R1),TABLE
                                            32    *
                                            33    *

                                            34    *        Setzt 'a'..'z' in 'A'..'Z' um
                                            35    *
                                            36    *                 0 1 2 3 4 5 6 7 8 9 A B C D E F
    000020 0001020304050607                 37    TABLE    DC    X'000102030405060708090A0B0C0D0E0F'    00 - 0F
    000030 1011121314151617                 38             DC    X'101112131415161718191A1B1C1D1E1F'    10 - 1F
    000040 2021222324252627                 39             DC    X'202122232425262728292A2B2C2D2E2F'    20 - 2F
    000050 3031323334353637                 40             DC    X'303132333435363738393A3B3C3D3E3F'    30 - 3F
    000060 4041424344454647                 41             DC    X'404142434445464748494A4B4C4D4E4F'    40 - 4F
```

```
    000070 5051525354555657                42            DC    X'505152535455565758595A5B5C5D5E5F'    50 - 5F
    000080 6061626364656667                43            DC    X'606162636465666768696A6B6C6D6E6F'    60 - 6F
    000090 7071727374757677                44            DC    X'707172737475767778797A7B7C7D7E7F'    70 - 7F
    0000A0 80C1C2C3C4C5C6C7                45            DC    X'80C1C2C3C4C5C6C7C8C98A8B8C8D8E8F'    80 - 8F
    0000B0 90D1D2D3D4D5D6D7                46            DC    X'90D1D2D3D4D5D6D7D8D99A9B9C9D9E9F'    90 - 9F
    0000C0 A0A1E2E3E4E5E6E7                47            DC    X'A0A1E2E3E4E5E6E7E8E9AAABACADAEAF'    A0 - AF
    0000D0 B0B1B2B3B4B5B6B7                48            DC    X'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'    B0 - BF
    0000E0 C0C1C2C3C4C5C6C7                49            DC    X'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'    C0 - CF
    0000F0 D0D1D2D3D4D5D6D7                50            DC    X'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'    D0 - DF
SIEMENS F-ASSEMBLER LISTING                                                      10:02:47  90-11-27   PAGE 0003
FLAG LOCTN OBJECT CODE    ADDR1  ADDR2  STMNT M  SOURCE STATEMENT
    000100 E0E1E2E3E4E5E6E7                51            DC    X'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'    E0 - EF
    000110 F0F1F2F3F4F5F6F7                52            DC    X'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'    F0 - FF

    000120 00000001                        54                  =F'1'
                                           55            END
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : -
THIS PROGRAM WAS ASSEMBLED BY THE SIEMENS ASSEMBLER (F) V30.0A20   CORR LEVEL: (2:0000010110:0000000000)
SOURCE LIBRARY  :      :V:$USERID.PLAM.EXAMPLE
SOURCE PROGRAM  :      UPPER.ASS
SOURCE VERS/DATE:        @/901127
MODULE  LIBRARY   :    :V:$USERID.PLAM.EXAMPLE
LIBRARY  ELEMENT  :    LETTER    VER-
ASSEMBLY TIME :        0.4235  SEC.
```

```
/EXEC $USERID.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//C (PLAM.EXAMPLE, TRANSFORM.PROG), *SYSOUT, *STD
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG       =    OFF          BY DEFAULT
GENERATE    =    ON           BY DEFAULT
MAP         =    OFF          BY DEFAULT
STANDARD    =    OFF          BY DEFAULT
XREF        =    OFF          BY DEFAULT

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,TRANSFORM.PROG(*UPPER-LIMIT,S))

     1      program TRANSFORM (input, output);
     2
     3      var
     4            s : string;
     5
     6         function upper (s: string): string; internal;
     7
     8
     9      begin
    10         writeln (output, 'Bitte einen Text eingeben');
    11         readln  (input);
    12         read    (input, s);
    13         writeln (output, upper (s));
    14      end.
```

```
************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :         0                     *
*  WARNINGS                 :         0                     *
*  NOTES                    :         0                     *
*  SIZE OF CODE MODULE      :       600 BYTES               *
*  SIZE OF DATA MODULE      :       356 BYTES               *
*  COMPILATION TIME         :     0.379 SEC                 *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//END
   END OF THE PASCAL SESSION  -  USED TIME = 0.450 SECONDS
```

Ausführen des Programms

```
/SYSFILE TASKLIB=$USERID.PASLIB-XT
/EXEC (TRANSFO, PLAM.EXAMPLE)
%  BLS0001 *** DBL VERSION 067 RUNNING ***
%  BLS0517 MODULE 'TRANSFO' LOADED
Bitte einen Text eingeben
123aAbBccCdD$=Q
123AABBCCCDD$=Q
```

Beispiel 2

In diesem Beispiel wird über die Intern-Schnittstelle das Unterprogramm LINK aufgerufen, das ein Programm dynamisch nachladen soll. Das Hauptprogramm LOADER bereitet die Beschreibung für das Link-Makro (siehe DLINK-Makro [6]) und übergibt sie als Parameter. Im Parameter USERBLK liefert das Unterprogramm LINK die Liste der vom DLL geladenen Module zurück.
Zur Demonstration wird das Programm TRANSFORM aus Beispiel 1 aus der Bibliothek PLAM.EXAMPLE geladen und die Liste der geladenen Module ausgegeben.

```
/EXEC $USERID.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//C (PLAM.EXAMPLE, LINK.PROG), *SYSOUT, *STD
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT

CURRENT COMPILATION UNIT (SOURCE FILE)

             ($USERID.PLAM.EXAMPLE,LINK.PROG(*UPPER-LIMIT,S))

     1      program LOADER (input, output);
     2
     3
     4      {***     Allgemeine Definitionen    ***}
     5
     6      type
     7         symbol    = packed array [1..8]  of char;
     8         filename  = packed array [1..54] of char;
     9
    10
    11      {***     Aufbau der Tabelle fuer die Externverweisliste    ***}
    12
    13      type
    14         maxsym  = 1 .. 100;
    15         symtab  = record
    16                     len   : short_integer;            { Satzlaengenfeld }
    17                     res1  : short_integer;            { reserviert      }
    18                     sym   : array [maxsym] of symbol;
    19                   end;
    20
    21      const
    22         end_of_table = symbol (#'00':8);
    23
    24      {***     Operandenliste fuer das LINK-Makro    ***}
    25
    26      const
```

```
27          krzinhbt = #01;
28          krztsklb = #02;
29          krzlibnm = #04;
30          krzuserb = #40;
31
32          defaults = krzinhbt + krztsklb + krzlibnm + krzuserb;
33
34       type
35          byte    = 0 .. 255;
36          linkop  = record
37                       krzcontr  : byte;
38                       krzentry  : symbol;
39                       krzlibry  : filename;
40                       krzform   : byte;
41                       case boolean of
42                         true:  (krzpage : short_integer);
43                         false: (krzmpid : long_integer);
44                       end;
45
46
47       var
48          userblk   : symtab;
49          linkdescr : linkop;
50          result    : integer;
51
52       {***        Return-Codes          ***}
53
54       const
55          error  =  #FFFFFFFF;   { ungueltige Operanden oder Adresse    }
56          ok     =  0;           { Modul geladen, Adresse in Register 1 }
57
58
59       procedure link (     descr   : linkop;
60                            tab     : symtab;
61                         var retcode : integer ); internal;
62
63
64       function loaded: boolean;
65        begin
66           if result = error then begin
67              loaded  := false;
68              writeln (output, 'Operandenfehler oder ungueltige Adresse');
69            end
70           else
71              loaded := (result mod 256) = ok;
72        end { loaded };
73
74
75       procedure print_list;
76        var
77             i : maxsym;
78        begin
79           writeln (output, 'Liste der vom DLL geladenen Module:');
80           for i := first (maxsym) to last (maxsym) do
81              if userblk.sym [i] <> end_of_table then
82                 writeln (output, '   ', userblk.sym [i]);
83        end { print_list };
84
```

```
                85
                86
                87      begin  { LOADER }
                88         userblk.len := sizeof (symtab);
                89         with linkdescr do begin
                90            krzcontr := defaults;
                91            krzentry := 'TRANSFO ';
                92            krzlibry := 'PLAM.EXAMPLE                                    ';
                93            end;
                94         link (linkdescr, userblk, result);
                95         if loaded then
                96            print_list;
                97      end.


            ************************************************************
            *                    COMPILATION SUMMARY                   *
            ************************************************************
            *  ERRORS DETECTED          :         0                     *
            *  WARNINGS                 :         0                     *
            *  NOTES                    :         0                     *
            *  SIZE OF CODE MODULE      :      1156 BYTES               *
            *  SIZE OF DATA MODULE      :       980 BYTES               *
            *  COMPILATION TIME         :     0.532 SEC                 *
            ************************************************************
               >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
            //END
               END OF THE PASCAL SESSION  -  USED TIME = 0.602 SECONDS


/EXEC $ASSEMB
%  BLS0500 PROGRAM 'ASSEMB', VERSION '300' OF '89-11-03' LOADED.
V30.0A20 OF SIEMENS BS 2000 ASSEMBLER READY

SIEMENS F-ASSEMBLER LISTING                                                    10:02:16  90-11-27   PAGE 0001
         SYMBOL   TYPE ID  ADDR  LENGTH                    EXTERNAL SYMBOL DICTIONARY

         LNKCALL   SD 0001 00000  00008
         LINK      LD 0001 00000
SIEMENS F-ASSEMBLER LISTING                                                    10:02:16  90-11-27   PAGE 0002
FLAG LOCTN OBJECT CODE    ADDR1  ADDR2  STMNT M  SOURCE STATEMENT

    000000                                1    LNKCALL  CSECT

    000000                                2             ENTRY LINK
                                          3    *
                                          4    *        Dieses Unterprogramm wird ueber die Intern-Schnittstelle
                                          5    *        aufgerufen. Es soll das Link-Makro aufrufen.
                                          6    *        Das Link-Macro benoetigt die Register R0, R1 und R15.
                                          7    *
                                          8    *        PROCEDURE LINK (     operands : { ein Record-Typ };
                                          9    *                             userblk  : { ein Record-Typ };
                                         10    *                          var result   : Integer );
                                         11    *        Parameter:
                                         12    *           R1:  Adresse der Operandenliste
                                         13    *           R2:  Adresse des userblocks
                                         14    *           R3:  Adresse des Ergebnisses
                                         15    *
                      000000             16    R0       EQU   0
                      000001             17    PAR1     EQU   1                1. Parameter
                      000002             18    PAR2     EQU   2                2. Parameter
                      000003             19    PAR3     EQU   3                3. Parameter
                      00000E             20    RR       EQU   14
                      00000F             21    RX       EQU   15
                                         22    *
```

```
    000000 18 02                          23    LINK     LR    R0,PAR2          USERBLK Adresse laden
                                          24             LINK  MF=(E,(1))       Aufruf des Link-Macros
                                          25  1          #INTF REFTYPE=REQUEST,INTNAME=LINK,INTCOMP=1         00019000
    000002 0A 6E                          26  1          SVC   110                                            00064000
    000004 18 3F                          27             LR    PAR3,RX          Ergebnis in den 3. Parameter
    000006 07 FE                          28             BR    RR

                                          29             END
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : -
THIS PROGRAM WAS ASSEMBLED BY THE SIEMENS ASSEMBLER (F) V30.0A20   CORR LEVEL: (2:0000010110:0000000000)
SYSTEM MACROLIBRARY :  :D:$TSOS.MACROLIB
SOURCE LIBRARY  :      :V:$USERID.PLAM.EXAMPLE
SOURCE PROGRAM  :      LINK.ASS
SOURCE VERS/DATE:        @/901127
MODULE  LIBRARY   :    :V:$USERID.PLAM.EXAMPLE
LIBRARY  ELEMENT  :    LNKCALL   VER-
ASSEMBLY TIME :        0.8187  SEC.
```

Ausführen des Programms

```
/SYSFILE TASKLIB=$USERID.PASLIB-XT
/EXEC (LOADER, PLAM.EXAMPLE)
%  BLS0001 *** DBL VERSION 067 RUNNING ***
%  BLS0517 MODULE 'LOADER' LOADED
Liste der vom DLL geladenen Module:
   TRANSFO
   TRANSFOC
   TRANSFOD
   LETTER
```

UTM-Anschluß

Ab UTM V3.0 können UTM-Teilprogramme auch in Pascal-XT geschrieben werden. An dieser Stelle werden nur kurz die Besonderheiten und Randbedingungen zusammengefaßt, die bei der Implementierung von UTM-Teilprogrammen in Pascal-XT zu beachten sind. Für eine ausführliche Darstellung sei auf die Manuale

- UTM Planen und entwerfen [9]
- UTM Anwendungen generieren und administrieren [10]
- UTM Anwendungen programmieren in Pascal-XT [11]

verwiesen.

Für die Generierungsprozedur KDCROOT gilt: In der Programmieranweisung muß COMP wie folgt gesetzt werden:
Bis Pascal-XT V2.1 COMP = PASCAL
Ab Pascal-XT V2.2 COMP = ILCS

UTM-Teilprogramme werden vom UTM-Anschlußprogramm als Unterprogramme aufgerufen. Die Kommunikation des gerufenen Teilprogramms mit den übrigen Komponenten der UTM-Anwendung geschieht ausschließlich durch Aufrufe der (im Anschlußprogramm KDCROOT definierten) externen Prozedur KDCS (Kompatible Data-Communication Schnittstelle).

Pascal-XT V2.2A benutzt als Schnittstelle zu UTM ausschließlich ILCS (Inter-Language Communication Services, siehe 7.1), ältere Pascal-XT-Versionen benutzen IUTMHLL. Pascal-XT V2.2A setzt deshalb UTM V3.2 voraus. Ältere Pascal-XT-Programme können aber auch unter UTM V3.2 ablaufen, da diese Version die Schnittstellen IUTMHLL und ILCS unterstützt:

	UTM V3.1	UTM V3.2
Bis Pascal-XT V2.1	IUTMHLL	IUTMHLL
Ab Pascal-XT V2.2	——	ILCS

Tabelle 8-1: Mögliche Kopplungen zwischen Pascal-XT und UTM

Das UTM-Anschlußprogramm KDCROOT ruft das Pascal-XT-Teilprogramm nach ILCS-Konvention auf. In Pascal-XT geschriebene UTM-Teilprogramme müssen daher den im Kapitel 7 (Sprachverknüpfungen) genannten Bedingungen genügen:

- UTM-Teilprogramme müssen in einer Paket-Spezifikation als ENTRY-Prozeduren deklariert werden.
- Die externe Prozedur KDCS muß mit der Direktive COBOL deklariert werden.

Sprachverknüpfungen unter UTM

Ab Pascal-XT V2.2A ist es erlaubt, aus einem in Pascal-XT geschriebenen UTM-Teilprogramm fremdsprachige Unterprogramme aufzurufen, die selbst keine UTM-Teilprogramme sind. Wenn die fremdsprachigen Unterprogramme ILCS benützen, können sie auch an der Fehlerbehandlung von ILCS teilnehmen.

Umgekehrt ist es auch erlaubt, aus fremdsprachigen UTM-Teilprogrammen, die ILCS verwenden, Pascal-XT-ENTRY-Routinen aufzurufen. In diesem Fall sind die Pascal-XT-ENTRY-Routinen keine eigenen UTM-Teilprogramme.

Datentypen und Konstanten

Die beim Aufruf der externen Prozedur KDCS in den Parametern verwendeten Datenstrukturen und Konstanten werden dem Benutzer in Form von vordefinierten Paketen zur Verfügung gestellt:

- KCKBL enthält
  - die Definition der KDCS-Operationscodes
  - die Typdeklaration für den Kopf des Kommunikationsbereichs
  - die Typdeklaration des Parameterfeldes

- KCINL enthält
  - die Typdeklaration für die Rückgabeinformation des INFO-Aufrufs
- KCDFL enthält
  - die Konstantendefinitionen der Bildschirmausgabefunktionen
- FIELD_ATTRIBUTE_PACKAGE enthält
  - die Typ- und Konstantendeklarationen für die Feldattribute bei Betrieb der Datenstation(en) im Formatmodus
- KCMSL enthält
  - die Konstanten- und Typdeklarationen der UTM-Meldungen.

Hinweis

Obwohl die Implementierungen der genannten Pakete sämtlich "leer" sind, müssen sie übersetzt und beim Binden der Teilprogramme mit eingebunden werden, da sie bei der Initialisierung der Pascal-XT-Umgebung angesprochen werden. (Da der Pascal-XT Compiler bei der Kompilierung einer Übersetzungseinheit nur die Spezifikationen der referenzierten Pakete liest, kann er nicht "wissen", ob eine Paketimplementierung leer ist).

Grundstruktur von UTM-Teilprogrammen

Anzahl und Struktur der Prozedurparameter sind durch UTM weitgehend festgelegt (s. Manual "Anwendungen programmieren in Pascal-XT" [11]). Die Paket-Spezifikationen für UTM-Teilprogramme müssen daher folgendem Muster entsprechen:

```
with KCKBL;
from KCKBL use kckb, kcpal;
...
package UTM_TP;
...
type
   ...
   t_progb = ...;
   ...

   t_kckb  = record          (* Kommunikationsbereich *)
      kbkopf: kckb;          (* ..- Kopf *)
      progb : t_progb;       (* Programmbereich *)
      end;
   ...

   t_nb = ...;
   ...

   t_spab = record           (* Standard Primaerer Arbeitsbereich *)
      parm: kcpal;           (* Parameterfeld *)
      nb  : t_nb;            (* Nachrichtenbereich *)
```

```
      end;
   ...

   procedure KDCS (var p: kcpal; var n: t_nb); cobol;

   (* Hinweis: Wenn der Nachrichtenbereich als Record-Typ
               deklariert ist, kann für den zweiten Parameter der
               Prozedur KDCS auch ein Komponententyp des Nachrichten-
               bereiches verwendet werden *)

   ...

   entry procedure teilprogramm_1 (var kb: t_kckb; var spab: t_spab);
   ...
   entry procedure teilprogramm_n (var kb: t_kckb; var spab: t_spab);

   (* Hinweis: Teilprogramme, die das gleiche Parameterlayout
               verwenden, können in einem Paket zusammengefaßt werden *)
end (* UTM_TP *).
```

Bei der Paket-Implementierung von UTM-Teilprogrammen müssen folgende Besonderheiten beachtet werden:

- Vor dem ersten KDCS-Aufruf (mit dem Operationscode INIT) können - außer bei den Benutzerausgängen START, SHUT und VORGANG, in denen keine KDCS-Aufrufe erlaubt sind - die aktuellen Parameter der Entry-Prozedur nicht sinnvoll verwendet werden.
- die Kommunikation mit dem UTM-Anschlußprogramm muß - außer bei den Benutzerausgängen START, SHUT und VORGANG - durch einen KDCS-Aufruf mit dem Operationscode PEND beendet werden.
  UTM kehrt danach nicht wieder in die Entry-Prozedur zurück!

  Daraus ergeben sich folgende Konsequenzen:

  – Die Entry-Prozedur darf nicht normal beendet werden, sonst tritt ein PEND-Error auf.

  – Das **return**-Statement darf im äußersten Block einer solchen Entry-Prozedur nicht verwendet werden, sonst tritt ein PEND-Error auf.

  – Lokale Dateien, die im äußersten Block einer solchen Entry-Prozedur deklariert sind, werden nach dem PEND-Aufruf nicht geschlossen und dürfen daher nur in inneren Blöcken verwendet werden.

Speicheranforderung mittels "NEW"

Ab Pascal-XT V2.2A können in Pascal-XT geschriebene UTM-Teilprogramme mittels "New" (siehe [1], 15.2) dynamisch Speicher anfordern. Der Anwender ist selbst dafür zuständig, mittels "dispose", "release" oder "Heapsupport.Release_Heap" den Speicher vor dem KDCS-Aufruf PEND wieder freizugeben. Das Pascal-XT-Laufzeitsystem gibt den

dynamisch angeforderten Speicher weder am Ende des UTM-Teilprogramms noch am Ende der UTM-Gesamtanwendung frei. Auch wenn der mit "New" angeforderte Speicher nicht freigegeben wird, kann ein Folgeprogramm nicht darauf zugreifen, weil dieses bei UTM unter einer anderen Task laufen kann. Für diese Task steht der zuvor mit "New" angeforderte Speicher nicht zur Verfügung.

Externe Dateien

Bei der Verwendung externer Dateien in UTM-Teilprogrammen muß berücksichtigt werden, daß sie nicht (wie bei einem Pascal-XT-Hauptprogramm) bei Programmende geschlossen werden. Der Anwender muß also selbst für das Schließen der eröffneten Dateien sorgen. Dies geschieht zweckmäßig in einer Entry-Prozedur für den Benutzerausgang SHUT. Lokale Dateien von Unterprogrammen (Prozeduren und Funktionen) werden vom Pascal-System geschlossen.

Hinweis

Werden die Teilprogramme einer UTM-Anwendung in mehreren verschiedenen Paketen implementiert, und wird die gleiche externe Datei in mehreren Paketen verwendet, so darf sie nur in einem Paket in der Programmparameterliste als Dateivariable deklariert werden. In den anderen Paketen muß sie durch eine Kontext-Spezifikation (WITH-/USE-Liste) importiert werden.

Fehlerbehandlung ab Pascal-XT V2.2A

Beim Aufruf eines in Pascal-XT geschriebenen UTM-Teilprogramms meldet sich das Pascal-XT-Laufzeitsystem bei der Fehlerbehandlung von ILCS an. Tritt während des Ablaufs des Pascal-XT-Teilprogramms ein Fehler auf oder wird von einem fremdsprachigen Unterprogramm ein Fehler propagiert, sucht das Laufzeitsystem in dem Teilprogramm nach einem Ausnahmebehandler. Wenn ein Ausnahmebehandler definiert ist, wird das Pascal-XT-Teilprogramm dort fortgesetzt.
Wenn kein Ausnahmebehandler definiert ist, propagiert das Pascal-XT-Laufzeitsystem den Fehler an den Rufer (UTM). UTM gibt einen User-Dump aus und beendet den Vorgang mit einem KDCS-Aufruf PEND /Error.

Fehlerbehandlung bis Pascal-XT V2.2A

Im UTM-Anschlußprogramm KDCROOT ist eine eigene Fehlerbehandlung implementiert, die im Fehlerfalle (STXIT-Ereignisse) dafür sorgt, daß nicht abgeschlossene UTM- und ggf. Datenbanktransaktionen zurückgesetzt (nur UTM-S) werden und die Anwendung ggf. definiert beendet wird.

Enthält eine UTM-Anwendung wenigstens ein Pascal-XT-Teilprogramm, wird ein Pascal-XT-spezifisches Verbindungsmodul mit eingebunden, das folgende Aktionen durchführt:

- Beim dynamisch ersten Aufruf eines Pascal-XT-Teilprogramms wird dem Pascal-XT Laufzeitsystem mitgeteilt, daß die STXIT-Behandlung durch UTM vorgenommen wird; die STXIT-Behandlung des Pascal-XT Laufzeitsystems wird nicht aktiviert.
- Tritt ein STXIT-Ereignis während der Aktivphase eines Pascal-XT Teilprogramms ein (im Pascal-XT-Code oder während der Ausführung eines KDCS-Aufrufs), wird die Fehlerbehandlung des Pascal-XT Laufzeitsystems aufgerufen (bei den Ereignissen PROCHK, ERROR und TIMER). Wird der Fehler im Pascal-XT Teilprogramm nicht durch einen Ausnahme-Behandler behandelt, dann wird die UTM-Fehlerbehandlung nach Ausgabe der Fehlerursache und der dynamischen Aufrufkette durch das Pascal-XT Laufzeitsystem fortgesetzt.
- Das Pascal-XT Laufzeitsystem meldet an UTM die erste Zeile der dynamischen Aufrufkette, wenn während der Ausführung eines Pascal-XT Teilprogramms ein Laufzeitfehler auftritt, der nicht durch einen Ausnahme-Behandler behandelt wird. UTM gibt dann diese Zeile zusammen mit dem UTM-Dump aus.

Hinweis

Durch dieses Verhalten hat der Anwender die Möglichkeit, von UTM im Rückgabefeld des Kommunikationsbereichs gemeldete Fehlersituationen (durch die Standardprozedur RAISE) selbst zu behandeln.

Binden der Anwendung

Für das Binden einer UTM-Anwendung sind keine besonderen Vorkehrungen erforderlich. Es gelten die Aussagen von Kapitel 6 "Binden und Ausführen von Objektprogrammen".

Das vordefinierte Paket UTM_ADAPTER wird ab Pascal-XT V2.2A zwar nicht mehr benötigt, aber weiterhin angeboten, damit bestehende in Pascal-XT geschriebene UTM-Teilprogramme nicht rekompiliert werden müssen.

Die Testhilfe PATH

Die Testhilfe PATH ist für Pascal-XT spezifisch und Bestandteil aller Pascal-XT-Implementierungen. Die Benutzeroberfläche ist, bedingt durch die verschiedenen Betriebssysteme, bis auf wenige Unterschiede in allen Implementierungen gleich.

Die Kapitel 9.1 bis 9.3 beschreiben allgemeingültige, also nicht BS2000-spezifische Eigenschaften der Testhilfe PATH. Die Kapitel 9.4 und 9.5 beschreiben die Handhabung von PATH im BS2000.

Elemente und Eigenschaften von PATH

PATH ist eine interaktive Pascal-Testhilfe, die das symbolische Testen von Pascal-XT-Programmen im Dialog oder in vorbereiteten Testabläufen ermöglicht. Um eine Pascal-XT-Übersetzungseinheit mit PATH testen zu können, muß bei der Übersetzung mit dem Pascal-XT-Compiler die Option DEBUG=ON oder DEBUG=RESTRICTED zur Erzeugung einer Testtabelle gesetzt sein.

Es wird empfohlen, nicht mehrere Pascal-XT-Anweisungen in eine Zeile des Quellprogramms zu schreiben (siehe 9.1.4.2).

Die wichtigsten Funktionen von PATH sind:

- Anhalten von Pascal-Programmen
- Ausgabe und Änderung von Programmdaten
- Bedingte Ausführung von Kommandos
- Ausgabe der dynamischen Aufrufkette (Aufruf-Vergangenheit)

Die Programmdaten werden symbolisch (d.h. mit den im Quellprogramm definierten Namen) und entsprechend ihrem Typ angesprochen.

Insbesondere bildet PATH die Typen von Pascal-XT bei der Ausgabe nach (z.B. VAR FARBE: (ROT, GELB, GRUEN); FARBE:=ROT; wird als FARBE=ROT ausgegeben).

PATH berücksichtigt außerdem die blockorientierte Struktur von Pascal und bildet den durch die Prozedurschachtelung definierten Gültigkeitsbereich (Scope) der Bezeichner zur Testzeit nach. Ebenso ermöglicht PATH auch ein Testen von rekursiven Prozeduren bzw. Funktionen eines Programms.

Bei Sprachverknüpfungen können auch Pascal-XT Pakete, die von Nicht-Pascal-XT-Programmen über die ENTRY-Schnittstelle aufgerufen werden, mit PATH getestet werden.

Zur Verwendung von PATH ist keinerlei Kenntnis des vom Pascal-XT-Compiler erzeugten Objektcodes erforderlich.

Solange keine Testhilfe-Kommandos abgesetzt werden, hat PATH keinen Einfluß auf die Laufzeit des Programms (abgesehen von den Initialisierungen für PATH).

PATH läuft interaktiv ab, hat eine Pascal-ähnliche Syntax, ist formatfrei und rekursiv und ermöglicht strukturierte Testhilfe-Kommandos (Verbund-Kommando). Für die gebräuchlichsten Kommandos gibt es Abkürzungen (in der Formatbeschreibung hervorgehoben). Die Kommando-Namen und Abkürzungen können, sofern sie keine Pascal-Schlüsselwörter sind, auch als Programm-Bezeichner angesprochen werden.

Die für den Compiler geltenden Konventionen bezüglich Groß- und Kleinschreibung, Ersatzsymbole, sedezimale Angabe von INTEGER, CHAR und STRING, Bezeichner mit Unterstreichungszeichen gelten sinngemäß auch für die Testhilfe. Ebenso wie der Pascal-XT-Compiler erkennt auch PATH Syntax- und Semantikfehler. Wird ein fehlerhaftes Kommando eingegeben, so wird dieses bis zu jener Stelle ausgegeben, an welcher der Fehler erkannt wurde. Der Rest der Eingabezeile wird nicht mehr analysiert und in der Ausgabe durch drei Punkte ersetzt. Das fehlerhafte Kommando und eventuelle Kommandos, in denen es enthalten ist, sind neu einzugeben.

Treten Fehler bei der Ausführung von Testhilfe-Kommandos auf, so wird dies gemeldet und das Kommando nicht bzw. unvollständig ausgeführt.

Alle Testhilfe-Ausgaben werden zur einfacheren Unterscheidung von Programm-Ausgaben mit einem Präfix versehen. In der BS2000 Version von PATH und in den Beispielen dieser Anwenderbeschreibung wird "%%" als Präfix verwendet.

Kommandoübersicht

Man unterscheidet bei PATH Testpunkt-Kommandos und Aktions-Kommandos. Sie sind jeweils mit einem Semikolon abzuschließen. Ihre Syntaxformate und ihre genaue Bedeutung sind unter 9.2 beschrieben.

TESTPUNKT-KOMMANDOS

AT	Setzen von Testpunkten und Ablegen eines Testhilfe-Kommandos, welches jedesmal beim Erreichen eines dieser Testpunkte ausgeführt wird. AT ist nur zusammen mit einem anderen PATH-Kommando sinnvoll.
REMOVE	Löschen der angegebenen (mit AT gesetzten) Testpunkte.
SLEEP	Deaktivieren der für die angegebenen Testpunkte (mit AT) gesetzten Kommandos.
AWAKE	Aktivieren der (mit SLEEP) deaktivierten Kommandos der angegebenen Testpunkte.
GETCMD	Unterbrechen des Testlaufs und Eingabeaufforderung für weitere Kommandos.
RESUME	Fortsetzung an der durch GETCMD unterbrochenen Stelle.

AKTIONS-KOMMANDOS

DISPLAY	Ausgabe von Daten auf das PATH-Ausgabe-Medium.
ASSIGN	Zuweisung von Werten an Variable des Programms.
IF	Bedingte Ausführung von Testhilfe-Kommandos.
BEGIN ... END	Zusammenfassung der zwischen BEGIN und END stehenden Testhilfe-Kommandos zu einem (Verbund-)Kommando.
SYSTEM	Übergang in den Systemmodus oder Ausführen des in Hochkommas eingeschlossenen System-Kommandos.
EDIT	Aufruf eines Editors.
SHOW	Ausgabe von Zustandsinformationen (SHOW UNITS, SHOW CALLS, SHOW WHERE).
DUMP	Ausgabe aller Variablen der dynamischen Kette (der Variablen am Stack) und der globalen Variablen aller Pakete auf das PATH-Ausgabe-Medium.
KILL	Abbruch des Testlings-Programmlaufs.
SWITCH	Zuordnung von PATH-Ein- und -Ausgabe-Dateien.

Begriffsdefinitionen

Die hier erläuterten Begriffe werden im folgenden häufig verwendet.

- Testling

  das zu testende Pascal-XT-Programm einschließlich aller dazugehörigen Pakete.

- Übersetzungseinheit

  Hauptprogramm oder ein Paket (= Summe aus Spezifikation und Implementierung).

- aktuelle Übersetzungseinheit

  Übersetzungseinheit, in der sich der aktuelle Testpunkt (siehe 9.1.4) befindet.

- globale Sichtbarkeit

  bedeutet, daß nur die Namen aller Übersetzungseinheiten, sowie vordefinierte Namen (z.B.: Maxint) sichtbar sind. In Übersetzungseinheiten deklarierte Namen können in diesem Fall nur durch Voranstellen einer Block-Qualifikation (siehe 9.1.3) angesprochen werden.

- potentieller Testpunkt

  Stelle, an der ein Testpunkt gesetzt werden kann (z.B. an den meisten Anweisungs-Anfängen). Welche Anweisungs-Anfänge potentielle Testpunkte bilden, und ob auch andere Stellen (z.B. Prozedur-Ende) potentielle Testpunkte darstellen, ist maschinenabhängig.

- Block

  Hauptprogramm, ein Paket, eine Prozedur oder eine Funktion.

Syntax-Elemente

Die folgenden syntaktischen Einheiten kommen in PATH-Kommandos häufig vor. Wie sie zu verwenden sind, ist in 9.2 beschrieben.

- Block-Qualifikation

  Folge von Namen, die durch einen Punkt abgeschlossen wird. Die einzelnen Namen sind jeweils durch einen Punkt voneinander getrennt. Der erste Name kann der einer Übersetzungseinheit oder eines vom aktuellen Testpunkt aus sichtbaren Unterprogramms sein. Jeder weitere, hinzukommende Name bezeichnet eine Prozedur oder Funktion, die unmittelbar im bisher qualifizierten Block (siehe 9.1.2) deklariert ist.

  Beispiele

  ```
  procx.procy.funcz.
  unity.procx.funcx.
  unity.
  ```

- Ausführungs-Qualifikation

  Block-Qualifikation, der eine Ausführungsnummer und ein Punkt folgt. Eine Ausführungsnummer besteht aus einem %-Zeichen und einer unmittelbar anschließenden vorzeichenlosen Integer-Zahl. Beim rekursiven Aufruf einer Prozedur entstehen verschiedene Ausführungen dieses Unterprogramms; mit %i kann nun eine bestimmte Ausführung bezeichnet werden; %1 bezeichnet die "aktuellste" Ausführung.

  Beispiele

  ```
  procx.%2.
  procx.funcy.%3.
  unity.procx.%4.
  ```

- Testpunkt-Spezifikation

  Liste potentieller Testpunkte (siehe 9.1.2). Sie besteht aus Zeilennummern und/oder Zeilennummernbereichen des Übersetzungslistings, die jeweils durch Kommas voneinander getrennt werden. Im einfachsten Fall besteht diese Liste aus einer Zeilennummer.

  Als Zeilennummernbereich kann auch %ALL geschrieben werden, um alle potentiellen Testpunkte einer Übersetzungseinheit oder eines Unterprogramms zu spezifizieren.

  Beispiele

  ```
  25
  30..40
  25, 30, 35..45, 50
  %ALL
  ```

Jeder Zeilennummer bzw. jedem Zeilennummernbereich kann eine Block-Qualifikation voranstehen.

Wenn der Zeilennummernbereich nicht %ALL ist, so darf die Block-Qualifikation nur eine Übersetzungseinheit, nicht aber ein Unterprogramm bezeichnen.

Vor %ALL sind hingegen sowohl Block-Qualifikationen zulässig, die eine Übersetzungseinheit bezeichnen (dadurch werden alle potentiellen Testpunkte dieser Übersetzungseinheit einschließlich aller darin deklarierten Unterprogramme spezifiziert) als auch solche, die ein Unterprogramm bezeichnen (dadurch werden alle potentiellen Testpunkte dieses Unterprogramms, aber nicht solche in darin enthaltenen Unterprogrammen, spezifiziert).

Beispiele

```
unity.30
unitx.35, unity.42, unitz.28..35
unitx.%ALL, unity.100..108, unitz.%ALL
unitx.procy.funcz.%ALL
procx.%ALL
```

Es ist auch möglich, Zeilennummern bzw. Zeilennummernbereiche einer Übersetzungseinheit zusammenzufassen, indem die Liste der einzelnen Zeilennummern bzw. Zeilennummernbereiche für diese Übersetzungseinheit in runde Klammern eingeschlossen wird.

Beispiel

```
unity.123, 128..134, unitz.(15, 25, 35..40), 78
```

In diesem Beispiel werden potentielle Testpunkte in der Zeile 123 der Übersetzungseinheit unity, in den Zeilen 78 und 128 bis 134 der aktuellen Übersetzungseinheit und in den Zeilen 15, 25 und 35 bis 40 der Übersetzungseinheit unitz spezifiziert.

Neben dieser Art der Testpunkt-Spezifikation (also einer Aufzählung von Zeilennummern bzw. Zeilennummernbereichen) gibt es noch die Testpunkt-Spezifikation %ALL UNITS, die alle potentiellen Testpunkte aller testbaren Übersetzungseinheiten des Testlings spezifiziert.

An das Ende der Testpunkt-Spezifikation (also der gesamten Liste) kann ": ENTER" angefügt werden. Das bewirkt, daß nur jene Testpunkte aus den in der Liste spezifizierten potentiellen Testpunkten ausgewählt werden, die erster potentieller Testpunkt irgend eines Blocks (siehe 9.1.2) sind.

Beispiele

```
%ALL_UNITS: ENTER
unitx.%ALL, unity.30..100: ENTER
```

Die Bedeutung der obigen Formen der Testpunkt-Spezifikation ist im Abschnitt 9.1.4.2 detailliert beschrieben.

Ferner ist es auch möglich, als Testpunkt-Spezifikation das Schlüsselwort EXCEPTION anzugeben. Dieser potentielle Testpunkt wird "Exception-Testpunkt" (siehe 9.1.4.4) genannt.

Hinweis

Das Schlüsselwort EXCEPTION darf nicht gemeinsam mit Zeilennummern, Zeilennummernbereichen, %ALL, %ALL_UNITS, Block-Qualifikationen oder :ENTER in der selben Testpunkt-Spezifikation auftreten.

- Faktor

  Integer-, Real-, Character- oder String-Literal, eine im Testling deklarierte Konstante oder Variable oder ein qualifizierter Mengenbildner. Strukturierte Konstante und Variable können indiziert ([ ]), qualifiziert (.) oder dereferenziert (@) sein. Als Index ist selbst wieder ein Faktor möglich. Weiters kann ein Teilbereich eines Array's oder String's durch Angabe eines Slice ([..]), dessen Grenzen selbst wieder Faktoren sind, angesprochen werden. Einer Konstanten oder Variablen kann eine Block-Qualifikation vorangestellt sein.

  Beispiele

```
unity.procz.i
unitx.a[4, b[i], 2..k]
p[j]@.x@.y[l]
a[1..10]
char_set_type (['A', 'X'..'Z'])
int_set_type ([5..i, k, v[1]..p.x, 100])
```

  In den ersten beiden Beispielen ist mit unity.procz. bzw. unitx. eine Block-Qualifikation angegeben.

  Es ist möglich, eindimensionale Array- und String-Slices (z.B. A [7..10]) auszugeben. Zuweisung und Vergleich von Array-Slices sind hingegen nicht erlaubt. Ebenso ist es verboten, Slices nochmals zu "slicen" oder zu indizieren.

  Beispiel

```
VAR A: ARRAY [1..10, 1..20] OF STRING[50];
```

  Erlaubt:

```
DISPLAY (A, A[5], A[6, 15, 43], A[7, 8, 30..40], A[I, J..16], A[2..8]);
```

  Verboten:

```
DISPLAY (A[3..4, 15], A[5..9, 10..20], A[I, 12..17][10..30]);
```

  Dies alles gilt auch für Variable, die über eine Ausführungs-Qualifikation angesprochen werden.

Testhilfe-Kommentare und -Optionen

Zum Zwecke der Dokumentation von Testläufen können in Testhilfe-Kommandos wie in Pascal-XT-Programme Kommentare der Form (*Kommentar*) oder {Kommentar} eingefügt werden.

Hinweis

Im Gegensatz zu Pascal-XT darf in PATH-Kommandos ein Kommentar nicht über Zeilengrenzen gehen.

Ferner gibt es die Möglichkeit, Testhilfe-Optionen in der Form von Pseudo-Kommentaren anzugeben. Derzeit ist nur eine Option implementiert, in späteren Versionen kommen möglicherweise noch weitere hinzu.

`{$R-}`

Nur (zur Analysezeit) als fehlerhaft erkannte Testhilfe-Kommandos werden als Eingabewiederholung auf das PATH-Ausgabe-Medium ausgegeben.

`{$R+}`

Alle eingegebenen Testhilfe-Kommandos werden auf dem PATH-Ausgabe-Medium wiederholt (Default).

Testpunkte

PATH-Aktionen erfolgen ausschließlich an "Testpunkten". Es gibt folgende Arten von Testpunkten:

Anfangs-Testpunkt
benutzergesetzte Testpunkte
Post-Mortem-Testpunkt
Exception-Testpunkt
Entry-Testpunkt

Anfangs-Testpunkt

Bevor die zu einem Programm gehörenden Pakete initialisiert und das Hauptprogramm ausgeführt wird, wird an einem impliziten Testpunkt ein GETCMD-Kommando ausgeführt. Dadurch wird es dem Benutzer ermöglicht, PATH-Kommandos einzugeben (insbesondere Testpunkte in den verschiedenen Übersetzungseinheiten zu setzen). Da ein Anfangstestpunkt in keinem bestimmten Paket liegt, herrscht globale Sichtbarkeit (siehe 9.1.2).

Meldungen am Anfangs-Testpunkt:

```
%% testpoint before program start
%% scope seen from outside all compilation units ———————————— (01)
* ———————————————————————————————————————————————————————— (02)
```

(01) Globale Sichtbarkeit

(02) Kommando-Eingabeaufforderung

Benutzergesetzte Testpunkte

Bei den Testpunkt-Kommandos AT, SLEEP, AWAKE und REMOVE, sowie bei SHOW WHERE wird durch eine Testpunkt-Spezifikation (siehe 9.1.3) festgelegt, an welchen Stellen (potentiellen Testpunkten) des zu testenden Programms Testpunkte gesetzt, gelöscht oder angezeigt werden sollen.

Im folgenden wird nur auf das Setzen von Testpunkten (AT-Kommando) eingegangen; Testpunkt-Spezifikationen in den anderen genannten Kommandos haben analoge Bedeutung.

Im einfachsten Fall (Setzen eines einzigen Testpunkts in der aktuellen Übersetzungseinheit, siehe 9.1.2), besteht die Testpunkt-Spezifikation nur aus einer Zeilennummer, die dem Übersetzungslisting entnommen werden kann.

Der Testpunkt wird dann in dieser Zeile am ersten potentiellen Testpunkt (siehe 9.1.2) gesetzt. Wenn die angegebene Zeile keinen potentiellen Testpunkt enthält, so wird ein Fehler gemeldet.

Beispiel

```
1    {$DEBUG} PROGRAM STATEMENTS;
2    VAR I: INTEGER;
3    BEGIN
4       I := 1;
5       I := 2; I := 3; I
6       := 4;
7    END.
```

AT 1 DO ...; führt zu einer Fehlermeldung;
AT 2 DO ...; führt zu einer Fehlermeldung;
AT 3 DO ...; führt zu einer Fehlermeldung;

AT 4 DO ...; setzt einen Testpunkt vor `I` := 1;
AT 5 DO ...; setzt einen Testpunkt vor `I` := 2;

AT 6 DO ...; führt zu einer Fehlermeldung;
AT 7 DO ...; setzt einen Testpunkt vor END.

Die Anweisungen I := 3; und I := 4; in Zeile 5 sind von der Testhilfe nicht direkt (einzeln) ansprechbar.

Es ist daher ratsam, bei Programmen, die man testen will, für jede Anweisung eine Zeile zu verwenden.

Es ist auch möglich, mit einem AT-Kommando Testpunkte an mehrere aufeinanderfolgende potentielle Testpunkte zu setzen, indem man einen Zeilennummernbereich in der Form Zeilennummer .. Zeilennummer oder in der Form %ALL angibt.

So bewirkt in obigem Beispiel das Kommando

```
AT 5..6 DO ...;
```

daß vor die Anweisungen I := 2; I := 3; und I := 4; je ein Testpunkt gesetzt wird (also 3 Testpunkte).

Mit dem Kommando

```
AT %ALL DO ...;
```

wird je ein Testpunkt vor die Anweisungen I := 1; I := 2; I := 3; I := 4 und vor END gesetzt (also 5 Testpunkte).

Somit ist es also auch möglich, Testpunkte vor Anweisungen zu setzen, die nicht in einer eigenen Zeile beginnen (z.B. I := 3;). Solche Anweisungen können aber wie erwähnt nicht einzeln sondern nur innerhalb eines Zeilennummernbereichs angesprochen werden.

Mit dem Kommando

```
AT %ALL: ENTER DO ...;
```

wird in diesem Beispiel nur ein Testpunkt in Zeile 4 (vor die Anweisung I := 1;) gesetzt.

Indem man als Testpunkt-Spezifikation mehrere Zeilennummern und/oder Zeilennummernbereiche durch Komma getrennt aneinander reiht, kann man mit einem AT-Kommando Testpunkte an mehrere nicht notwendigerweise aufeinanderfolgende Anweisungen setzen.

Beispiel

```
AT 550, 576..612, 434, 639 DO ...;
```

Alle bisher beschriebenen Formate erlauben es nur, Testpunkte in der aktuellen Übersetzungseinheit zu setzen. Die Syntax der Testpunktspezifikation ermöglicht aber auch ein Ansprechen von potentiellen Testpunkten in anderen Übersetzungseinheiten des Testlings.

Beispiel

```
AT 125, unitx.30, unity.(120, 130..135), 150, unitz.%ALL DO ...;
```

setzt Testpunkte an den ersten potentiellen Testpunkt in den Zeilen 125 und 150 der aktuellen Übersetzungseinheit, in die Zeile 30 der Übersetzungseinheit unitx, in die Zeile 120 und an aufeinanderfolgende potentielle Testpunkte in den Zeilen 130 bis 135 der Übersetzungseinheit unity, sowie an alle potentiellen Testpunkte der Übersetzungseinheit unitz.

Fügt man am Ende der obigen Testpunkt-Spezifikation ": ENTER" an, also

```
AT 125, unitx.30, unity.(120, 130..135), 150, unitz.%ALL: ENTER DO ...;
```

so werden aus den spezifizierten potentiellen Testpunkten nur jene ausgewählt, die die Eigenschaft besitzen, erster potentieller Testpunkt eines Blocks (siehe 9.1.2) zu sein.

Beispiel

```
AT unitz.(87, 316..318, 772..775, 961) DO ...;
```

Die Klammern in diesem Beispiel sind notwendig, denn das Kommando

```
AT unitz.87, 316..318, 772..775, 961 DO ...;
```

würde Testpunkte an den ersten potentiellen Testpunkt in Zeile 87 der Übersetzungseinheit unitz, sowie in den Zeilen 316 bis 318, 772 bis 775 und 961 der gerade aktuellen Übersetzungseinheit setzen.

Die Testpunkt-Spezifikation %ALL_UNITS spezifiziert alle potentiellen Testpunkte aller Übersetzungseinheiten des Testlings, die mit der Option DEBUG=ON oder DEBUG=RESTRICTED übersetzt wurden und deren Testtabellen verfügbar sind. Insbesondere eignen sich %ALL und %ALL_UNITS dazu, alle benutzergesetzten Testpunkte einer Übersetzungseinheit bzw. aller Übersetzungseinheiten des Testlings zu löschen (REMOVE-Kommando), zu deaktivieren (SLEEP-Kommando) oder zu aktivieren (AWAKE-Kommando).

Post-Mortem-Testpunkt

Wenn beim Testen eines Programms mit PATH in einem Pascal-XT-Programmteil (siehe 7) ein Laufzeitfehler auftritt oder von einem fremdsprachigen Unterprogramm dorthin weitergereicht wird und wenn der Laufzeitfehler dort nicht behandelt wird, dann werden folgende Schritte ausgeführt:

(1) Wenn der getestete Programmteil das Hauptprogramm ist, gibt das Pascal-XT-Laufzeitsystem die Fehlerursache und die dynamische Aufrufkette (siehe 10.2.4) aus. Anschließend wird Schritt (2) ausgeführt.

(2) In jedem Fall verhält sich die Testhilfe so, als ob an einer der beiden folgenden Stellen ein benutzergesetzter Testpunkt mit Eingabeaufforderung (GETCMD-Kommando) gesetzt wäre:
   - an der Anweisung, wo der Fehler auftritt.
   - an der Stelle, wo das fremdsprachige Unterprogramm aufgerufen wird, das den Laufzeitfehler weiterreicht.

PATH gibt dann die Eingabeaufforderung * aus und der Benutzer kann PATH-Kommandos eingeben, um die Fehlerursache herauszufinden. Es bieten sich vor allem die PATH-Kommandos DUMP, SHOW CALLS und DISPLAY an.

Das PATH-Kommando KILL führt zum Programmabbruch.

Das PATH-Kommando RESUME zeigt je nach Art des getesteten Pascal-XT-Programmteils unterschiedliche Wirkung:

In einem Hauptprogramm wird der Testling beendet, sofern kein erneuter Testlauf (siehe 9.5 "Restart") gewünscht wird.

In einem anderen Programmteil propagiert das Pascal-XT-Laufzeitsystem den Fehler an den Rufer. Durch die Fehlerpropagierung ist es möglich, daß weitere Pascal-XT-Programmteile den Fehler gemeldet bekommen. Die Testhilfe bietet dann für jeden Pascal-XT-Programmteil, der den Fehler propagiert und nicht behandelt, einen Post-Mortem-Testpunkt an.

Beispiel

```
 1     {$DEBUG=ON} PROGRAM DIVIDE
 2     VAR
 3        dividend:  Integer;
 4        divisor:   Integer;
 5        result:    REAL;
 6
 7     BEGIN
 8        { Die Zuweisung eines Wertes an die }
 9        { Variablen dividend und divisor    }
10        { erfolgt von der Testhilfe aus     }
11        result := dividend / divisor;
12     END.
```

Nach dem Starten des Programms:

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at divide.11 do getcmd;
*resume
%% program continued
%% testpoint at line DIVIDE.11
*assign dividend := 5;
*assign divisor := 0; {provoziert im Testling eine Division durch 0}
*resume;
%% program continued
 NUMERIC_ERROR ( 104) RAISED FROM  DIVIDE.DIVIDE  AT 000E5F3A
%% testpoint because of unhandled Numeric_Error at line DIVIDE.11
*dump;
%% global variables of DIVIDE:
%%    dividend = 5
%%    divisor = 0
%%    result = 0.000000000000000E+00
*resume;
%% program aborted
```

Exception-Testpunkt

Bei den Testpunkt-Kommandos AT, SLEEP, AWAKE, REMOVE und SHOW WHERE kann durch Angabe des Schlüsselwortes EXCEPTION ein sogenannter Exception-Testpunkt gesetzt, deaktiviert, (re)aktiviert, gelöscht und angezeigt werden.

Wenn ein Exception-Testpunkt gesetzt ist und wenn in einem Pascal-XT-Programmteil (siehe 7) ein Laufzeitfehler auftritt oder dorthin von einem fremdsprachigen Unterprogramm weitergereicht wird, meldet sich die Testhilfe an folgenden Stellen:

- an der Anweisung, wo der Fehler auftritt.
- in allen Pascal-XT-Programmteilen an der Stelle, an der das fremdsprachige Unterprogramm aufgerufen wird, das den Fehler weiterreicht.

Unabhängig davon, ob für den Fehler ein Ausnahmebehandler definiert ist, wird ein Exception-Testpunkt ausgelöst und die dafür angegebenen PATH-Kommandos werden ausgeführt. Dadurch kann der Benutzer die Fehlerursache auch dann herausfinden, wenn der Fehler durch einen Ausnahmebehandler behandelt wird. Erst anschließend wird nach einem Ausnahmebehandler gesucht, sofern der Testling nicht durch das PATH-Kommando KILL abgebrochen wird. Ist ein Ausnahmebehandler definiert, wird der Testling dort fortgesetzt. Ist kein Ausnahmebehandler definiert, wird ein Post-Mortem-Testpunkt ausgelöst.

Beispiel

```
 1     {$DEBUG=ON} PROGRAM POWER;
 2     VAR
 3        result: Real;
 4
 5     FUNCTION power_of_e (x: Integer): Real;
 6     BEGIN
 7        power_of_e := Exp (x);
 8     EXCEPTION
 9        IF Error_Number = Numeric_Error THEN
10           power_of_e := Maxreal
11        ELSE { propagate other errors }
12           Raise (Error_Number);
13     END;
14
15     BEGIN
16        result := power_of_e (maxint);
17     END.
```

Nach dem Starten des Programms:

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at exception do display ('Ausnahme ist aufgetreten'); ———————————— (01)
*at power.16 do getcmd;
*resume;
%% program continued
%% testpoint at line POWER.16
*show where exception; ———————————————————————————————————————— (02)
%% testpoint at exception ———————————————————————————————————— (03)
*at exception do begin dump; getcmd; end; ————————————————————— (04)
*resume;
%% program continued
%% testpoint because of Numeric_Error at line POWER.7 (power_of_e) ——— (05)
%% 'Ausnahme ist aufgetreten' ———————————————————————————————— (06)
%% parameters and local variables of power_of_e: ————————————— (07)
%%    x = 2147483647
%% global variables of POWER;
%%    result = 0.000000000000000E+00
*resume; ———————————————————————————————————————————————————— (08)
%% program continued
```

(01) Falls im Testling eine Ausnahme auftritt, soll die Testpunkt-Meldung ausgegeben (05) und das DISPLAY-Kommando ausgeführt werden (06).

(02) Es soll angezeigt werden, ob der Exception-Testpunkt gesetzt ist.

(03) Gibt an, daß der Exception-Testpunkt gesetzt ist.

(04) Das in (01) angegebene im Fehlerfall auszuführende Kommando wird um ein DUMP- und ein GETCMD-Kommando erweitert.

(05) Ausgabe der Testpunkt-Meldung am Exception-Testpunkt.

(06) Ausgaben des in (01) definierten DISPLAY-Kommandos.

(07) Ausgaben des in (04) definierten DUMP-Kommandos.

(08) Die Ausführung des in (04) definierten GETCMD-Kommandos bewirkt eine Unterbrechung des Testlaufs, der durch die Eingabe des RESUME-Kommandos wieder fortgesetzt wird.

Entry-Testpunkt

Wird eine Pascal-XT-Entry-Prozedur aus einem fremdsprachigen Programmteil aufgerufen, so wird, wenn das den Entry enthaltende Pascal-XT-Paket der Testhilfe noch nicht bekannt ist, noch vor der Paket-Initialisierung an einem impliziten Testpunkt (Entry-Testpunkt) ein GETCMD-Kommando ausgeführt. Dies ermöglicht es dem Benutzer PATH-Kommandos einzugeben, insbesondere Testpunkte in diesem Paket und in allen durch WITH-Klauseln referierten Paketen zu setzen.

Am Entry-Testpunkt herrscht "globale Sichtbarkeit" (siehe Kapitel 9.1.2), die Meldungen am Entry-Testpunkt lauten:

```
testpoint before entry call
scope seen from outside all compilation units
```

Beim erneuten Aufruf desselben oder eines anderen Entries eines der Testhilfe bereits bekannten Pakets wird kein Entry-Testpunkt mehr ausgelöst, da der Benutzer bereits früher die Gelegenheit hatte, in diesem Paket Testpunkte zu setzen und die vielen Entry-Testpunkte bei häufigen Aufrufen von Entries lästig wären.

Beispiel

Das Assembler-Hauptprogramm ruft den Pascal-XT-Entry ’divide’ auf, der eine Division durchführt.

Assembler-Hauptprogramm:

```
 1     CALLDIV  CSECT
 2     *
 3              EXTRN DIVIDE
 4     *
 5     R0       EQU   0
 6     R1       EQU   1
 7     R10      EQU   10
 8     R13      EQU   13            Adresse der Save Area
 9     R14      EQU   14            Rueckkehradresse
10     R15      EQU   15            Einsprungadresse
11     *
12              BALR  R10,R0
13              USING *,R10
14              LA    R1,PARLIST    Adresse der Parameterliste laden
15              LA    R13,SAVEAREA  Adresse der Save Area laden
16              L     R15,=A(DIVIDE) Adresse der Pascal-Prozedur laden
17              BALR  R14,R15       Aufruf der Pascal-Entry-Prozedur
18              TERM
19     *
20     PARLIST  DC    A(PAR1)
21              DC    A(PAR2)
22              DC    A(PAR3)
23     PAR1     DC    F’10’         Dividend
24     PAR2     DC    F’5’          Divisor
25     PAR3     DS    D             Ergebnis
26     SAVEAREA DS    18A           Save Area fuer Pascal
27              END
```

Pascal-XT-Paket-Spezifikation:

```
 1     PACKAGE DIVI;
 2     ENTRY PROCEDURE divide (VAR x, y: Integer; VAR z: Real);
 3     END.
```

Pascal-XT-Paket-Implementierung:

```
 1     {$Debug=On}
 2     WITH Errors;
 3     PACKAGE BODY DIVI;
 4     PROCEDURE divide (VAR x, y: Integer; VAR z: Real);
 5     BEGIN
 6        z := x / y;
 7     EXCEPTION
 8        IF Error_Number = Numeric_Error THEN
 9           z := Minreal
10        ELSE { propagate other errors }
11           Errors.ReRaise;
12     END;
13
14     BEGIN
15     END.
```

Nach dem Starten des Programms:

```
%% testpoint before entry call ———————————————————————————— (01)
%% scope seen from outside all compilation units
*show units; ————————————————————————————————————————————— (02)
%% compilation-units:
%% DIVI, compiled 90-01-12 10:17:37, complete testtable ?
*at divi.12 do getcmd; ——————————————————————————————————— (03)
*resume; ————————————————————————————————————————————————— (04)
%% program continued
%% testpoint at line DIVI.12 (divide) ——————————————————————— (05)
*display (%param);
%% parameters of divide:
%%    x = 10
%%    y = 5
%%    z =  2.000000000000000E+00
*resume; ————————————————————————————————————————————————— (06)
%% program continued
%% program terminated
```

(01) Ausgabe der Testpunkt-Meldung am Entry-Testpunkt.

(02) Ausgabe einer Liste aller zu diesem Zeitpunkt der Testhilfe bekannten Übersetzungseinheiten.

(03) Setzen eines Testpunkts in der Pascal-XT-Prozedur.

(04) Verlassen des Entry-Testpunkts.

(05) Der in (03) gesetzte Testpunkt wird erreicht.

(06) Verlassen des benutzergesetzten Testpunkts.

Hinweise

- Der Entry-Testpunkt wird also noch nicht beim Starten des fremdsprachigen Hauptprogramms angeboten, sondern erst beim ersten Aufruf einer Pascal-XT-Prozedur.
- Dabei werden der Testhilfe das Paket, das den Entry enthält und die von diesem aus direkt und indirekt referierten Pakete bekannt. Es kann aber noch weitere Pakete geben. Diese werden der Testhilfe erst bekannt, wenn ein in ihnen enthaltener Entry aufgerufen wird. Dabei wird dann neuerlich ein Entry-Testpunkt angeboten, der es dem Benutzer ermöglicht, nun auch in weiteren Paketen Testpunkte zu setzen, u.s.w. Welche Pakete der Testhilfe jeweils (schon) bekannt sind, kann mit dem PATH-Kommando SHOW UNITS (kurz "SH U") jederzeit abgefragt werden.
- Da beim Aufruf eines Entries eines der Testhilfe bereits bekannten Pakets kein Entry-Testpunkt mehr angeboten wird, hat der Benutzer zu diesem Zeitpunkt keine Möglichkeit mehr, in diesem Entry Testpunkte zu setzen. Das bedeutet, daß er bereits an dem schon früher angebotenen Entry-Testpunkt, im Zuge dessen das Paket der Testhilfe bekannt wird, die Gelegenheit zum Setzen von Testpunkten in diesem Paket nützen muß.

Gültigkeitsbereich von Bezeichnern

Der Gültigkeitsbereich (Scope) von Bezeichnern wird für einen Testpunkt (sowohl beim Erreichen eines Testpunkts, als auch im Inneren eines AT-Kommandos) entsprechend der statischen Prozedurschachtelung im Pascal-XT-Quellprogramm nachgebildet.

Auch die in Pascal-XT vordefinierten Konstanten-Bezeichner (z.b. Maxreal, Index_Error,...) sind in PATH (DISPLAY, ASSIGN, IF) ansprechbar.

Auch der durch WITH-Anweisungen im Quellprogramm erweiterte Scope wird von der Testhilfe nachgebildet.

Bezeichner, die durch gleichlautende Bezeichner in inneren Prozeduren verdeckt sind, können durch Voranstellen einer Block-Qualifikation (siehe 9.1.3) angesprochen werden.

Bezeichner, die zwar im Quellprogramm nicht sichtbar sind, da sie weder im aktuellen noch in einem umgebenden Unterprogramm, wohl aber in einer Unterprogramm-Ausführung der dynamischen Aufrufkette deklariert sind, können durch Voranstellen einer Ausführungs-Qualifikation (siehe 9.1.3) angesprochen werden.

Beispiel für Sichtbarkeit

```
 1    {$DEBUG}
 2    PROGRAM sichtbarkeit;
 3    VAR i: integer;
 4        rec: RECORD
 5           i: integer;
 6           END;
 7    PROCEDURE proc;
 8    VAR i: integer;
 9    BEGIN
10        i:= 1;
11    END;
12
13    BEGIN
14       WITH rec DO BEGIN
15          i := 2;
16          END;
17       i := 3;
18       proc;
19    END.
```

Zeilennummern	Bedeutung
10	Damit wird ein Testpunkt in der Prozedur proc angesprochen. Wird an diesem Testpunkt die Variable i angesprochen, so bezieht man sich auf die zu proc prozedurlokale Variable i (= proc.i = sichtbarkeit.proc.i, entsprechend Scope im Quellprogramm). Mit sichtbarkeit.i wird hingegen die verdeckte, im Hauptprogramm (sichtbarkeit) deklarierte globale Variable i angesprochen.

17,18 Es werden Testpunkte im Hauptprogramm angesprochen; i bezieht sich auf die Hauptprogrammvariable i (= sichtbarkeit.i, entsprechend Scope im Quellprogramm).

15 Da auch der durch WITH-Anweisungen im Quellprogramm veränderte Scope von der Testhilfe nachgebildet wird, wird an dieser Stelle mit i (genau wie im Quellprogramm) die RECORD-Komponente i und nicht die Hauptprogramm-Variable i angesprochen. Letztere kann durch sichtbarkeit.i angesprochen werden.

Wenn mit einem AT-Kommando mehrere Testpunkte in verschiedenen (Unter-)Programmen gesetzt werden, so können im inneren Kommando dieses AT-Kommandos nur diejenigen Namen angesprochen werden, die vom engsten gemeinsamen umschließenden (Unter-)Programm aus sichtbar sind.

Wenn mit einem AT-Kommando mehrere Testpunkte in verschiedenen Übersetzungseinheiten gesetzt werden, so herrscht im inneren Kommando dieses AT-Kommandos nur die "globale Sichtbarkeit" (siehe 9.1.2).

Die in den vorigen beiden Absätzen beschriebene Einschränkung der Sichtbarkeit gilt aber nur im inneren Kommando (deferred action) solch eines AT-Kommandos. Wenn hingegen aufgrund eines im AT-Kommando enthaltenen GETCMD-Kommandos an einem der gesetzten Testpunkte in den Kommando-Eingabe-Modus verzweigt wird, so können dort natürlich alle von diesem Testpunkt aus sichtbaren Namen angesprochen werden.

Auch Bezeichner, die gleich lauten wie PATH-Kommandos oder deren Abkürzungen (z.B. SHOW, SH, CALLS, C, DISPLAY, D, ...) können angesprochen werden.

Beispiel

```
1  {$DEBUG} PROGRAM xyz;
2  CONST remove = True;
3  VAR display: INTEGER;
. . .

*display (display, remove);
%% display = 347
%% remove = True
*assign display := 1;
```

Ansprechbarkeit von Bezeichnertypen

Programmvariable und Programmkonstanten aller Typen können symbolisch angesprochen und ihrem Typ entsprechend ausgegeben, zugewiesen und verglichen werden. Die Regeln der Typverträglichkeit entsprechen denen in Pascal-XT.

Ganze RECORD-, ARRAY- und SET-Variable können (auch im DISPLAY-Kommando) angesprochen werden. Im Falle der Ausgabe von Records mit Varianten werden die Komponenten des fixen Teils ausgegeben und die gerade gültige Variante, sofern ein Kennungsfeld existiert oder der Kennungstyp mit dem Kennungstyp der übergeordneten Variante übereinstimmt. Bei Variantenteilen ohne Kennungsfeld (... CASE typename OF ...) werden in diesem Fall nur die Komponenten des fixen Teils (sofern vorhanden) ausgegeben; die Komponenten der Varianten können dann nur einzeln angesprochen werden.

Erzeugung der Testtabellen

Um eine Pascal-XT-Übersetzungseinheit mit PATH testen zu können, muß bei der Übersetzung mit dem Pascal-XT-Compiler die Option DEBUG=ON oder DEBUG=RESTRICTED zur Erzeugung einer Testtabelle gesetzt sein.

Eine "komplette Testtabelle" wird erzeugt, wenn DEBUG in einer Paket-Implementierung bzw. in einem Hauptprogramm gesetzt ist. Bei DEBUG=ON sind an Testpunkten in dieser Übersetzungseinheit alle PATH-Kommandos verwendbar, bei DEBUG=RESTRICTED sind keine Zuweisungen (ASSIGN-Kommando) erlaubt.

Eine "partielle Testtabelle" wird erzeugt, wenn DEBUG in einer Paket-Spezifikation, nicht aber in der zugehörigen Paket-Implementierung gesetzt ist. Sie ermöglicht ein Ansprechen aller in der Paket-Spezifikation deklarierten Variablen, Konstanten und Typen. Variable, deren Typ ein privater Zeigertyp ist, können nicht dereferenziert werden.

Eine "minimale Testtabelle" wird erzeugt, wenn DEBUG=OFF sowohl in der Paket-Spezifikation als auch in der zugehörigen Paket-Implementierung gilt. Sie ermöglicht nur die Ausgabe der dynamischen Aufruf-Kette (insbesondere im Fehlerfall).

Hinweis

Bei ausgeschalteter Option GENERATE wird implizit auch DEBUG ausgeschaltet. Durch Einschalten der Option DEBUG wird implizit die Option OPTIMIZE ausgeschaltet.

Die PATH-Kommandos

Beim Ausführen eines GETCMD-Kommandos an einem Testpunkt wird eine Kommandofolge vom PATH-Eingabe-Medium eingelesen. Die einzelnen Kommandos dieser Folge werden jeweils durch Semikolon abgeschlossen. Kommandos werden, sobald sie mit einem Semikolon abgeschlossen und als syntaktisch und semantisch richtig erkannt sind, ausgeführt.

Beim Erreichen eines benutzergesetzten Testpunkts (siehe 9.1.4.2) werden die mit einem oder mehreren AT-Kommandos für diesen Testpunkt festgelegten "deferred actions" ausgeführt (interpretiert). Weitere Kommandos können nur bei der Ausführung eines GETCMD-Kommandos eingegeben werden.

Das Semikolon kann am Ende einer Eingabezeile entfallen, wenn an dieser Stelle des Kommandos syntaktisch ohnehin nur noch ein Semikolon erlaubt ist.

Ein Semikolon am Ende einer Zeile darf daher nicht weggelassen werden, wenn das Kommando in der nächsten Zeile fortgesetzt werden könnte, also nach einem

- IF-Kommando ohne ELSE-Zweig, denn die nächste Zeile könnte ja den ELSE-Zweig enthalten,
- ASSIGN-Kommando, denn der Ausdruck auf der rechten Seite könnte ja in der nächsten Zeile noch indiziert, qualifiziert oder dereferenziert werden,
- REMOVE-, SLEEP-, AWAKE- oder SHOW WHERE-Kommando, denn die nächste Zeile könnte ja noch Teile der Testpunkt-Spezifikation enthalten,
- EDIT-, SYSTEM- oder SWITCH-Kommando ohne String, denn dieser könnte ja noch in der nächsten Zeile stehen.

Bei Fehlern wird das fehlerhafte Kommando einschließlich der Stelle, an welcher der Fehler erkannt wurde, mit einer entsprechenden Fehlermeldung auf das PATH-Ausgabe-Medium ausgegeben und ein neues Kommando erwartet.

Es gibt Testpunktkommandos und Aktionskommandos.

Das SHOW-Kommando ist unter den Aktionskommandos beschrieben, obwohl eines seiner Formate, nämlich SHOW WHERE die Eigenschaften eines Testpunktkommandos hat.

Einige Kommandos und andere PATH-Schlüsselwörter können abgekürzt werden:

ASSIGN	A	SHOW CALLS	SH C
BEGIN	B	SHOW UNITS	SH U
DISPLAY	D	SHOW WHERE	SH W
EDIT	ED		
GETCMD	G	SWITCH INPUT	SW I
RESUME	R	SWITCH OUTPUT	SW O
SYSTEM	SY	SWITCH LIST	SW L

Testpunkt-Kommandos

Fünf der Testpunkt-Kommandos dienen zum Setzen (AT-Kommando), Löschen (REMOVE-Kommando), Deaktivieren (SLEEP-Kommando), Aktivieren (AWAKE-Kommando) und Anzeigen (SHOW WHERE-Kommando) von benutzergesetzten Testpunkten (siehe 9.1.4.2).

Diese Kommandos können eine Testpunkt-Spezifikation (siehe 9.1.3, 9.1.4.2 und 9.1.4.4) enthalten und beziehen sich auf potentielle Testpunkte (siehe 9.1.2).

Die beiden restlichen Testpunkt-Kommandos dienen zum Einschalten (GETCMD-Kommando) und Ausschalten (RESUME-Kommando) des Kommando-Eingabe-Modus.

AT-Kommando

Format 1:

```
AT testpunkt-spezifikation DO kommando
```

Dieses Kommando setzt Testpunkte an den durch die Testpunktspezifikation (siehe 9.1.3, 9.1.4.2 und 9.1.4.4) spezifizierten potentiellen Testpunkte (siehe 9.1.2). Der Programmablauf wird bei Erreichen eines der gesetzten Testpunkte unterbrochen und das nach dem DO stehende Kommando (z.B. GETCMD-, Verbund- oder auch AT-Kommando) wird dann ausgeführt ("deferred action").
Nach Ausführung des Kommandos wird der Programmlauf fortgesetzt. Eine Eingabeaufforderung erfolgt am Testpunkt nur dann, wenn die deferred action dieses Testpunkts ein GETCMD-Kommando ist oder eines enthält (mit RESUME wird an der durch GETCMD unterbrochenen Stelle fortgesetzt).

Ein AT-Kommando auf einen bestehenden Testpunkt erweitert die für diesen Testpunkt auszuführenden Kommandos (additiv) in der Weise, daß die Kommandos in der Reihenfolge ihres Absetzens ausgeführt werden.

Beispiel

```
1  {$DEBUG}
2  PROGRAM SCHLEIFE;
3  VAR i, k: INTEGER;
4  BEGIN
5     FOR i := 1 TO 5 DO BEGIN
6        k := i;
7        k := k;
8        END;
9  END.
```

Starten des Testprogramms:

Vor dem Programmablauf wird an einem impliziten Testpunkt ("Anfangstestpunkt") die Eingabe von Testhilfe-Kommandos gefordert.

```
%% testpoint before program start
%% scope seen from outside all compilation units ———————————— (01)
*at schleife.6..7 do display (i, k); ———————————————————————— (02)
*resume; ———————————————————————————————————————————————————— (03)
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 1
%% k = 0
%% program continued
%% testpoint at line SCHLEIFE.7
%% i = 1
%% k = 1
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 2
%% k = 1
%% program continued
%% testpoint at line SCHLEIFE.7
%% i = 2
%% k = 2
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 3
%% k = 2
%% program continued
%% testpoint at line SCHLEIFE.7
%% i = 3
%% k = 3
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 4
%% k = 3
%% program continued
%% testpoint at line SCHLEIFE.7
%% i = 4
%% k = 4
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 5
%% k = 4
%% program continued
%% testpoint at line SCHLEIFE.7
%% i = 5
%% k = 5
```

(01) Globale Sichtbarkeit (siehe 9.1.2).

(02) In den Zeilen 6 bis 7 sollen die Werte von i und k ausgegeben werden.

(03) Mit RESUME wird der Anfangs-Testpunkt verlassen.

Format 2:

```
AT DO kommando
```

Mit dem AT-Kommando ohne Testpunkt-Spezifikation kann an dem Testpunkt, an dem das AT-Kommando ausgeführt wird, eine weitere deferred action abgelegt werden, die dann beim nächsten Erreichen dieses Testpunkts ausgeführt wird.
Ist dieser Testpunkt der Anfangs-Testpunkt (siehe 9.1.4.1), so wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.
Wird das AT-Kommando ohne Testpunkt-Spezifikation hingegen am Post-Mortem- oder Exception-Testpunkt (siehe 9.1.4.3 und 9.1.4.4) ausgeführt, so wirkt es nicht auf diesen Testpunkt, sondern auf den potentiellen Testpunkt (siehe 9.1.2) an dieser Stelle.

GETCMD-Kommando

```
GETCMD
```

Ein GETCMD-Kommando bewirkt ein Unterbrechen des Testlaufs und den Übergang in den Eingabezustand, d.h. PATH-Kommandos werden erwartet.

Vom PATH-Eingabe-Medium können beliebig viele Testhilfe-Kommandos eingegeben werden.

Hinweis

Ein GETCMD-Kommando ist nur im Inneren eines AT-Kommandos (also in einer deferred action) erlaubt.

RESUME-Kommando

RESUME

Die durch ein GETCMD-Kommando ermöglichte Kommando-Eingabe wird beendet.

Der Testlauf wird an der durch das GETCMD-Kommando unterbrochenen Stelle fortgesetzt, d.h. eventuelle weitere nach einem GETCMD-Kommando befindliche Kommandos werden abgearbeitet (können auch weitere GETCMD-Kommandos sein).

Bei der Ausführung eines RESUME-Kommandos wird der Testhilfeeingabepuffer gelöscht; das bedeutet, daß eventuelle weitere in der selben Zeile stehende Kommandos beim nächsten GETCMD-Kommando nicht mehr verarbeitet werden.

Ein RESUME-Kommando innerhalb eines AT-Kommandos (also in einer deferred action) ist sinnlos und daher verboten.

Sind alle für den Testpunkt definierten Kommandos (die gesamte deferred action) abgearbeitet, so wird der Programmlauf fortgesetzt.

Beispiel für das Zusammenspiel von AT, GETCMD und RESUME

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at schleife.6 do getcmd; resume;
%% program continued
%% testpoint at line SCHLEIFE.6
*display (i); resume;
%% i = 1
%% program continued
%% testpoint at line SCHLEIFE.6
*display (k); resume;
%% k = 1
%% program continued
*
```

REMOVE-Kommando

Format 1:

```
REMOVE testpunkt-spezifikation
```

Zur Bedeutung der Testpunkt-Spezifikation siehe 9.1.3, 9.1.4.2 und 9.1.4.4.

Die durch die Testpunkt-Spezifikation spezifizierten Testpunkte, die vorher mittels AT-Kommando(s) gesetzt wurden, werden wieder gelöscht.

Wenn beim weiteren Ablauf des Testlings diese potentiellen Testpunkte erreicht werden, so wird er nicht mehr unterbrochen, und die deferred actions werden nicht mehr ausgeführt.

Wenn unter den durch die Testpunkt-Spezifikation spezifizierten potentiellen Testpunkten kein einziger gesetzter Testpunkt gefunden (und gelöscht) wird, so erfolgt eine Fehlermeldung (siehe 9.3).

Wenn aber z.B. bei der Ausführung des Kommandos

```
REMOVE 63, 135, 716;
```

wenigstens ein Testpunkt gefunden und gelöscht wird, so gilt das Kommando als erfolgreich und es erfolgt keine Fehlermeldung (egal, ob auch in den anderen Zeilen Testpunkte gefunden werden).

Beispiel

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at schleife.6 do getcmd; at schleife.7 do getcmd; resume;
%% program continued
%% testpoint at line SCHLEIFE.6
*display (i); resume;
%% i = 1
%% program continued
%% testpoint at line SCHLEIFE.7
*remove 6;                                                    (01)
*resume;
%% program continued
%% testpoint at line SCHLEIFE.7
*
```

(01) In Zeile 6 wird nicht mehr gehalten.

Format 2:

```
REMOVE
```

Das REMOVE-Kommando ohne Testpunkt-Spezifikation bewirkt, daß der Testpunkt, an dem das REMOVE-Kommando ausgeführt wird, gelöscht wird.
Ist dieser Testpunkt der Anfangs-Testpunkt (siehe 9.1.4.1), so wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.
Wenn das Kommando am Post-Mortem- oder Exception-Testpunkt (siehe 9.1.4.3 und 9.1.4.4) ausgeführt wird, so wirkt es nicht auf diesen Testpunkt. Befindet sich aber an der Stelle, an der der Fehler aufgetreten ist, (zufällig) ein benutzergesetzter Testpunkt (siehe 9.1.4.2), so wird dieser gelöscht. Wenn dort kein Testpunkt gesetzt ist, wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.

SLEEP-Kommando

Format 1:

```
SLEEP testpunkt-spezifikation
```

Zur Bedeutung der Testpunkt-Spezifikation siehe 9.1.3, 9.1.4.2 und 9.1.4.4.

Die durch die Testpunkt-Spezifikation spezifizierten, vorher mittels AT-Kommando(s) gesetzten Testpunkte werden bis auf weiteres deaktiviert; die für die Testpunkte definierten Kommandos (deferred actions) werden beim Erreichen nicht mehr ausgeführt. Im Gegensatz zum REMOVE-Kommando existieren die Kommandos aber noch logisch und können mit dem AWAKE-Kommando wieder aktiviert werden.

Wird ein AT-Kommando auf einen mit dem SLEEP-Kommando deaktivierten Testpunkt abgesetzt, so wirkt das SLEEP-Kommando auch auf das neu hinzugekommene Kommando.

Analyse und Ausführung des SLEEP-Kommandos erfolgen analog dem REMOVE-Kommando.

Format 2:

```
SLEEP
```

Das SLEEP-Kommando ohne Testpunkt-Spezifikation bewirkt, daß der Testpunkt, an dem das SLEEP-Kommando ausgeführt wird, deaktiviert wird.
Ist dieser Testpunkt der Anfangs-Testpunkt (siehe 9.1.4.1), so wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.
Wenn das Kommando am Post-Mortem- oder Exception-Testpunkt (siehe 9.1.4.3 und 9.1.4.4) ausgeführt wird, so wirkt es nicht auf diesen Testpunkt. Befindet sich aber an der Stelle, an der der Fehler aufgetreten ist, (zufällig) ein benutzergesetzter Testpunkt (siehe 9.1.4.2), so wird dieser deaktiviert. Wenn dort kein Testpunkt gesetzt ist, wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.

AWAKE-Kommando

Format 1:

```
AWAKE testpunkt-spezifikation
```

Zur Bedeutung der Testpunkt-Spezifikation siehe 9.1.3, 9.1.4.2 und 9.1.4.4.

Die durch die Testpunkt-Spezifikation spezifizierten, vorher mittels SLEEP-Kommando(s) deaktivierten Testpunkte werden wieder gültig; d.h. die vor Durchführung des SLEEP-Kommandos für diese Testpunkte gültigen Kommandos und eventuelle danach mittels AT-Kommando(s) gesetzte weitere Kommandos werden ab dem nächsten Erreichen des Testpunkts wieder ausgeführt.

Wenn unter den durch die Testpunkt-Spezifikation spezifizierten potentiellen Testpunkten kein einziger gesetzter Testpunkt gefunden (und reaktiviert) wird, so erfolgt eine Fehlermeldung (siehe 9.3).

Beispiel

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at schleife.6 do display (i); at schleife.7 do getcmd; resume;
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 1
%% program continued
%% testpoint at line SCHLEIFE.7
*display (i, k);
%% i = 1
%% k = 1
*resume;
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 2
%% program continued
%% testpoint at line SCHLEIFE.7
*sleep 6; resume;
%% program continued
%% testpoint at line SCHLEIFE.7
*at 6 do display ('Wird erst nach AWAKE 6 ausgegeben');
*resume;
%% program continued
%% testpoint at line SCHLEIFE.7
*awake 6; resume;
%% program continued
%% testpoint at line SCHLEIFE.6
%% i = 5
%% 'Wird erst nach AWAKE 6 ausgegeben'
%% program continued
%% testpoint at line SCHLEIFE.7
*
```

Format 2:

```
AWAKE
```

Das AWAKE-Kommando ohne Testpunkt-Spezifikation bewirkt, daß der Testpunkt, an dem das AWAKE-Kommando ausgeführt wird, aktiviert wird.
Ist dieser Testpunkt der Anfangs-Testpunkt (siehe 9.1.4.1), so wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.
Wenn das Kommando am Post-Mortem- oder Exception-Testpunkt (siehe 9.1.4.3 und 9.1.4.4) ausgeführt wird, so wirkt es nicht auf diesen Testpunkt. Befindet sich aber an der Stelle, an der der Fehler aufgetreten ist, (zufällig) ein (mit SLEEP deaktivierter) Testpunkt, so wird dieser wieder aktiviert. Wenn dort kein Testpunkt gesetzt ist, wird das Kommando mit einer Fehlermeldung (siehe 9.3) abgewiesen.

Beispiel für das Zusammenspiel der Testpunkt-Kommandos

```
*at schleife.6 do getcmd;
*at schleife.7 do display (’erstes AT auf 7’);
*at schleife.9 do display (’Programmende’);
*resume; display (i);                                              (01)
%% program continued
%% testpoint at line SCHLEIFE.6
*resume;
%% program continued
%% testpoint at line SCHLEIFE.7
%% ’erstes AT auf 7’
%% program continued
%% testpoint at line SCHLEIFE.6
*at 7 do begin display (i); getcmd; end;                           (02)
*resume;
%% program continued
%% testpoint at line SCHLEIFE.7
%% ’erstes AT auf 7’
%% i = 2
*resume;
%% program continued
%% testpoint at line SCHLEIFE.6
*sleep 7;
*at 7 do display (’drittes AT auf 7’);                             (03)
*resume;
%% program continued
%% testpoint at line SCHLEIFE.6                                    (04)
*resume;
%% program continued
%% testpoint at line SCHLEIFE.6
*awake 7; resume;
%% program continued
%% testpoint at line SCHLEIFE.7
%% ’erstes AT auf 7’
%% i = 5
*resume;                                                           (05)
%% ’drittes AT auf 7’
%% program continued
%% testpoint at line SCHLEIFE.9
%% ’Programmende’
*remove 7
*
```

(01) Nach dem RESUME-Kommando wird der Rest der Eingabezeile gelöscht, das Kommando display (i) also nicht mehr ausgeführt.

(02) Erweitert das für Zeile 7 gültige DISPLAY-Kommando um das Verbund-Kommando.

(03) Wird erst nach AWAKE 7 aktiv.

(04) Wegen SLEEP 7 wird der Programmlauf in Zeile 7 nicht unterbrochen.

(05) Rückkehr an die durch GETCMD unterbrochene Stelle; d.h. Ausführung des danach befindlichen mit dem dritten AT-Kommando gesetzten DISPLAY-Kommandos.

Aktions-Kommandos

Die Aktions-Kommandos dienen zur Ausgabe (DISPLAY- und DUMP-Kommando) und Änderung (ASSIGN-Kommando) von Programmdaten, zur bedingten Ausführung von Testhilfe-Kommandos (IF-Kommando), zur Zusammenfassung mehrerer Testhilfe-Kommandos zu einem Kommando (Verbund-Kommando), zur Ausführung von System-Kommandos (SYSTEM-Kommando), zum Aufruf des Editors (EDIT-Kommando) und zur Ausgabe von Zustandsinformationen (SHOW-Kommando).

DISPLAY-Kommando

```
DISPLAY (liste)
```

Das DISPLAY-Kommando bewirkt die (typengerechte) Ausgabe der in der Liste angegebenen, durch Komma getrennten Faktoren (siehe 9.1.3) und "Variablen gruppen" auf das PATH-Ausgabe-Medium.

Es gibt 2 Arten von Variablengruppen:

- %LOCAL umfaßt die Variablen (keine Parameter) eines Blocks (siehe 9.1.2). Dieser Block wird durch die %LOCAL vorangestellte Block- oder Ausführungs-Qualifikation (siehe 9.1.3) bestimmt.

  Wenn keine Block- oder Ausführungs-Qualifikation angegeben ist, so wird als Block derjenige angenommen, in dem der Testpunkt liegt, an dem das Display-Kommando ausgeführt wird.

  Ist der Block ein Paket, so bezieht sich %LOCAL auf die unmittelbar in der Paket-Spezifikation und Paket-Implementierung deklarierten Variablen, nicht aber auf etwaige durch USE-Klauseln sichtbar gemachte Variablen anderer Pakete.

  Enthält der Block keine Variablen, so wird eine entsprechende Meldung ausgegeben.

  Beispiel

  ```
  %% testpoint at line p.333
  *display (%local);
  %% variables of p:
  %%    strg = 'ab'
  %%    a_ch = 'a'
  %%    z_ch = 'z'
  %%    r
  %%       .i = 1
  %%       .j = 10
  *
  ```

  Es werden die im aktuellen Block, also im Paket p (Spezifikation und Implementierung) deklarierten Variablen ausgegeben.

```
*display (a.b.c.%local);
%% variables of a.b.c:
%%    i = 5
%%    ch = 'x'
```

Es werden die lokalen Variablen des durch die Block-Qualifikation a.b.c. bestimmten Unterprogramms ausgegeben.

```
*at 15, unity.(11, 30) do display (%local); resume
%% program continued
%% testpoint at line unitx.15 (proc12)
%% variables of proc12:
%%    b = True
%%    k = 10
%% program continued
%% testpoint at line unity.11 (proc21)
%% variables of proc21:
%%    i = 8
%% program continued
%% testpoint at line unity.30 (func22)
%% variables of func22: none
%% program continued
```

Es werden jeweils die lokalen Variablen jenes Blocks ausgegeben, in dem der jeweils gerade aktuelle (erreichte) Testpunkt liegt.

- %PARAM umfaßt die Parameter eines Blocks. Ist der Block kein Unterprogramm oder ein parameterloses Unterprogramm, so wird eine entsprechende Meldung ausgegeben.

  Beispiel

```
*display  (a.b.%2.%param);
%% parameters of a.b.%2:
%%    x = 1
%%    b = False

*at unity. (46, 67) do display (%param); resume
%% program continued
%% testpoint at line unity.46 (proc22)
%% parameters of proc22:
%%    m = 5
%%    n = 5
%% program continued
%% testpoint at line unity.67 (proc23)
%% parameters of proc23: none
%% program continued
```

Durch Nachstellen von :HEX können Variable und Konstante sedezimal ausgegeben werden. Bei der sedezimalen Ausgabe von strukturierten Werten werden die einzelnen Komponenten bzw. Elemente sedezimal ausgegeben.

Beispiel für die Anwendung des DISPLAY-Kommandos

```
 1    {$DEBUG} PROGRAM G (input, output);
 2    TYPE
 3       farb_typ = (schwarz, braun, rot, orange, gelb,
 4                   grün, blau, violett, grau, weiß);
 5       farben = set of farb_typ;
 6    VAR
 7       i : integer;
 8       b : boolean:
 9       ch: string [3];
10       s : farben;
11       f : farb_typ;
12
13    PROCEDURE gerade (var v_b: boolean;
14                          w_i: integer);
15    begin
16       if (w_i mod 2) = 0
17       then v_b := true
18       else v_b := false;
19    end;
20
21    begin
22       s := [grün, grau, gelb, blau];
23       f := weiß;
24       readln;
25       read (i);
26       gerade (b, i);
27       if b
28       then ch := 'yes';
29       else ch := 'no ';
30       writeln (ch);
31    end.
```

Nach dem Starten des Testprogramms:

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at g.19, g.31 do getcmd; resume;
%% program continued
*9 ———————————————————————————————————————————————————————————————— (01)
%% testpoint at line G.19 (gerade)
*display (v_b, w_i);
%% v_b = False
%% w_i = 9
*display (gerade.%local); ———————————————————————————————————————— (02)
%% variables of gerade: none
*display (gerade.%param); ———————————————————————————————————————— (03)
%% parameters of gerade:
%% v_b = False
%% w_i = 9
```

(01) Eingabe für das Programm.

(02) Ausgabe der lokalen Variablen (falls vorhanden) der Prozedur gerade.

(03) Ausgabe aller Parameter der Prozedur gerade.

```
*dump; ------------------------------------------------------------------ (04)
%% parameters and local variables of gerade:
%%    v_b = False
%%    w_i = 9
%% global variables of G:
%%    i = 9
%%    b = False
%%    ch = ''
%%    s = [gelb..blau, grau]
%%    f = weiss
*display (i, i:hex);
%% i = 9
%% i = #9
*resume;
%% program continued
no ----------------------------------------------------------------------- (05)
%% testpoint at line G.31
*display (%local);
%% variables of G:
%%    i = 9
%%    b = False
%%    ch = 'no '
%%    s = [gelb..blau, grau]
%%    f = weiss
*display (ch[1], ch[1]:hex);
%% ch[1] = 'n'
%% ch[1] = #'95'
*display (ch[2], ch[3]);
%% ch[2] = 'o'
%% ch[3] = ' '
*display (ch, ch:hex);
%% ch = 'no '
%% ch = #'959640'
*display ('string');
%% 'string'
*display (2, 'A', 3.14, true);
%% 2
%% 'A'
%% 3.1400000000000E+00
%% True = True
*display (farben ([weiß, braun.. orange, rot.. gelb])); ------------------ (06)
%% farben ([weiß, braun..orange, rot..gelb]) = [braun..gelb, weiß]
```

(04) Ausgabe aller Variablen der dynamischen Aufrufkette und der globalen Variablen aller Pakete (siehe 9.2.2.8).

(05) Vom Programm produzierte Ausgabe.

(06) Ausgabe eines qualifizierten Mengenbildners.

ASSIGN-Kommando

```
ASSIGN variable := faktor
```

Der Variablen wird der Wert des Faktors (siehe 9.1.3) zugewiesen. Wenn dieser Wert die durch den Typ der Variablen festgesetzten Grenzen überschreitet, wird die Zuweisung mit einer Fehlermeldung abgelehnt.

Das ASSIGN-Kommando ist nur dann zulässig, wenn die Übersetzungseinheit, in der der Testpunkt liegt, mit der Option DEBUG=ON übersetzt wurde, nicht hingegen bei DEBUG=RESTRICTED (siehe 9.1.7).

Beispiel

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at g.19 do getcmd; resume;
%% program continued
*12
%% testpoint at line G.19 (gerade)
*display (i, w_i);
%% i = 12
%% w_i = 12
*assign w_i := 5; ------------------------------------------------------------ (01)
*display (i, w_i);
%% i = 12
%% w_i = 5
*display (b, v_b);
%% b = True
%% v_b = True
*assign v_b := false; -------------------------------------------------------- (02)
*display (b, v_b);
%% b = False
%% v_b = False
*assign s := farben ([grau, rot.. grün, gelb.. blau]); ------------------- (03)
*display (s);
%% s = [rot..blau, grau]
*assign s := farben ([]);
*display (s);
%% s = []
*
```

(01) Durch die Zuweisung auf w_i wird die als Value-Parameter übergebene Variable i nicht geändert.

(02) Durch die Zuweisung auf v_b wird die als Var-Parameter übergebene Variable b geändert.

(03) Der Variablen s wird ein qualifizierter Mengenbildner zugewiesen.

IF-Kommando

Format 1:

```
IF bedingung THEN kommando
```

Format 2:

```
IF bedingung THEN kommando ELSE kommando
```

Dieses Kommando entspricht der IF-Anweisung in Pascal-XT.

Als Bedingung sind BOOLEAN-Variable, BOOLEAN-Konstante sowie Ausdrücke der Art

```
Faktor    Vergleichsoperator    Faktor
```

zulässig.

Als Vergleichsoperator sind folgende Operatoren erlaubt:
"=", "<>", "<=", ">=", "<", ">" und "in".
Als Kommandos im THEN- und im wahlweisen ELSE-Zweig sind sämtliche Testhilfe-Kommandos zulässig.

Beispiel

```
%% testpoint before program start
%% scope seen from outside all compilation units
*at g.26 do if i < 0 then getcmd;
*at g.27 do if blau in s then begin
*              if b then display ('gerade')
*                   else display ('ungerade')
*              end
*           else display ('blau nicht in s');
*resume;
%% program continued
*-6
%% testpoint at line G.26
*assign i := 3;
*if f in farben ([gelb.. blau]) then display (f); resume;
%% program continued
%% testpoint at line G.27
%% 'ungerade'
%% program continued
```

Verbund-Kommando

```
BEGIN kommandofolge END
```

Dieses Kommando entspricht der Verbundanweisung in Pascal-XT.

Man kann es dazu verwenden, um in einem AT- oder IF-Kommando mehrere Kommandos zusammenzufassen oder um die Ausführung einer Folge von Kommandos bis zum abschließenden END zu verzögern.

Falls in einem inneren Kommando eines Verbund-Kommandos ein Analysefehler auftritt, so muß das gesamte Verbund-Kommando neu eingegeben werden.

Beispiel

```
%% testpoint before program start
%% scope seen from outside all compilation units
*begin display (1); ------------------------------------------------------ (01)
*      display (2); end;
%% 1
%% 2
*at g.22 do begin display (i); getcmd;
*                display ('hallo !!!') end; resume;
%% program continued
%% testpoint at line G.22
%% i = 0
*resume; ----------------------------------------------------------------- (02)
%% 'hallo !!!'
%% program continued
*
```

(01) Das DISPLAY-Kommando wird erst ausgeführt, wenn das Verbund-Kommando mit END abgeschlossen ist.

(02) Das RESUME-Kommando bezieht sich auf das GETCMD-Kommando im Verbund-Kommando.

SYSTEM-Kommando

Format 1:

```
SYSTEM 'system-kommando'
```

Es wird das BS2000-Kommando 'system-kommando' ausgeführt und dann wieder in die Testhilfe zurückgekehrt. In 'system-kommando' können Groß- und Kleinbuchstaben angegeben werden.

Beispiel

```
%% testpoint at line G.22
*system 'fstat ,r';
% PUBLIC SPACE:       13212 PAGES FOR    206 FILES
*
```

Format 2:

```
SYSTEM
```

Wirkt wie die Break-Taste (K2). Nach Eingabe des Kommandos /RESUME wird in die Testhilfe zurückgekehrt.

Beispiel

```
%%testpoint at line G.22
*system;
BKPT PCOUNT 00722E
/FSTAT ,R
% PUBLIC SPACE:       13212 PAGES FOR    206 FILES
/RESUME
*
```

EDIT-Kommando

Format 1:

```
EDIT 'datei-name'
```

Der eingestellte Prozedurbereich wird gelöscht und ein EDT File-Kommando mit dem angegebenen Dateinamen abgesetzt. Anschließend wird die Datei in den Prozedurbereich geladen und in den EDT verzweigt. Nach Eingabe von '@RET' oder 'HALT', oder Drücken der K1-Taste wird wieder in die Testhilfe zurückgekehrt. In 'datei-name' können Groß- und Kleinbuchstaben angegeben werden.

Format 2:

```
EDIT
```

Es wird in den zuletzt eingestellten Prozedurbereich des EDT verzweigt. Beim ersten Aufruf ist dies der Prozedurbereich 0. Der EDT kann, wie oben beschrieben, verlassen werden.

SHOW-Kommando

Format 1:

```
SHOW WHERE testpunkt-spezifikation
```

Das SHOW WHERE-Kommando mit Testpunkt-Spezifikation (siehe 9.1.3, 9.1.4.2 und 9.1.4.4) gibt aus, an welchen der angegebenen potentiellen Testpunkte (siehe 9.1.2) Testpunkte gesetzt sind.

Wenn unter den durch die Testpunkt-Spezifikation spezifizierten potentiellen Testpunkten kein einziger gesetzter Testpunkt gefunden wird, so erfolgt eine Fehlermeldung (siehe 9.3).

Beispiel

```
%% testpoint at line unitx.26
*show where 1..100, unity.(10..50, 63);
%% testpoints at:
%% line unitx.26
%% line unitx.35
%% line unitx.42
%% line unity.17
*show where unity.%all;
%% testpoints at:
%% line unity.17
%% line unity.98
*show where %all_units;
%% testpoints at:
%% line unitx.26
%% line unitx.35
%% line unitx.42
%% line unitx.378
%% line unity.17
%% line unity.98
%% line unitz.136
*show where exception;
%% testpoint at exception
*
```

Format 2:

```
SHOW WHERE
```

Das "leere" SHOW WHERE-Kommando (ohne Testpunkt-Spezifikation) erzwingt die Ausgabe der aktuellen Testpunkt-Meldung vor der nächsten PATH-Ein- oder -Ausgabe oder spätestens bei Verlassen des Testpunkts, auch wenn sie an diesem Testpunkt bereits ausgegeben wurde (siehe 9.3).

Dieses Kommando ist nützlich, wenn man vergessen hat, an welchem Testpunkt man sich befindet, oder (als deferred action) zur Verfolgung des Programm-Ablaufs.

Beispiel

```
*show where;
%% testpoint at line unitx.26
*at 27..40 do show where; at 42 do getcmd;
*resume;
%% testpoint at line unitx.27; program continued
%% testpoint at line unitx.28; program continued
%% testpoint at line unitx.38; program continued
%% testpoint at line unitx.38; program continued
%% testpoint at line unitx.42
*
```

Format 3:

```
SHOW UNITS
```

Das SHOW UNITS-Kommando liefert eine Liste aller Übersetzungseinheiten (Hauptprogramm und referierte Pakete) des Testlings. Für jede Übersetzungseinheit ist aus dieser Liste folgendes zu entnehmen:

- ihr voller Name, falls die entsprechende Testtabelle geladen ist, ihr gekürzter Name, falls die entsprechende Testtabelle (noch) nicht geladen ist.
- die Einstellung der Option DEBUG in der Spezifikation und der Implementierung, ob also "minimale", "partielle" oder "komplette" Testtabellen (siehe 9.1.7) erzeugt wurden und ob Zuweisungen an Programmvariable erlaubt sind.
- Information über die Verfügbarkeit der Testtabellen.
- Zeitpunkt der Übersetzung. Ab Version V2.2A werden bei der Datumsangabe für alte und neue Module Jahreszahlen vierstellig ausgegeben.

Beispiel

```
*show units
%% compilation-units:
%% INTERPRETATION, compiled 1991-12-03 13:05:27, complete testtable
%% ANALYZATION (restricted), compiled 1991-11-31 10:13:36, complete testtable
%% NEGOTIAT, compiled 1991-12-04 10:24:57, partial testtable ?
%% XYZ, compiled 1991-12-04 10:21:26, minimal testtable ?
%% CONNECTI, compiled 1991-11-31 12:10:04, invalid testtable
%% AB, compiled 1991-12-04 10:17:23, unavailable testtable
```

Die letzten vier Übersetzungseinheiten stehen in dieser Liste mit ihrem gekürzten Namen. Die erste und zweite Übersetzungseinheit (Paket-Implementierung) wurden mit der DEBUG-Option übersetzt; "?" bedeutet, daß PATH auf die Testtabellen dieser Übersetzungseinheiten noch nicht zugegriffen hat und die Verfügbarkeit dieser Testtabellen noch nicht untersucht hat.

Format 4:

```
SHOW CALLS
```

Das SHOW CALLS-Kommando liefert eine Liste der Glieder der dynamischen Aufrufkette des aktuellen Testpunkts, d.h. es werden alle Unterprogrammausführungen, beginnend mit derjenigen, aus der die den aktuellen Testpunkt enthaltene Unterprogrammausführung aufgerufen wurde, bis hin zum Hauptprogramm ausgegeben.

Beispiel

```
%% testpoint at line PROG.237 (proc7)
*show calls
%% called from line PROG.100  (proc6)
%% called from line PROG.80  (proc5)
%% called from line PROG.77  (proc5)
%% called from line PROG.77  (proc5)
%% called from line PROG.253  (func2)
%% called from line PROG.103  (proc6)
%% called from line PROG.323
```

Der Begriff der Unterprogrammausführung ist erforderlich, da ja aufgrund direkt oder indirekt rekursiver Unterprogrammaufrufe ein und dasselbe Unterprogramm mehrmals in der dynamischen Aufrufkette enthalten sein kann.

Die Ausgabe der Reihenfolge entspricht der umgekehrten Aufruf-Reihenfolge im Programm; es wird also die "aktuellste" Ausführung als erste ausgegeben.

Die Variablen der einzelnen Unterprogrammausführungen können durch Ausführungs-Qualifikationen (siehe 9.1.3) angesprochen werden.

Wenn also z.B. in einer Funktion F eine Variable I deklariert ist, so kann auf das I in der zweiten Ausführung von F mittels F.%2.I zugegriffen werden, wobei F.%2. die Ausführungs-Qualifikation ist.

Auch die Ausgabe aller lokalen Variablen bzw. Parameter einer Unterprogrammausführung ist möglich (siehe 9.2.2.1).

Hinweis

Vor dem Zugriff auf Daten der dynamischen Kette ist es günstig, mit Hilfe des SHOW CALLS-Kommandos die Liste der in der dynamischen Kette enthaltenen (Unter-)Programmausführungen auszugeben. Besonders wenn man einen Testpunkt (mittels RESUME) verlassen hat und auf einen anderen (oder sogar zufällig wieder auf den gleichen) Testpunkt aufläuft, sollte man sich, bevor man auf Daten der dynamischen Kette zugreift, wieder des SHOW CALLS-Kommandos bedienen, da sich ja die dynamische Kette seit dem letzten Testpunkt geändert haben kann.

DUMP-Kommando

```
DUMP
```

Ausgabe aller Variablen der dynamischen Aufrufkette und aller globalen Variablen aller Pakete auf das PATH-Ausgabe-Medium.

Beispiel

```
*dump
%% parameters and local variables of aaa:
%%    i = 17
%%    b = False
%% parameters and local variables of aa: none
%% parameters and local variables of a:
%%    ch = 'a'
%%    b = True
%%    x = L'2D1'
%% global variables of HAUPT:
%%    a = 200
%%    b = 100
%%    p = L'C2F11'
%% global variables of PACKA:
%%    j = 1
%%    rec_a
%%       .p = L'0'
%%       .q = L'A2F3'
%%       .size = 512373
%% global variables of PACKB: none
%% global variables of PACKC:
%%    c = 0
%%    d = 1
%%    tab
%%       [1] = 'eins'
%%       [2] = 'zwei'
%%       [3] = 'drei'
```

KILL-Kommando

```
KILL
```

Die Wirkung von KILL hängt davon ab, welcher Pascal-XT-Programmteil gerade aktiv ist. In einer Entry-Prozedur wird der Testling abgebrochen. Im Hauptprogramm wird der Testling nur abgebrochen, falls kein erneutes Testen (siehe 9.5 "Restart") mehr gewünscht wird.

Beispiel

```
*kill
%% program aborted
```

SWITCH-Kommando

Format 1:

```
SWITCH INPUT    'datei-name'
SWITCH OUTPUT   'datei-name'
SWITCH LIST     'datei-name'
```

Das PATH-Eingabe-, -Ausgabe- bzw. -List-Medium von PATH wird der angegebenen Datei zugeordnet.

Format 2:

```
SWITCH INPUT
SWITCH OUTPUT
SWITCH LIST
```

Durch das "leere" SWITCH-Kommando wird die aktuelle Datei-Zuordnung gelöst und die zuletzt gültige wieder hergestellt.

Hinweis

Dieses Kommando ist derzeit noch nicht verfügbar.

Testhilfe-Meldungen

Zur Analysezeit werden die vom PATH-Eingabe-Medium eingegebenen Kommandozeilen auf das PATH-Ausgabe-Medium wieder ausgegeben (Eingabewiederholung). Falls sie fehlerhaft (Syntax- oder Semantikfehler) sind, bricht die Analyse beim ersten Fehler ab und gibt eine entsprechende Fehlermeldung aus. Die Eingabewiederholung fehlerfreier Kommandozeilen kann durch die Option {$R-} (siehe 9.1.3.1) unterdrückt werden.

- Vor jeder PATH-Ein- oder -Ausgabe wird die aktuelle Testpunktmeldung ausgegeben, falls sie seit dem Erreichen des Testpunkts noch nicht ausgegeben worden ist oder falls nach ihrer (letzten) Ausgabe ein (leeres) SHOW WHERE-Kommando ausgeführt wurde.

  Beispiel

  ```
  %% testpoint at line unitx.72
  *display (i);
  %% i = 9
  *show where;
  %% testpoint at line unitx.72
  *
  ```

  Wenn an einem (benutzergesetzten) Testpunkt keine PATH-Ein- oder Ausgabe erfolgt (sondern z.B. nur ein ASSIGN-Kommando) und kein (leeres) SHOW WHERE-Kommando ausgeführt wird, so wird keine Testpunktmeldung ausgegeben.

  Je nach Art des Testpunkts hat die Testpunktmeldung folgendes Format:

  Anfangs-Testpunkt:

  ```
  testpoint before program start
  ```

  Benutzergesetzter Testpunkt:

  ```
  testpoint at line <unit-name>.<line-nr>
  ```

  Post-Mortem-Testpunkt:

  ```
  testpoint because of unhandled <error> at line <unit-name>.<line-nr>
  ```

  Exception-Testpunkt:

  ```
  testpoint because of <error> at line <unit-name>.<line-nr>
  ```

  Entry-Testpunkt:

  ```
  testpoint before entry call
  ```

  Beim Erreichen eines Testpunktes, an dem "globale Sichtbarkeit" (siehe 9.1.2) herrscht, wird die Meldung

  ```
  scope seen from outside all compilation units
  ```

  ausgegeben, insbesondere am Anfangs-Testpunkt (siehe 9.1.4.1) und (bei nicht kompletter oder nicht verfügbarer Testtabelle, siehe 9.1.7) am Post-Mortem-Testpunkt

(siehe 9.1.4.3) oder am Exception-Testpunkt (siehe 9.1.4.4).

Beispiele

Das folgende Beispiel zeigt die Meldungen am Anfangs-Testpunkt:

```
%% testpoint before program start
%% scope seen from outside all compilation units
*
```

Das folgende Beispiel zeigt die Meldungen am Post-Mortem-Testpunkt eines Programms mit minimaler oder partieller Testtabelle:

```
POINTER_ERROR ( 92) RAISED FROM PROG AT 000013C0
FILE_INFO IN THE MOMENT OF RAISE : NONE.
%% testpoint because of Pointer_Error at line PROG.16
%% scope seen from outside all compilation units
*
```

- Beim Verlassen eines Testpunkts wird eine Testpunkt-Ende-Meldung ausgegeben, wenn seit dem Erreichen des Testpunkts bereits (mindestens) einmal die Testpunktmeldung ausgegeben wurde. Aus der Testpunkt-Ende-Meldung "program continued" bzw. "program aborted" ist zu entnehmen, ob das getestete Programm fortgesetzt oder aufgrund der Ausführung eines KILL-Kommandos abgebrochen wird.

  Falls ein (leeres) SHOW WHERE-Kommando ausgeführt wird, dann aber bis zum Verlassen des Testpunkts keine PATH-Ein- oder -Ausgabe (somit auch keine Testpunktmeldung) erfolgt oder, falls ein KILL-Kommando ausgeführt wird und an dem Testpunkt noch keine Testpunktmeldung ausgegeben wurde, so wird eine kombinierte Testpunkt- und Testpunkt-Ende-Meldung ausgegeben.

- Bei Fehlern während der Ausführung eines PATH-Kommandos wird eine der folgenden Fehlermeldungen ausgegeben:

Fehlermeldungen bezüglich Testtabellen

`test-table not available`

Die zum Testen einer Übersetzungseinheit erforderliche Testtabelle ist nicht verfügbar.
Die Ursache ist implementierungsabhängig.

`test-table invalid or incompatible`

Die Testtabelle und das geladene Objekt dieser Übersetzungseinheit stammen nicht von derselben Übersetzung oder die Testtabelle wurde mit einer nicht zur verwendeten PATH-Version passenden Compiler-Version erzeugt.

Maßnahme:
Übersetzungseinheit mit dem passenden Compiler übersetzen und gegebenenfalls neu Binden.

`unit compiled without DEBUG-Option`

Die Übersetzungseinheit wurde mit DEBUG=OFF übersetzt. Daher können darin keine Testpunkte gesetzt werden und Variablen, die oder deren Typen darin deklariert sind, nicht angesprochen werden.

Maßnahme:
Übersetzungseinheit mit DEBUG=ON bzw. DEBUG=RESTRICTED übersetzen.

Fehler in Testpunktkommandos

`no such testpoint found`

In der Testpunkt-Spezifikation eines SLEEP-, AWAKE-, REMOVE- oder SHOW WHERE-Kommandos ist kein vorher mittels AT-Kommando gesetzter Testpunkt enthalten.

`testpoint cannot be set`

Ein in einem AT-Kommando angegebener Testpunkt kann nicht gesetzt werden. Die Ursache ist implementierungsabhängig (z.B. Maschinencode schreibgeschützt).

`no testable unit`

Es wurde versucht, mittels AT %ALL_UNITS... Testpunkte in allen testbaren Übersetzungseinheiten zu setzen. Es ist aber keine Übersetzungseinheit testbar, da das Hauptprogramm und alle referierten Pakete mit DEBUG=OFF übersetzt wurden oder die Testtabellen nicht verfügbar sind.

Maßnahme:
Die Übersetzungseinheiten, in denen man testen will, mit DEBUG=ON bzw. DEBUG=RESTRICTED übersetzen.

Fehler in Aktionskommandos

`no call chain`

Ein SHOW CALLS-Kommando konnte nicht ausgeführt werden, da der aktuelle Testpunkt entweder der Anfangstestpunkt, der Post-Mortem-Testpunkt, oder ein Testpunkt im Hauptprogramm ist, und daher keine dynamische Aufrufkette existiert.

`no valid scope given`

Es herrscht globale Sichtbarkeit.

Maßnahme:
Der Variablengruppe (%LOCAL bzw. %PARAM) muß in diesem Fall eine Block-Qualifikation oder Ausführungs-Qualifikation (siehe 9.1.3) vorangestellt werden.

`no such incarnation found`

Die angegebene Ausführung (Inkarnation) des Unterprogramms existiert nicht.

Maßnahme:
Dynamische Aufrufkette mit dem Kommando SHOW CALLS ausgeben und gegebenenfalls die richtige Ausführungsnummer verwenden.

`invalid address`

Es wurde versucht, einen Pointer zu dereferenzieren, der eine ungültige Adresse enthält.

`invalid set`

In einem Aktionskommando wurde eine Variable von einem Mengen-Typ verwendet, die ungültige Werte enthält. Das kann insbesonders dann der Fall sein, wenn die Variable (noch) nicht initialisiert wurde.
Falls es sich um ein IF- oder ASSIGN-Kommando handelt, so wird es nicht ausgeführt. Beim DISPLAY- oder DUMP-Kommando wird der Inhalt der entsprechenden Variablen (einschließlich der ungültigen Werte) ausgegeben.

`invalid string length`

Die Länge eines Strings liegt außerhalb des zulässigen Bereichs. PATH kann daher auf diesen String nicht zugreifen.

`index out of range`

Der Wert des angegebenen Index-Ausdrucks liegt außerhalb des durch den ARRAY-Indextyp bestimmten Wertebereichs.

`low bound > high bound`

Der Wert der Slice-Untergrenze ist größer als der Wert der Slice-Obergrenze.

`pointer is nil`

Es wurde versucht, einen Pointer zu dereferenzieren, der den Wert Nil besitzt.

`out of range`

Der Wert einer Variablen liegt außerhalb des durch ihren Ordinaltyp bestimmten Wertebereichs.

`boolean variable is neither True nor False`

Die in einem IF-Kommando als Bedingung angegebene Variable besitzt einen ungültigen Wert. Es wird daher weder der THEN-Zweig, noch ein ELSE-Zweig ausgeführt.

`value to be assigned is out of bounds`

Es wurde versucht, einer Variablen einen Wert zuzuweisen, der außerhalb des durch ihren Ordinaltyp bestimmten Wertebereichs liegt. Die Variable wurde nicht verändert.

`set-value to be assigned exceeds bounds of variable`

Es wurde versucht, einer Mengen-Variablen einen Wert zuzuweisen, der Elemente enthält, welche außerhalb des Wertebereichs liegen, der durch den Basistyp des Typs der Variablen bestimmt wird.
Die Variable wurde nicht verändert.

`command not executed`

Das im Kommandostring eines SYSTEM-Kommandos angegebene Betriebssystem-Kommando wurde nicht ausgeführt, da es fehlerhaft ist.

`illegal parameter`

Die in einem EDIT-Kommando angegebene Datei kann nicht editiert werden.

`set-element is out of set-bounds`

Ein Element der durch den qualifizierten Mengenbildner angegebenen Menge liegt außerhalb des durch ihren Basistyp bestimmten Wertebereichs.

Ferner können in Ausnahmefällen Meldungen auftreten, die mit "DP:" beginnen:

Entweder liegt eine schwere Inkonsistenz im Testling vor (z.B. Änderung einer Paket-Spezifikation ohne Neuübersetzung betroffener Paket-Implementierungen) oder, falls dies nicht der Fall ist, ein interner Fehler im Pascal-XT-System.

Binden mit PATH

Ein Programm muß zum Testen mit der Testhilfe PATH gebunden werden. Die zum Binden benötigten Module der Testhilfe sind in der Bindemodulbibliothek $PASLIB-XT enthalten.
Das Testen von Programmen im Shared Code ist nicht möglich.

Statisches Binden

Beim statischen Binden mit dem TSOSLNK wird durch Einbinden des Moduls "#test" festgelegt, daß das Programm mit der Testhilfe PATH getestet werden soll. Die Testhilfe und die Testtabellenmodule werden dynamisch nachgeladen, sofern sie nicht statisch eingebunden werden.
Weitere Informationen zum Binden sind dem Abschnitt 6.2 zu entnehmen.

```
/EXEC $TSOSLNK
    PROGRAM  prog
    INCLUDE  progname           , modlib
    INCLUDE  #test              , $PASLIB-XT   ———————————— (01)
    INCLUDE  (#path##c,#path##d), $PASLIB-XT   ———————————— (02)
    INCLUDE  test-tabelle       , modlib       ———————————— (03)
    RESOLVE                     , modlib
    RESOLVE                     , $PASLIB-XT
    END
```

(01) Durch Einbinden des Moduls "#test" wird festgelegt, daß das erzeugte Objektprogramm unter Kontrolle der Testhilfe PATH ablaufen soll.

(02) Die Module der Testhilfe PATH werden eingebunden. Fehlt die INCLUDE-Anweisung, dann wird die Testhilfe beim Starten des Testlings dynamisch nachgeladen.

(03) Das angegebene Testtabellenmodul des Programms wird explizit eingebunden. Fehlt diese INCLUDE Anweisung, dann wird das Testtabellenmodul bei Bedarf dynamisch nachgeladen. Dasselbe gilt für die Testtabellenmodule der zu dem Programm gebundenen Pakete. Die Namensgebung der Testtabellenmodule ist dem Abschnitt 4.4 zu entnehmen.

Dynamisches Binden

Dynamisch gebundene Programme können nur innerhalb des Programmiersystems getestet werden. Dies geschieht mit der RUN-PROGRAM-Anweisung mit dem Operanden DEBUG=YES. Die Testtabellenmodule werden bei Bedarf dynamisch nachgeladen.

Testen mit PATH

Der Testling kann auf BS2000 Kommandoebene und innerhalb des Programmiersystems aufgerufen und ausgeführt werden.

Aufruf auf BS2000-Kommandoebene

```
/EXEC test-programm
```

Aufruf im Programmiersystem

```
RUN-PROGRAM (modlib, progname), DEBUG=YES
```

Der mit "progname" bezeichnete Testling wird aus der Bindemodulbibliothek "modlib" geladen und gestartet. Bei fehlender Angabe der Bibliothek und des Programmnamens wird das zuletzt übersetzte Programm getestet.

PATH überprüft für das Haupt-Programm und alle beteiligten Pakete, daß jeweils Code- und Daten-Modul von derselben Übersetzung stammen. Wenn dies nicht der Fall ist, so werden Fehlermeldungen der Art

```
code-module incompatible with data-module <unit-name>
```

ausgegeben und der Testlauf wird abgebrochen.

Nach dem Laden und Starten des Programms meldet sich PATH am Anfangstestpunkt und erwartet eine Kommandoeingabe:

```
%% testpoint before program start
%% scope seen from outside all compilation units
*
```

Restart des Testlings

Wenn das getestete Pascal-XT-Hauptprogramm beendet wird (normal, durch KILL oder aufgrund eines unbehandelten Laufzeitfehlers) kann der Benutzer den Testling beenden oder erneut starten. Es erscheint die Meldung

```
%% do you want to restart program with same testpoints ? (y/n)
```

Nach Eingabe von "y" oder "Y" wird das Programm nochmals mit denselben Testpunkten gestartet, wobei ILCS weder beendet noch neu initialisiert wird.

Nach Eingabe von "n" oder "N" wird in den Arbeitsmodus zurückgekehrt, der vor dem Aufruf des Pascal-XT-Programms gültig war (Pascal-XT-Programmiersystem oder BS2000-Kommandomodus).

Bei einem Hauptprogramm, das in einer anderen Programmmiersprache implementiert ist, ist kein Restart des Testlings möglich.

Abbrechen von PATH-Kommandos

PATH bietet die Möglichkeit, ein gerade aktives Testhilfe-Kommando (falls dieses z.B. überraschend viele Ausgaben liefert) durch Drücken der Taste (K2) abzubrechen. Nach Eingabe des BS2000-Kommandos "/INTR" bricht die Testhilfe das PATH-Kommando ab, gibt (eine bereits begonnen Ausgabezeile und) die Meldung

"... command(s) cancelled by user"

aus und fordert neue Kommandos an, d. h. alle weiteren für diesen Testpunkt vorgesehenen Kommandos werden ignoriert.

Das beschriebene Verhalten gilt nur bei der Eingabe von (K2)/INTR an einem Testpunkt, also zeitlich zwischen der Testpunkt-Meldung und der Testpunkt-Ende-Meldung (siehe 9.3). Wenn (K2)/INTR hingegen zu einem anderen Zeitpunkt eingegeben wird (an dem der Testling sich nicht an einem Testpunkt befindet), so wird nicht PATH unterbrochen, sondern im Testling die Ausnahme Break_Error ausgelöst. Der Testling wird abgebrochen, wenn er keinen Ausnahmebehandler enthält, der diese Ausnahme behandelt.

Paketnamen

Sind die Testtabellen nicht statisch eingebunden, dann sind nach dem Starten des Programms vom Hauptprogramm und den evtl. eingebundenen Paketen nur die auf 8 Zeichen verkürzten Namen bekannt. Erst nach dem Laden der Testtabellen sind die Namen in voller Länge bekannt.

Dynamisches Laden der Testtabellen

Bei einem mit der RUN-Anweisung gestarteten Programm bzw. einem statisch gebundenen Programm (Phase), in dem die Testtabellen nicht eingebunden sind, werden die Testtabellen bei Bedarf dynamisch nachgeladen. Zum Nachladen geht die Testhilfe nach folgender Strategie vor:

- Bei einem mit der RUN-Anweisung gestarteten Programm versucht die Testhilfe die Testtabellen aus jener Bindemodulbibliothek nachzuladen, aus der das Programm geladen wurde.
- Bei statisch gebundenen Programmen versucht die Testhilfe die Testtabellen aus der Datei TASKLIB bzw. $TASKLIB nachzuladen.

Sind sie dort nicht enthalten, dann sucht der dynamische Bindelader DLL insbesondere noch in einer vom Benutzer mit dem SYSFILE-Kommando eingestellten Tasklib.

Wird eine Testtabelle nicht gefunden, dann verlangt die Testhilfe die Eingabe eines Bibliotheksnamens oder eines "*" für die temporäre EAM-Bindemoduldatei oder eines Leerstrings. Mit dem Leerstring wird das Nichtvorhandensein der Testtabelle angezeigt. Weitere Testtabellen versucht die Testhilfe dann ebenfalls aus der angegebenen Bibliothek nachzuladen, wobei bei Mißerfolgen jeweils erneut nach einer Bibliothek gefragt wird.

Wenn die Testhilfe die Testtabelle eines vordefinierten Pakets (z.B. BS2000CALLS) benötigt und nicht findet (z.B. `... MOD "BS2000CT" NOT FOUND ...`), so ist als Bibliotheksname $userid.PASLIB-XT einzugeben.

Gültigkeit der Testtabellen

Die Testhilfe überprüft alle benötigten Testtabellen auf ihre Gültigkeit, d.h. ob sie durch dieselbe Übersetzung erzeugt wurden wie die Code- und Daten-Module. Bei ungültigen Testtabellen meldet die Testhilfe bei statisch eingebundenen Testtabellen

```
statically linked testtable is invalid, trying to link dynamically
```

und bei dynamisch nachgeladenen Testtabellen

```
dynamically linked testtable is invalid
```

und verlangt die Eingabe eines Bibliotheknamens (siehe oben).

Ein-/Ausgabe von PATH

Ausgaben der Testhilfe sind mit dem Präfix "**%%**" versehen und werden auf die Systemausgabedatei **SYSOUT** ausgegeben. Kommandos an die Testhilfe werden von der Systemeingabedatei **SYSDTA** gelesen.

Ausgabe von Zeigerwerten

Werte von Zeigervariablen werden in der Form "**L'adresse'**" ausgegeben. "adresse" wird in Hexadezimaldarstellung angegeben.

Behandlung "nicht-abdruckbarer" Zeichen des EBCDIC-Zeichensatzes

Nicht-abdruckbare Zeichen (z.B. in einer Variable eines variablen Zeichenkettentyps) werden als Punkt ('.') ausgegeben. Es ist natürlich nach wie vor möglich, die sedezimale Darstellung einiger oder aller Zeichen der Variablen durch Anfügen von :HEX auszugeben.

Beispiel

Sei strg eine Variable vom variablen Zeichenkettentyp String.

```
*DISPLAY (strg);
%% strg = '..Pascal.'
```

Die ersten zwei Zeichen und das letzte Zeichen der Zeichenkette sind nicht abdruckbar (oder möglicherweise das abdruckbare Zeichen '.'). Falls es wichtig ist, zu wissen, welche nicht abdruckbaren Zeichen in strg enthalten sind, kann man sich den Wert von strg oder Teile davon sedezimal ausgeben lassen:

```
*DISPLAY (strg[1..2] :HEX);
%% strg [#1..#2] = #'1A2E'
*DISPLAY (strg[9] :HEX);
%% strg [#9] = #'80'
```

Verhalten bei einem Laufzeitfehler des Testlings

Wenn der Exception-Testpunkt gesetzt ist, meldet sich die Testhilfe in einem Pascal-XT-Programmteil entweder an der Stelle, wo der Fehler aufgetreten ist oder an der Aufrufstelle des fremdsprachigen Unterprogramms, das den Laufzeitfehler weitergereicht hat, und führt die vorher für den Fehlerfall angegebenen PATH-Kommandos aus.

Ist im Quellprogramm des getesteten Pascal-XT-Programmteils für den Laufzeitfehler ein Ausnahmebehandler definiert, wird dort der Testling nach dem Verlassen des Exception-Testpunkts fortgesetzt.

Ist kein Ausnahmebehandler definiert, wird ein Post-Mortem-Testpunkt ausgelöst und die Testhilfe meldet sich, wie unter 9.1.4.3 beschrieben, entweder an der Stelle des Pascal-XT-Programmteils, an der der Laufzeitfehler auftrat oder an der Aufrufstelle des fremdsprachigen Unterprogramms, das den Fehler weitergereicht hat.

Die Testpunktmeldung enthält die Nummer der Quellzeile, in welcher der Laufzeitfehler auftrat. Danach können Testhilfekommandos eingegeben werden.

Das PATH-Kommando KILL führt zum Programmabbruch.

Das PATH-Kommando RESUME zeigt je nach Art des getesteten Pascal-XT-Programmteils unterschiedliche Wirkung (siehe 9.1.4.3).

Vor Ausgabe der Testpunktmeldung wird bei nicht geladener Testtabelle der Name der Bindemodulbibliothek angefordert.

Gleichzeitige Verwendung der Testhilfe PATH und der Systemtesthilfe AID

Die gleichzeitige Benutzung der Testhilfen PATH und AID führt dann zu keinen Konflikten, wenn mit PATH das Pascal-Programm und mit AID eingebundene Assembler-Programme getestet werden. An PATH-Testpunkten können aber auch AID-Kommandos mit dem Kommando SYSTEM abgesetzt werden. Dabei ist allerdings zu beachten, daß bei-

spielsweise bei der Ausgabe von Registerinhalten durch AID nicht die Werte des Testlings, sondern die der Testhilfe ausgegeben werden.

Potentieller Testpunkt

Neben den meisten Anweisungs-Anfängen stellen auch das Ende eines Unterprogramms, eines Hauptprogramms bzw. einer Paket-Implementierung potentielle Testpunkte (siehe 9.1.2) dar. Auch das Auftreten jeder Ausnahme (siehe 9.1.4.4) oder der Aufruf eines Entry-Unterprogramms in einer Übersetzungseinheit, die der Testhilfe noch nicht bekannt ist (siehe 9.1.4.5), stellt einen potentiellen Testpunkt dar.

Laufzeitfehler und Fehlerbehandlung

Laufzeitfehler sind Fehler und Ausnahmen, die während des Ablaufs eines Programms auftreten. Sie können verschiedene Ursachen haben, z.B. kann ein Adressierungsfehler auftreten oder der Anwender kann durch die vordefinierte Prozedur Raise (siehe [1], 15.11) eine Ausnahme erzeugen.

Ein Laufzeitfehler wird entweder in dem Programmteil, in dem er festgestellt wird, durch einen Ausnahmebehandler (exception handler, siehe [1], 14.2) abgefangen oder an den aufrufenden Programmteil weitergereicht. Das Weiterreichen eines Laufzeitfehlers an den Rufer wird auch "Fehlerpropagierung" genannt.

Ab Pascal-XT V2.2A ist es durch ILCS (siehe 7.1) möglich, Laufzeitfehler auch über Sprachgrenzen hinweg zu propagieren, d.h. wenn Rufer und aufgerufenes Unterprogramm in verschiedenen Programmiersprachen implementiert sind. Die Fehlerpropagierung funktioniert in beide Richtungen, also von einer Pascal-XT-ENTRY-Prozedur an einen fremdsprachigen Rufer oder von einem fremdsprachigen Unterprogramm an einen Pascal-XT-Rufer. Da aber nicht alle Programmiersprachen die Fehlerpropagierung unterstützen, kann es bei Sprachverknüpfungen zu einem anderen Systemverhalten als erwartet kommen. Welche Mechanismen zur Fehlerbehandlung andere Programmiersprachen zur Verfügung stellen, ist in der jeweiligen Sprach- bzw. Compilerbeschreibung nachzulesen. Im folgenden wird die Fehlerbehandlung für Pascal-XT V2.2A beschrieben und auf mögliche Probleme hingewiesen.

STXIT-Ereignisse und ILCS

Das Konzept der Ausnahmebehandlung der Sprache Pascal-XT erlaubt es dem Anwender, mögliche Laufzeitfehler beim Ablauf eines Programms gezielt zu behandeln. Laufzeitfehler, die das Betriebssystem feststellt, werden in BS2000-Sprechweise Ereignisse genannt und in STXIT-Ereignisklassen (siehe [6]) eingeteilt.
Ab Pascal-XT V2.2A werden die folgenden drei Ereignisklassen nicht mehr wie bisher direkt vom Laufzeitsystem, sondern über den SEH (Standard-Event-Handler) bzw. den SSH (Standard-STXIT-Handler) von ILCS beim Betriebssystem angemeldet:

(a) Programmüberprüfung (PROCHK)

(b) Nicht behebbarer Programmfehler (ERROR)

Diese beiden Ereignisklassen werden über den SEH von ILCS beim Betriebssystem angemeldet. Tritt ein solches Ereignis ein, übergibt das Betriebssystem an ILCS das Unterbrechungsgewicht des Ereignisses, das ILCS dann als Ereigniscode verwendet.

(c) Mitteilung an das Programm (INTR)

Diese Ereignisklasse wird vom Pascal-XT-Laufzeitsystem während seiner Initialisierung über den SSH von ILCS beim Betriebssystem angemeldet. Vorausgesetzt, ein Pascal-XT-Programmteil ist aktiv, wird dieses Ereignis durch die Tastenkombination K2 /INTR ausgelöst und als Pascal-XT-Break_Error interpretiert.

Die Pascal-XT-Feherbehandlung und die Testhilfe PATH funktionieren nur, wenn der SEH von ILCS nicht durch einen fremdsprachigen Programmteil abgemeldet oder durch den SSH in denselben Ereignisklassen überlagert ist.

Fehlerbehandlung und Ausgaben im Fehlerfall

Behandlung von SEH-Ereignissen durch Pascal-XT

Ab Pascal-XT V2.2A werden Laufzeitfehler unabhängig von ihrer Herkunft grundsätzlich über ILCS an das Pascal-XT-Laufzeitsystem gemeldet. ILCS übergibt dabei einen Ereigniscode, an dem das Laufzeitsystem erkennt, um welchen Fehler es sich handelt. ILCS bestimmt den Ereigniscode aufgrund eines Werts (bei STXIT-Ereignissen das Unterbrechungsgewicht, bei Programmfehlern die Error_Number), den die Instanz an ILCS übergibt, die den Fehler zuerst feststellt.

Was sind SEH-Ereignisse?

SEH-Ereignisse sind Laufzeitfehler, die das Pascal-XT-Laufzeitsystem durch den SEH von ILCS gemeldet bekommt. Aufgrund des übergebenen Ereigniscodes unterscheidet das Laufzeitsystem zwischen SEH-STXIT-Ereignissen und SEH-NON-STXIT-Ereignissen.

Zu den SEH-STXIT-Ereignissen gehören die STXIT-Ereignisklassen PROCHK und ERROR, zu den SEH-NON-STXIT-Ereignissen gehören Laufzeitfehler, die in einem Pascal-XT-Programmteil (z.B. wegen Raise) oder in einem fremdsprachigen Programmteil auftreten.

Wie werden SEH-Ereignisse behandelt?

Wenn beim Ablauf eines Programms zum erstenmal ein Pascal-XT-Programmteil (Hauptprogramm oder ENTRY-Prozedur) die Kontrolle erhält, meldet das Pascal-XT-Laufzeitsystem bei ILCS für SEH-Ereignisse eine eigene Ereignisbehandlungsroutine an. Diese bewirkt, daß ILCS das Eintreten eines SEH-Ereignisses an das Laufzeitsystem meldet. Die Ereignisbehandlungsroutine bleibt angemeldet, wenn aus dem Pascal-XT-Programmteil ein fremdsprachiges Unterprogramm aufgerufen wird. Dadurch wird die Behandlung von Fehlern möglich, die aus dem fremdsprachigen Unterprogramm an den rufenden Pascal-XT-Programmteil weitergereicht werden.

Wenn ein SEH-Ereignis in einem Pascal-XT-Programmteil eintritt oder aus einem Unterprogramm weitergereicht wird, meldet ILCS dieses Ereignis mit dem Ereigniscode an das Pascal-XT-Laufzeitsystem, das dann folgende Aktionen durchführt:

(1) Bestimmen der Pascal-XT-Fehlernummer, die dem SEH-Ereignis entspricht.

Dem Ereigniscode entnimmt das Laufzeitsystem, ob es sich um ein SEH-STXIT-Ereignis oder ein SEH-NON-STXIT-Ereignis handelt.

Bei SEH-STXIT-Ereignissen wird aus dem Unterbrechungsgewicht der Pascal-XT-Systemfehlercode und implizit auch die Pascal-XT-Fehlernummer bestimmt. Es spielt keine Rolle, ob der Programmteil, in dem das STXIT-Ereignis aufgetreten ist,

in Pascal-XT oder in einer anderen Programmiersprache implementiert ist.

Bei SEH-NON-STXIT-Ereignissen, die in einem Pascal-XT-Programmteil aufgetreten sind, wird der Ereigniscode als Pascal-XT-Fehlernummer verwendet. SEH-NON-STXIT-Ereignisse, die in einem fremdsprachigen Unterprogramm auftreten und an einen Pascal-XT-Programmteil weitergereicht werden, werden als SYSTEM_ERROR (Fehlernummer -1) mit dem Systemfehlercode 5002 interpretiert.

Fehler, die in mathematischen Routinen auftreten und an einen Pascal-XT-Programmteil weitergereicht werden, werden auf Numeric_Error mit einem speziellen Systemfehlercode abgebildet.

(2) Suchen eines Ausnahmebehandlers im Pascal-XT-Programmteil.

Enthält eine Prozedur oder Funktion der dynamischen Aufrufkette des Pascal-XT-Programmteils einen Ausnahmebehandler, dann wird das Programm dort fortgesetzt, wo er definiert ist. Alle Programmteile, über die der Fehler hinwegpropagiert wurde, werden durch eine sog. Unterprogramm-Termination-Routine beendet.

Wenn kein Ausnahmebehandler definiert ist, hängt das weitere Verhalten davon ab, ob der durchsuchte Programmteil eine Pascal-XT-ENTRY-Prozedur oder das Hauptprogramm ist.

Ist es das Hauptprogramm, gibt das Laufzeitsystem die Fehlerursache und die dynamische Aufrufkette (siehe 10.2.4) aus. Anschließend wird das Programm beendet.

Ist es eine ENTRY-Prozedur, wird der Fehler an den Rufer weitergereicht. Wenn der Rufer ein Pascal-XT-Programmteil ist oder selbst direkt oder indirekt aus einem Pascal-XT-Programmteil aufgerufen wurde, werden wieder die Aktionen (1) und (2) durchgeführt.
Ist der Rufer ein fremdsprachiger Programmteil und wird der Fehler weder dort noch in irgendeinem darüberliegenden fremdsprachigen Programmteil behandelt, wird das Programm abgebrochen.

Wenn während der Aktionen (1) und (2) ein weiterer Laufzeitfehler ("Secondary_Error") auftritt, wird das gesamte Programm sofort abgebrochen.

Der Pascal-XT-Break_Error

Das STXIT-Ereignis INTR (ausgelöst durch die Tastenfolge (K2) /INTR) wird nur dann als Pascal-XT-Break_Error interpretiert und ggf. an den Rufer weitergereicht, wenn ein Pascal-XT-Programmteil aktiv ist. In diesem Fall wechselt die vom Pascal-XT-Laufzeitsystem angemeldete Benutzer-STXIT-Routine für das Ereignis INTR in den Prozeß des unterbrochenen Pascal-XT-Programmteils und löst ein SEH-NON-STXIT-Ereignis mit dem Ereigniscode "Break_Error" aus. Ist für dieses Ereignis ein Ausnahmebehandler definiert, wird das Programm dort fortgesetzt, ansonsten wird der Fehler an den Rufer weitergereicht.

Tritt das Ereignis INTR ein, während ein fremdsprachiger Programmteil aktiv ist, sucht der SSH nach anderen Benutzer-STXIT-Routinen für das Ereignis INTR. Wenn solche Routinen definiert sind, bekommen sie von SSH das Ereignis gemeldet. Ansonsten wird das Programm kommentarlos an der unterbrochenen Stelle fortgesetzt.

Sprachverknüpfungen zwischen Pascal-XT und Assembler

Da die Pascal-XT-Fehlerbehandlung und auch die Testhilfe PATH nur korrekt arbeiten, wenn der SEH nicht durch einen fremdsprachigen Programmteil abgemeldet oder durch den SSH überlagert wird, sind bei Sprachverknüpfungen mit Assembler folgende Hinweise zu beachten:

Ein Anwender kann bei Bedarf in Assembler-Unterprogrammen selbst STXIT-Ereignisse anmelden und behandeln (Siehe Handbuch ASSEMBH [15]). Wenn jedoch Assembler-Programmteile für die Ereignisklassen PROCHK und ERROR beim BS2000 eigene STXIT-Routinen anmelden, ist darauf zu achten, daß das Assembler-Programm die STXIT-Routinen so verläßt, daß das Pascal-XT-Laufzeitsystem die Fehler weiterhin korrekt gemeldet bekommt.

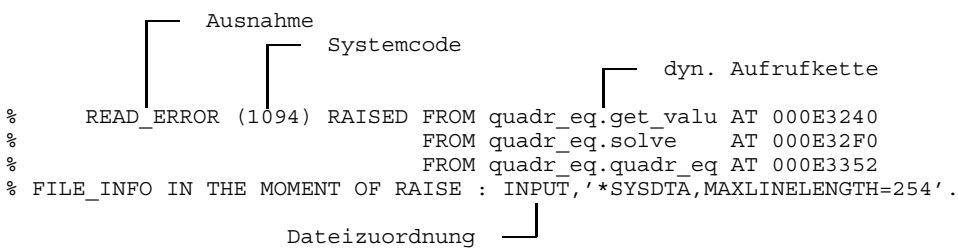
Ausgaben bei einem Laufzeitfehler

Beim Auftreten eines Laufzeitfehlers speichert das Pascal-XT-Laufzeitsystem die Fehlernummer und die Unterprogrammnamen der dynamischen Aufrufkette.

Die gespeicherten Informationen werden automatisch ausgegeben, wenn ein Laufzeitfehler im gesamten Programm nicht behandelt wird und wenn das Hauptprogramm in Pascal-XT implementiert ist. Tritt in Pascal-XT-ENTRY-Prozeduren ein Laufzeitfehler auf, werden die Informationen nicht ausgegeben. Der Anwender hat aber die Möglichkeit, einen Ausnahmebehandler zu definieren und dort mit der vordefinierten Prozedur Print_Error_Info des Pakets ERRORS (siehe A.7) die dynamische Aufrufkette auf die Systemdateien SYSOUT und SYSLST auszugeben.

Wird ein Laufzeitfehler an den Rufer weitergereicht, ist die unterschiedliche Wirkung von Raise (Error_Number) und Raise (0) (bzw. ERRORS.reraise, siehe A.7) zu beachten. Raise (Error_Number) löst dieselbe Ausnahme erneut aus und der ursprüngliche Fehlerort, d.h. die Unterprogramm-Aufrufkette von dort bis zu Raise, geht verloren.
Im Gegensatz dazu reicht Raise (0) die Ausnahme an den Rufer weiter, ohne die Unterprogramm-Aufrufkette zu zerstören (siehe auch die Beispiele am Ende des Abschnitts).

Die ausgegebenen Informationen haben folgenden Aufbau:



Bei Verknüpfungen verschiedener Sprachen werden noch weitere Informationen ausgegeben. Dies ist näher beim Aufbau der Dynamischen Aufrufkette (siehe unten) beschrieben.

Ausnahme

Die Ausnahme besteht aus einer ganzzahligen Nummer, wie sie von der vordefinierten Funktion Error_Number [1] geliefert wird. Handelt es sich um eine vordefinierte Ausnahme, dann gibt das Laufzeitsystem anstelle der Fehlernummer den von der Sprache vordefinierten Bezeichner aus. In der folgenden Tabelle sind diese Bezeichner und die zugehörigen Fehlernummern angegeben.

Systemcode

Unmittelbar hinter der Ausnahme wird in Klammern ein Systemcode zur näheren Klassifizierung der Ausnahme angegeben, wie ihn auch die Funktion System_Code des vordefinierten Pakets ERRORS (siehe A.3) liefert. Er ist nur für einige der vordefinierten Ausnahmen definiert (siehe Tabelle) und seine Bedeutung ist im Abschnitt 10.4 beschrieben.
Das Laufzeitsystem gibt die Fehlernummer als Hexadezimalzahl aus, wenn es sich um einen Systemcode des BS2000 handelt.
Der Systemcode hat den Wert 0, wenn die Ausnahme aufgrund der eingeschalteten Check-Option (siehe auch 10.4) ausgelöst wurde.

vordefinierte Ausnahme	ERROR_NUMBER	ERRORS.system_code
SYSTEM_ERROR	- 1	siehe 10.4
NUMERIC_ERROR	- 2	siehe 10.4
RANGE_ERROR	- 3	undefiniert
SET_ERROR	- 4	undefiniert
STRING_ERROR	- 5	siehe 10.4
INDEX_ERROR	- 6	undefiniert
POINTER_ERROR	- 7	siehe 10.4
VARIANT_ERROR	- 8	undefiniert
CASE_ERROR	- 9	undefiniert
FILE_ERROR	-10	siehe 10.4
EOF_ERROR	-11	undefiniert
OPEN_ERROR	-12	siehe 10.4
READ_ERROR	-13	siehe 10.4
MEMORY_ERROR	-14	siehe 10.4
BREAK_ERROR	-15	undefiniert [1)]
ELAB_ERROR	-16	undefiniert
	sonst	undefiniert

1) Ein BREAK_ERROR wird durch Drücken der Taste K2 (Übergang in den BS2000 Kommandomodus) und anschließender Eingabe des BS2000 Kommandos /INTR erzeugt.

Dynamische Aufrufkette

Die dynamische Aufrufkette gibt die Abfolge der Unterprogramm-Aufrufe wieder, die zu dem Zeitpunkt gültig ist, an dem ein Laufzeitfehler auftritt. Die Ausgabe der dynamischen Aufrufkette hat folgenden Aufbau:

Erste und zweite Zeile:
Wenn der Laufzeitfehler in einem Pascal-XT-Programmteil aufgetreten ist, enthält die erste Zeile den Namen der Pascal-XT-Prozedur, in der der Fehler aufgetreten ist.
Wenn der Fehler von einem fremdsprachigen Unterprogramm an den Pascal-XT-Programmteil weitergereicht wurde, enthält die erste Zeile folgenden Text:

```
Fehlernummer (Systemcode) RAISED FROM OUTSIDE PASCAL ENVIRONMENT
```

Die zweite Zeile enthält in diesem Fall den Namen der Pascal-XT-Prozedur, an die der Fehler als erstes weitergereicht wurde.

Folgezeilen:
Die Folgezeilen enthalten jeweils die dynamischen Pascal-XT-Vorgänger maximal bis zum Pascal-XT-Hauptprogramm. Für jede der angegebenen Prozeduren wird der Paketname und Prozedurname, jeweils mit maximal 8 Zeichen, bei ENTRY-Prozeduren zusätzlich mit dem Schlüsselwort ENTRY ausgegeben. Wenn zwischen einer Pascal-XT-Entry-Prozedur und einem Pascal-XT-Rufer ein fremdsprachiges Unterprogramm liegt, wird der Text ausgegeben:

```
FROM NON-PASCAL-ROUTINE(S)
```

Letzte Zeile:
Maximal werden die ersten 20 Zeilen der dynamischen Aufrufkette ausgegeben. Wenn es noch mehr dynamische Vorgänger gibt, wird dies in der letzten Zeile durch drei Punkte hinter dem Prozedurnamen angezeigt. Wenn nicht, enthält die letzte Zeile entweder den Namen des Pascal-XT-Hauptprogramms oder, wenn das Hauptprogramm in einer anderen Programmiersprache implementiert ist, den Namen der letzten Pascal-XT-ENTRY-Prozedur, die den Fehler weitergereicht hat.

Dateizuordnung

Tritt der Laufzeitfehler bei einem Dateizugriff auf, dann wird eine zusätzliche Zeile ausgegeben mit:

- Name der Pascal-Datei (maximal 8 Zeichen), sofern es sich um keine lokale Datei handelt.
- Wurde die aktuelle Datei mittels ASSIGNFILE zugewiesen, dann wird die externe Beschreibung aus diesem Aufruf ausgegeben.
- Bei nicht temporären Dateien wird der Name der aktuellen Datei ausgegeben.

Tritt ein Laufzeitfehler beim Testen (mit PATH) eines Programms auf, dann wird nach Ausgabe der Laufzeitfehlermeldung am Post-Mortem-Testpunkt die Nummer der Quell-

zeile ausgegeben, in der der Laufzeitfehler auftrat.

Secondary Error

Bei schwerwiegenden Fehlern gibt das Laufzeitsystem eine der folgenden Meldungen aus:

`SECONDARY ERROR ...` oder `INTERNAL_ERROR...`

Das heißt, daß eine Inkonsistenz im System entdeckt wurde. Zur Vermeidung von unendlichen Rekursionen gibt das Laufzeitsystem die obige Meldung aus und bricht das Programm ab. Der Grund für den Fehler oder die Inkonsistenz kann durch das Programm selbst verursacht worden sein (z.B. Überschreibungseffekte) oder es liegt ein noch nicht erkannter Fehler im System vor.

Beispiele

1. Unterschied zwischen Raise (Error_Number) und Raise (0)

Die unterschiedliche Wirkung von Raise (Error_Number) und Raise (0) wird anhand der beiden Beispiele 14-4 und 14-5 des Kapitels 14.3 der Sprachbeschreibung (siehe [1]) dargestellt.

Verhalten von Beispiel 14-4

```
 1      PROGRAM quadr_equation (Input, Output);
 2
 3      PROCEDURE get_value (name: String; VAR value: Real);
 4      BEGIN
 5         Writeln ('Bitte den Wert fuer ', name, ' eingeben:');
 6         Read (value);
 7      END;
 8
 9      PROCEDURE solve;
10      VAR
11        p, q, d: Real;
12      BEGIN
13         Writeln;
14         Writeln ('Quadratische Gleichung  x**2 + p*x + q = 0');
15         get_value ('p', p);
16         get_value ('q', q);
17         d := Sqrt (Sqr (p) / 4 - q);
18         IF d = 0
19            THEN Writeln ('x = ', -p / 2)
20            ELSE Writeln ('x = ', -p / 2 + d, ' oder ', -p / 2 - d);
21      EXCEPTION
22         IF Error_Number = Numeric_Error
23            THEN Writeln ('Die Gleichung hat keine Loesung')
24            ELSE Raise (Error_Number);
25      END;
26
27      BEGIN {PROGRAM quadr_equation}
28         WHILE True
29            DO solve;
30      EXCEPTION
31         IF Error_Number = Eof_Error
32            THEN Writeln ('Auf Wiedersehen')
33            ELSE Raise (Error_Number);
34      END.
```

```
Quadratische Gleichung  x**2 + p*x + q = 0
Bitte den Wert fuer p eingeben:
6
Bitte den Wert fuer q eingeben:
5
x = -1.000000000000000E+00 oder -5.000000000000000E+00

Quadratische Gleichung  x**2 + p*x + q = 0
Bitte den Wert fuer p eingeben:
6
```

```
Bitte den Wert fuer q eingeben:
9
x = -3.000000000000000E+00

Quadratische Gleichung  x**2 + p*x + q = 0
Bitte den Wert fuer p eingeben:
6
Bitte den Wert fuer q eingeben:
10
Die Gleichung hat keine Loesung

Quadratische Gleichung  x**2 + p*x + q = 0
Bitte den Wert fuer p eingeben:
xxx
READ_ERROR (1094) RAISED FROM quadr_eq.quadr_eq AT 000E3660(000E5000).
FILE_INFO IN THE MOMENT OF RAISE : INPUT,'*SYSDTA,MAXLINELENGTH=254'.
```

Wie man sieht, wird im Falle des durch die Eingabe von "xxx" hervorgerufenen (im Programm immer wieder durch "Raise (Error_Number)" propagierten) Read_Error die dynamische Aufrufkette nur ab der Stelle der letzten Propagierung ausgegeben (also nur Hauptprogramm "quadr_equation"). Es ist nicht mehr zu erkennen, daß der Fehler in der Prozedur "get_value" aufgetreten ist.

Verhalten von Beispiel 14-5

Das Programm wurde gegenüber dem in der Beschreibung etwas modifiziert: Statt des Aufrufs "Raise (0)" wird der äquivalente Aufruf "Errors.ReRaise" verwendet.

```
 1      WITH Errors;
 2      PROGRAM quadr_equation (Input, Output);
 3
 4      PROCEDURE get_value (name: String; VAR value: Real);
 5      BEGIN
 6         Writeln ('Bitte den Wert fuer ', name, ' eingeben:');
 7         Read (value);
 8      END;
 9
10      PROCEDURE solve;
11      VAR
12        p, q, d: Real;
13      BEGIN
14         Writeln;
15         Writeln ('Quadratische Gleichung  x**2 + p*x + q = 0');
16         get_value ('p', p);
17         get_value ('q', q);
18         d := Sqrt (Sqr (p) / 4 - q);
19         IF d = 0
20            THEN Writeln ('x = ', -p / 2)
21            ELSE Writeln ('x = ', -p / 2 + d, ' oder ', -p / 2 - d);
22      EXCEPTION
23         IF Error_Number = Numeric_Error
24            THEN Writeln ('Die Gleichung hat keine Loesung')
25            ELSE Errors.ReRaise;
26      END;
27
28      BEGIN {PROGRAM quadr_equation}
29         WHILE True
30            DO solve;
31      EXCEPTION
32         IF Error_Number = Eof_Error
33            THEN Writeln ('Auf Wiedersehen')
34            ELSE BEGIN
35               Writeln ('Programm-Abbruch, Details auf SYSLST');
36               Errors.ReRaise;
37               END;
35      END.

Quadratische Gleichung  x**2 + p*x + q = 0
Bitte den Wert fuer p eingeben:
xxx
Programm-Abbruch, Details auf SYSLST
%     READ_ERROR (1094) RAISED FROM quadr_eq.get_valu AT 000E3240
%                              FROM quadr_eq.solve    AT 000E32F0
%                              FROM quadr_eq.quadr_eq AT 000E3352
% FILE_INFO IN THE MOMENT OF RAISE : INPUT,'*SYSDTA,MAXLINELENGTH=254'.
```

Bei dieser Lösung wird die dynamische Aufrufkette ab der eigentlichen Fehlerstelle ausgegeben.

Dieselbe Fehlerausgabe kann man auch erreichen, indem man in Zeile 36 statt "ReRaise" die Prozedur "Errors.Print_Error_Info" aufruft und das Programm dann "normal" beendet.

2. Unbehandelter Laufzeitfehler in Pascal-XT/Cobol-Sprachverknüpfung

Dieses Beispiel entspricht Beispiel Nr. 1 aus Kapitel 7 mit einem Unterschied: Im Cobol-Unterprogramm tritt ein Laufzeitfehler auf (Numeric_Error wegen Division durch 0) und wird an den Rufer (Pascal-XT-Hauptprogramm) weitergereicht. Da der Fehler dort nicht behandelt wird, führt er zum Programmabbruch.

Quellcode des Pascal-XT-Hauptprogramms "PASHAUPT":

```
PROGRAM PASHAUPT (INPUT,OUTPUT);
TYPE
   RA   =   1..8;
   STR8 =   ARRAY [RA] OF CHAR;
   SATZ =   RECORD
            A(0)  :  INTEGER;
            B(4)  :  STR8;
            END;
CONST
   AGGR =   STR8('T','E','S','T',' ':4);
VAR
   RC   :   SATZ;
   I    :   INTEGER;
PROCEDURE COBUPROG (VAR PAR:SATZ); EXTERNAL;

BEGIN   { PASHAUPT }
    RC.A := 1111;
    RC.B := AGGR;
    COBUPROG (RC);
    WITH RC DO BEGIN ------------------------------------------------------ (01)
       WRITELN ('A:',A);
       WRITE ('B:');
       FOR I := FIRST(RA) TO LAST(RA) DO
          WRITE (B[I]);
       WRITELN;
       END;
END.
```

Quellcode des Cobol-Unterprogramms "COBUPROG":

```
       ID DIVISION.
       PROGRAM-ID. COBUPROG.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           TERMINAL IS SCREEN.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 TEILDR    PIC 9(8).
       77 DIV       PIC 9. ---------------------------------------------- (02)
```

```
LINKAGE SECTION.
01  SATZ.
    02  TEILB   PIC S9(8) COMP.
    02  TEILA   PIC X(8).
PROCEDURE DIVISION USING SATZ.
ANF.
    MOVE TEILB TO TEILDR.
    DISPLAY "TEILB: " TEILDR UPON SCREEN.
    DISPLAY "TEILA: " TEILA UPON SCREEN.
    MOVE "XXXXZZZZ" TO TEILA.
    MOVE 0 TO DIV.                                                    (03)
    COMPUTE TEILDR = TEILDR / DIV.                                    (04)
BACK.
    EXIT PROGRAM.
```

Aufruf der ablauffähigen Phase "PASCOBOL" und Ablaufprotokoll:

```
/EXEC PASCOBOL ──────────────────────────────────────────────────────────── (05)
%  BLS0500 PROGRAM 'PASCOBOL', VERSION ' ' OF '...' LOADED
TEILB: 00001111 ─────────────────────────────────────────────────────────── (06)
TEILA: TEST
9089 INTERRUPT-CODE= 68  AT PROGRAM COUNT= 000008F4 , PROGRAM-ID IS  COBUPROG
    NUMERIC_ERROR ( 104) RAISED FROM OUTSIDE PASCAL ENVIRONMENT ──────────── (07)
                              FROM PASHAUPT.PASHAUPT AT 0000027E ──────────── (08)
%  EXC0732 ABNORMAL PROGRAM TERMINATION. ERROR CODE 'NRT0101' /HELP-MSG NRT0101
%  CMD0205 ERROR IN PRECEDING COMMAND OR PROGRAM AND PROCEDURE STEP TERMINATION : COMMANDS ...
                    ... WILL BE IGNORED UNTIL /SET-JOB-STEP OR /LOGOFF OR /ABEND IS RECOGNIZED
```

Erläuterung:

(01) Ab hier wird das Pascal-XT-Hauptprogramm nicht mehr ausgeführt, weil im Cobol-Unterprogramm ein Laufzeitfehler auftritt, der zum Programmabbruch führt.

(02) Deklaration der Variablen DIV vom Typ PIC 9.

(03) Der Variablen DIV wird der Wert 0 zugewiesen.

(04) Die Anweisung, den Wert von TEILDR (1111) durch den Wert von DIV (0) zu dividieren, erzeugt einen Laufzeitfehler. Siehe (07).

(05) Starten der ablauffähigen Phase PASCOBOL.

(06) Ausgabe aus dem Cobol-Unterprogramm: Name und Wert der Variablen TEILB und TEILA.

(07) Weil im Cobol-Unterprogramm ein Laufzeitfehler aufgetreten ist und das Hauptprogramm diesen nicht behandelt, wird das Programm abgebrochen und die dynamische Aufrufkette ausgegeben. Die erste Zeile gibt an, daß es sich um einen Numeric_Error handelt und daß der Fehler aus einem fremdsprachigen Progammteil weitergereicht wurde.

(08) Diese Zeile gibt die Stelle des Pascal-XT-Programmteils aus, an der das fremdsprachige Unterprogramm, das den Laufzeitfehler weitergereicht hat, aufgerufen wurde. Vor dem Punkt steht ggf. der Name des Pakets und nach dem Punkt der Name der Entry-Prozedur. Ist der Pascal-XT-Programmteil wie hier ein Hauptprogramm, sind die beiden Namen vor und nach dem Punkt identisch.

3. Behandelter Laufzeitfehler in Pascal-XT/Cobol-Sprachverknüpfung

Dieses Beispiel entspricht Beispiel Nr. 2 mit einem Unterschied: Der Laufzeitfehler, den das Cobol-Unterprogramm an das Pascal-XT-Hauptprogramm weiterreicht, wird dort durch einen Exception-Handler abgefangen und führt deshalb nicht zum Programmabbruch. Da der Laufzeitfehler ein SEH-STXIT-Ereignis ist (Division durch 0), spielt es keine Rolle, daß er in einem fremdsprachigen Unterprogramm aufgetreten ist: Er wird als Numeric_Error weitergereicht, genau wie wenn er in einem Pascal-XT-Programmteil aufgetreten und dort nicht behandelt worden wäre.

Quellcode des Pascal-XT-Hauptprogramms "PASHAUPT":

```
WITH ERRORS;
PROGRAM PASHAUPT (INPUT,OUTPUT);
TYPE
   RA   =   1..8;
   STR8 =   ARRAY [RA] OF CHAR;
   SATZ =   RECORD
            A(0)  :  INTEGER;
            B(4)  :  STR8;
            END;
CONST
   AGGR =   STR8('T','E','S','T',' ':4);
VAR
   RC   :   SATZ;
   I    :   INTEGER;
PROCEDURE COBUPROG (VAR PAR:SATZ); EXTERNAL;

BEGIN  { PASHAUPT }
    RC.A := 1111;
    RC.B := AGGR;
    COBUPROG (RC);
    WITH RC DO BEGIN
       WRITELN ('A:',A);
       WRITE ('B:');
       FOR I := FIRST(RA) TO LAST(RA) DO
          WRITE (B[I]);
       WRITELN;
       END;
EXCEPTION
    IF ERROR_NUMBER = NUMERIC_ERROR ------------------------------------------ (01)
       THEN WRITELN ('FEHLER IM EXCEPTION-HANDLER ABGEFANGEN.')
       ELSE ERRORS.PRINT_ERROR_INFO;
END.
```

Quellcode des Cobol-Unterprogramms "COBUPROG" (unverändert):

```
       ID DIVISION.
       PROGRAM-ID. COBUPROG.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           TERMINAL IS SCREEN.
       DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
77 TEILDR    PIC 9(8).
77 DIV       PIC 9.
LINKAGE SECTION.
01  SATZ.
    02  TEILB   PIC S9(8) COMP.
    02  TEILA   PIC X(8).
PROCEDURE DIVISION USING SATZ.
ANF.
    MOVE TEILB TO TEILDR.
    DISPLAY "TEILB: " TEILDR UPON SCREEN.
    DISPLAY "TEILA: " TEILA UPON SCREEN.
    MOVE "XXXXZZZZ" TO TEILA.
    MOVE 0 TO DIV.
    COMPUTE TEILDR = TEILDR / DIV.
BACK.
    EXIT PROGRAM.
```

Aufruf der ablauffähigen Phase "PASCOB" und Ablaufprotokoll:

```
/EXEC PASCOB ------------------------------------------------------------ (02)
%  BLS0500 PROGRAM 'PASCOB', VERSION ' ' OF '...' LOADED
TEILB: 00001111 --------------------------------------------------------- (03)
TEILA: TEST
FEHLER IM EXCEPTION-HANDLER ABGEFANGEN. --------------------------------- (04)
```

Erläuterung:

(01) Im Exceptionhandler wird abgefragt, ob ein Numeric_Error aufgetreten ist bzw. von einem Unterprogramm weitergereicht wurde. Wenn ja - dies trifft hier zu -, wird eine Meldung ausgegeben. Siehe (04). Wenn nein, wird die dynamische Aufrufkette ausgegeben. Hierfür ist es notwendig, zu Beginn des Programms das Paket "Errors" zu inkludieren.

(02) Starten der ablauffähigen Phase "PASCOB".

(03) Ausgaben aus dem Cobol-Unterprogramm.

(04) Meldung des Exceptionhandlers aus dem Pascal-XT-Hauptprogramm.

4. Unbehandelter Laufzeitfehler in Pascal-XT/Fortran/Pascal-XT-Sprachverknüpfung

Ein Pascal-XT-Hauptprogramm ruft ein Fortran-Unterprogramm auf, das selbst eine Pascal-XT-Entry-Prozedur aufruft. Die Pascal-XT-Entry-Prozedur ruft zur Berechnung der Quadratwurzel die mathematische Funktion SQRT auf und übergibt einen negativen Wert. Da negative Werte als Übergabeparameter für diese Funktion nicht zulässig sind, tritt ein Laufzeitfehler (Numeric_Error) auf, der an den Rufer, d.i. die Pascal-XT-Entry-Prozedur, weitergereicht wird. Da der Fehler dort nicht behandelt wird, wird er an das Fortran-Unterprogramm und von dort an das Pascal-XT-Hauptprogramm weitergereicht. Auch im Pascal-XT-Hauptprogramm wird der Fehler nicht behandelt, weshalb er zum Programmabbruch führt.

Quellcode des Pascal-XT-Hauptprogramms "PASFOPAS":

```
PROGRAM PASFOPAS (INPUT, OUTPUT);
VAR
   PAR1    :   LONG_REAL;
PROCEDURE FOPAS (VAR PAR1 : LONG_REAL); EXTERNAL; ———————————— (01)
BEGIN
   PAR1 := -88.1;
   FOPAS (PAR1); ———————————————————————————————————————————— (02)
   WRITELN ('AUFRUF PROZEDUR FOPAS');
   WRITELN ('PAR1: ',PAR1);
END.
```

Quellcode des Fortran-Unterprogramms "FOPAS":

```
       SUBROUTINE FOPAS (VAR1)
       REAL*8  VAR1
       VAR1    = -1
       CALL PASPROZ (VAR1) ———————————————————————————————— (03)
       WRITE(2,100) VAR1 ———————————————————————————————— (04)
100    FORMAT (' VAR1: ',F6.2)
       RETURN
       END
```

Quellcode der Pascal-XT-Entry-Prozedur "PASPROZ" im Paket "PASERR":

Paketspezifikation:

```
PACKAGE PASERR;
ENTRY PROCEDURE PASPROZ (VAR F1 : LONG_INTEGER); ———————————— (05)
END.
```

Paketimplementierung:

```
PACKAGE BODY PASERR (OUTPUT);
PROCEDURE PASPROZ (VAR F1 : LONG_REAL);
VAR
   R  :  REAL;
BEGIN
   WRITELN ('F1: ',F1);
   R := SQRT(F1); ———————————————————————————————————————— (06)
END;
BEGIN
END.
```

Aufruf der ablauffähigen Phase "PAFOPAER" und Ablaufprotokoll:

```
/EXEC PAFOPAER
% BLS0500 PROGRAM 'PAFOPAER', VERSION ' ' OF '...' LOADED
BS2000  F O R 1 : IMPROVED MATHEMATICAL ACCURACY
F1: -1.0E+00 ———————————————————————————————————————————— (07)
    NUMERIC_ERROR (1004) RAISED FROM OUTSIDE PASCAL ENVIRONMENT———— (08)
                              FROM    PASERR.PASPROZ   AT 00000C16 ENTRY
     FROM NON-PASCAL-ROUTINE(S) FROM  PASFOPAS.PASFOPAS  AT 000002A0
%  EXC0732 ABNORMAL PROGRAM TERMINATION. ERROR CODE 'NRT0101' /HELP-MSG NRT0101
%  CMD0205 ERROR IN PRECEDING COMMAND OR PROGRAM AND PROCEDURE STEP TERMINATION : COMMANDS ...
                    ... WILL BE IGNORED UNTIL /SET-JOB-STEP OR /LOGOFF OR /ABEND IS RECOGNIZED
```

Erläuterung:

(01) Deklaration der fremdsprachigen Prozedur FOPAS.

(02) Aufruf des Unterprogramms FOPAS: die Variable PAR1 mit dem Wert -88.1 wird übergeben.

(03) Aufruf der Pascal-XT-Entry-Prozedur PASPROZ: die Variable VAR1 mit dem Wert -1 wird übergeben.

(04) Ab hier wird das Fortran-Unterprogramm nicht mehr ausgeführt, weil die Pascal-XT-Entry-Prozedur nicht ordnungsgemäß abgeschlossen wird, sondern von der mathematischen Funktion SQRT einen Laufzeitfehler gemeldet bekommt, den sie selbst an ihren Rufer weiterreicht.

(05) Deklaration der Pascal-XT-Entry-Prozedur PASPROZ in der Spezifikation des Pakets PASERR.

(06) Der Aufruf der Quadratwurzelfunktion SQRT mit einem negativen Wert erzeugt einen Laufzeitfehler, der an die rufende Pascal-XT-Entry-Prozedur weitergereicht wird.

(07) Ausgabe der Variablen F1 aus der Pascal-XT-Entry-Prozedur.

(08) Weil in der mathematischen Routine SQRT ein Laufzeitfehler aufgetreten ist und dieser weder in der aufrufenden Pascal-XT-Entry-Prozedur noch im Hauptprogramm behandelt wird, wird das Programm abgebrochen und die dynamische Aufrufkette ausgegeben.
Der ersten Zeile der dynamischen Aufrufkette ist zu entnehmen, daß der Fehler von einem fremdsprachigen Programmteil (hier von der mathematischen Routine SQRT) an einen Pascal-XT-Programmteil weitergereicht wurde.
Die zweite Zeile gibt den Pascal-XT-Programmteil (hier die Pascal-XT-Entry-Prozedur PASPROZ) an, der als erstes den Laufzeitfehler vom fremdsprachigen Programmteil gemeldet bekommt.
Die dritte Zeile gibt an, daß der Laufzeitfehler an ein oder mehrere fremdsprachige(s) Unterprogramm(e), deren Name jedoch nicht im einzelnen erscheint, und von dort an das Pascal-XT-Hauptprogramm PASFOPAS weitergereicht wurde.

Erkennung von Laufzeitfehlern

Für alle im Sprachmanual beschriebenen Fehler wird in den folgenden Tabellen angegeben, wann sie erkannt werden. Für jede Fehlerklasse sind im Abschnitt 10.4 die Systemfehlercodes angegeben, die zusätzliche Informationen zur Fehleranalyse liefern.

NUMERIC_ERROR

Fehler	Fehlererkennung
- In einem Ausdruck der Form x/y ist y = 0.	immer erkannt
- In einem Ausdruck der Form i DIV j ist j = 0.	immer erkannt
- In einem Ausdruck der Form i MOD j ist j <= 0.	j=0 immer erkannt j<0 bei Check=On
- Das Ergebnis einer arithmetischen Operation liegt nicht im Wertebereich des Ergebnistyps:	
- für Operanden vom Typ Short_Integer oder eines Teilbereichs davon ist dies der Typ Integer	immer erkannt
- für Operanden vom Typ Long_Integer oder eines Teilbereichs davon ist dies der Typ Long_Integer	immer erkannt
- für Operanden vom Typ Short_Real ist dies der Typ Short_Real	immer erkannt
- für Operanden vom Typ Long_Real ist dies der Typ Long_Real	immer erkannt
- Bei Abs(x) liegt das Funktionsergebnis nicht im Wertebereich des Ergebnistyps (Integer bzw. Long_Integer bzw. Short_Real bzw. Long_Real).	immer erkannt
- Bei Sqr(x) liegt das Funktionsergebnis nicht im Wertebereich des Ergebnistyps (Integer bzw. Long_Integer bzw. Short_Real bzw. Long_Real).	immer erkannt
- Das Ergebnis von Exp(x) liegt nicht im Wertebereich des Ergebnistyps (Short_Real bzw. Long_Real).	immer erkannt
- Bei Ln(x) ist x <= 0.	immer erkannt
- Bei Sqrt(x) ist x < 0.	immer erkannt
- Das Ergebnis von Trunc(x) oder Short_Trunc(x) oder Long_Trunc(x) liegt nicht im Wertebereich des Ergebnistyps (Integer bzw. Short_Integer bzw. Long_Integer).	bei Check=On

Fortsetzung nächste Seite

Fehler	Fehlererkennung
- Das Ergebnis von Round(x) oder Short_Round(x) oder Long_Round(x) liegt nicht im Wertebereich des Ergebnistyps (Integer bzw. Short_Integer bzw. Long_Integer).	bei Check=On

- In einem Ausdruck der Form x**n ist
  - x von einem Integer-Typ und n < 0, oder
  - x = 0 bzw. x = 0.0 und n <= 0.
	bei Check=On

- In einer Zuweisung liegt der Wert des Ausdrucks (rechte Seite) vom Typ Long_Real nicht im Wertebereich der Variablen bzw. des Funktionsbezeichners (linke Seite) vom Typ Short_Real.	kann nicht auftreten
- Bei Wertparameterübergabe liegt der Wert des Aktualparameters vom Typ Long_Real nicht im Wertebereich des Formalparameters vom Typ Short_Real.	kann nicht auftreten
- In einem Aggregat liegt der Wert eines Aggregat-Elements vom Typ Long_Real nicht im Wertebereich der zugehörigen Aggregatkomponente vom Typ Short_Real.	kann nicht auftreten
- Beim Lesen aus einer Nicht-Text-Datei mit Read(f,v) liegt der Wert der Puffervariablen f↑ vom Typ Long_Real nicht im Wertebereich der Variablen v vom Typ Short_Real.	kann nicht auftreten
- Beim Schreiben in eine Nicht-Text-Datei mit Write(f,a) liegt der Wert des Ausdrucks a vom Typ Long_Real nicht im Wertebereich der Puffervariablen f↑ vom Typ Short_Real.	kann nicht auftreten

RANGE_ERROR

Fehler	Fehlererkennung
- In einer Zuweisung liegt der Wert des Ausdrucks (rechte Seite) von einem Ordinal-Typ nicht im Wertebereich des Typs der Variablen bzw. des Funktionsbezeichners (linke Seite).	bei Check=On
- Bei Wertparameterübergabe liegt der Wert des Aktualparameters von einem Ordinal-Typ nicht im Wertebereich des Typs des Formalparameters.	bei Check=On
- In einem Aggregat liegt der Wert eines Aggregat-Elements von einem Ordinal-Typ nicht im Wertebereich des Typs der zugehörigen Aggregatkomponente.	bei Check=On
- Beim Lesen aus einer Nicht-Text-Datei mit Read(f,v) liegt der Wert der Puffervariablen f↑ von einem Ordinal-Typ nicht im Wertebereich des Typs der Variablen v.	bei Check=On
- Beim Schreiben in eine Nicht-Text-Datei mit Write(f,a) liegt der Wert des Ausdrucks a von einem Ordinal-Typ nicht im Wertebereich des Typs der Puffervariablen f↑.	bei Check=On
- Der Zeichenwert Chr(x) liegt nicht im Wertebereich des Typs Char.	bei Check=On
- Das Ergebnis von Succ(x) liegt nicht im Wertebereich des Typs von x.	bei Check=On
- Das Ergebnis von Pred(x) liegt nicht im Wertebereich des Typs von x.	bei Check=On
- Bei der Ausführung der Anweisung in einer FOR-Anweisung liegt der Anfangs- oder End-Wert der FOR-Anweisung nicht im Wertebereich des Typs der Laufvariablen.	bei Check=On
- Beim Lesen einer Integerzahl aus einer Textdatei (mit Read(f,x)) oder aus einem Zeichenkettenausdruck (mit Readstring(s,x)) liegt der Wert der Zahl nicht im Wertebereich der Variablen x und es ist kein Read_Error (s.u.) aufgetreten.	bei Check=On
- Bei Write(f,a:l1:l2) bzw. Writestring(s,a:l1:l2) ist die Gesamtausgabelänge l1 < 1 oder die Anzahl der Ziffern nach dem Dezimalpunkt l2 < 1. Bei Write(f,a:l1) bzw. Writestring(s,a:l1) ist die Gesamtausgabelänge l1 < 1 bzw. l1 < 0 wenn a von einem String-Typ ist.	bei Check=On

SET_ERROR

Fehler	Fehlererkennung
- In einer Zuweisung liegt der Wert des Ausdrucks (rechte Seite) von einem Set-Typ nicht im Wertebereich des Typs der Variablen bzw. des Funktionsbezeichners (linke Seite).	bei Check=On
- Bei Wertparameterübergabe liegt der Wert des Aktualparameters von einem Set-Typ nicht im Wertebereich des Typs des Formalparameters.	bei Check=On
- In einem Aggregat liegt der Wert eines Aggregat-Elements von einem Set-Typ nicht im Wertebereich des Typs der zugehörigen Aggregatkomponente.	bei Check=On
- Beim Lesen aus einer Nicht-Text-Datei mit Read(f,v) liegt der Wert der Puffervariablen f↑ von einem Set-Typ nicht im Wertebereich des Typs der Variablen v.	bei Check=On
- Beim Schreiben in eine Nicht-Text-Datei mit Write(f,a) liegt der Wert des Ausdrucks a von einem Set-Typ nicht im Wertebereich des Typs der Puffervariablen f↑.	bei Check=On
- In einem Mengenbildner liegt der Wert einer Elementbestimmung nicht im Wertebereich des Basis-Typs des Mengenbildners.	bei Check=On
- Bei Setmin(s) oder Setmax(s) ist der Wert des Ausdrucks s gleich der leeren Menge (14.3.4).	bei Check=On

STRING_ERROR

Fehler	Fehlererkennung
- In einer Zuweisung ist die aktuelle Länge der Zeichenkette (rechte Seite) größer als die Maximallänge des String-Typs der Variablen bzw. des Funktionsbezeichners (linke Seite).	bei Check=On
- In einer Zuweisung ist die aktuelle Länge des Zeichenketten-Ausdrucks von einem String-Typ (rechte Seite) ungleich der Länge des Zeichenkettentyps fester Länge der verallgemeinerten Variablen bzw. des Funktions-Bezeichners (linke Seite) (10.1.2).	bei Check=On
- Bei Wertparameterübergabe ist die aktuelle Länge der Zeichenkette des Aktualparameters größer als die Maximallänge des String-Typs des Formalparameters.	bei Check=On
- Bei Wertparameterübergabe ist die aktuelle Länge der Zeichenkette des Aktualparameters ungleich der Länge des Zeichenkettentyps fester Länge des Formalparameters.	bei Check=On
- In einem Aggregat ist die aktuelle Länge einer Zeichenkette eines Aggregat-Elements größer als die Maximallänge des String-Typs der zugehörigen Aggregatkomponente.	bei Check=On
- In einem Aggregat ist die aktuelle Länge einer Zeichenkette eines Aggregat-Elements (von einem String-Typ) ungleich der Länge der zugehörigen Komponente des Aggregats (von einem Zeichenketten-Typ fester Länge).	bei Check=On
- Beim Lesen aus einer Nicht-Text-Datei mit Read(f,v) ist die aktuelle Länge der Zeichenkette von einem String-Typ in der Puffervariablen f↑ größer als die Maximallänge des Typs der String-Variablen v.	bei Check=On
- Beim Lesen aus einer Nicht-Text-Datei mit Read(f,v) ist die aktuelle Länge der Zeichenkette von einem String-Typ in der Puffervariablen f↑ ungleich der Länge des Zeichenkettentyps fester Länge der Variablen v.	bei Check=On
- Beim Schreiben in eine Nicht-Text-Datei mit Write(f,a) ist die aktuelle Länge der Zeichenkette größer als die Maximallänge des String-Typs der Puffervariablen f↑.	bei Check=On
- Beim Schreiben in eine Nicht-Text-Datei mit Write(f,a) ist die aktuelle Länge der Zeichenkette a von einem String-Typ ungleich der Länge der Puffervariablen f↑ von einem Zeichenkettentyp fester Länge.	bei Check=On
- Bei Read(f,v) bzw. Readstring(s,v) ist die Maximallänge der String-Variablen v kleiner als die Länge der eingelesenen Zeichenkette .	immer erkannt

Fortsetzung nächste Seite

Fehler	Fehlererkennung
- Bei Readstring(a,v1,...,vn) enthält der Zeichenkettenausdruck a nicht soviele Zeichen, wie durch die Leseparameter v1,...,vn angefordert werden.	immer erkannt
- Bei Writestring(s,p1,...,pn) ist die Maximallänge der String-Variablen s kleiner als die aus den Schreibparametern p1,...,pn gebildete Zeichenkette.	immer erkannt
- Bei Delete(s,i,l) ist i<1 oder l<0 oder (i+l-1) > Length(s).	bei Check=On
- Bei Insert(s1,s2,i) ist i<1 oder Length(s2) + Length(s1) > Maxlength(s2).	bei Check=On
- Bei Substring(s,i,l) ist i<1 oder l<0 oder (i+l-1) > Length(s).	bei Check=On
- Bei Pack(a,i,z) ist die Maximallänge der String-Variable z zu klein, um alle Zeichen aus dem ungepackten Array a ab Index i aufzunehmen.	bei Check=On

INDEX_ERROR

Fehler	Fehlererkennung
- Bei der Indizierung einer Array-Variablen, einer Array-Konstanten, eines Array-Aggregats oder eines Funktionsergebnisses von einem Array-Typ liegt der Wert des Index-Ausdrucks nicht im Wertebereich des Indextyps des Array-Typs.	bei Check=On
- Bei der Indizierung einer Variablen, einer Konstanten oder eines Funktionsergebnisses von einem String-Typ ist der Wert des Index-Ausdrucks kleiner als 1 oder größer als die aktuelle Länge der Zeichenkette.	bei Check=On
- Bei einem Konformreihungsparameter ist der Indextyp des Aktualparameters nicht ein Teilbereich des Indextyps des Konformreihungsschemas.	bei Check=On
- Bei Pack(a,i,z) liegt der Wert des Ausdrucks i nicht im Wertebereich des Indextyps des ungepackten Array-Parameters a.	bei Check=On
- Bei Pack(a,i,z) wird beim Übertragen der Komponenten aus dem ungepackten Array a ab Index i in das gepackte Array z der Indexbereich von a überschritten.	bei Check=On
- Bei Unpack(z,a,i) liegt der Ordinalwert des Ausdrucks i nicht im Wertebereich des Indextyps des ungepackten Array-Parameters a.	bei Check=On
- Bei Unpack(z,a,i) ist der angegebene Bereich im ungepackten Array a ab Index i zu klein, um alle Komponenten des gepackten Array z aufzunehmen.	bei Check=On
- Bei Unpack(z,a,i) enthält der Zeichenkettenausdruck z mehr Zeichen, als in das ungepackte Array a ab Index i übertragen werden können.	bei Check=On

POINTER_ERROR

Fehler	Fehlererkennung
- Bei einer Zeigerdereferenzierung ist der Wert der Variablen, der Konstanten oder des Funktionsergebnisses von einem Zeigertyp gleich Nil.	bei Check=On
- Beim Aufruf von Dispose(p) hat p den Wert Nil.	immer erkannt
- Beim Aufruf von Release(p) wurde der Verweiswert von p nicht durch einen Aufruf von Mark erzeugt.	immer erkannt

VARIANT_ERROR

Fehler	Fehlererkennung
- Es wird auf eine nicht aktive Variante einer Variablen, einer Konstanten, eines Aggregats oder eines Funktionsergebnisses von einem Record-Typ zugegriffen.	bei Check=On

CASE_ERROR

Fehler	Fehlererkennung
- In einem Case-Statement entspricht keine Fall-konstante dem Wert des Fall-Index, und es ist auch keine Else-Alternative angegeben.	bei Check=On

FILE_ERROR

Fehler	Fehlererkennung
- Vor dem Aufruf von Page(f), Put(f), Write(f,...) oder Writeln(f,...) wurde die Datei f nicht zum Schreiben eröffnet.	immer erkannt
- Vor dem Aufruf von Page(f), Put(f), Write(f,...) oder Writeln(f,...) ist die Datei f undefiniert.	immer erkannt
- Vor dem Aufruf von Page(f), Put(f), Write(f,...) oder Writeln(f,...) ist die aktuelle Dateiposition nicht die Dateiende-Position, d.h. Eof(f) ist False.	immer erkannt
- Vor dem Aufruf von Get(f) oder Read(f,...) wurde die Datei f nicht zum Lesen eröffnet.	immer erkannt
- Vor dem Aufruf von Get(f) oder Read(f,...) ist die Datei f undefiniert.	immer erkannt
- Vor dem Aufruf von Read(f,...) ist die Puffer-variable f↑ der Datei f undefiniert.	immer erkannt
- Vor dem Aufruf von Eof(f) ist die Datei f unde-finiert.	bei Check=On
- Vor dem Aufruf von Eoln(f) ist die Datei f unde-finiert.	bei Check=On
- Bei Assignfile(f,ext) ist die Beschreibung der externen Datei im Operanden "ext" fehlerhaft.	immer erkannt

EOF_ERROR

Fehler	Fehlererkennung
- Beim Aufruf von Get(f) oder Read(f,...) ist das Dateiende bereits erreicht, d.h. Eof(f) ist True.	immer erkannt
- Beim Aufruf von Eoln(f) ist das Dateiende bereits erreicht, d.h. Eof(f) ist True.	bei Check=On

OPEN_ERROR

Fehler	Fehlererkennung
- Bei Aufruf von Reset oder Rewrite wurde die vordefinierte Textdatei Input oder Output angegeben.	immer erkannt
- Beim Aufruf von Reset(f) ist die Datei f undefiniert	immer erkannt
- Bei Reset(f) kann die an f gebundene, außerhalb des Programms liegende Datei nicht zum Lesen eröffnet werden.	immer erkannt
- Bei Rewrite(f) kann die an f gebundene, außerhalb des Programms liegende Datei nicht zum Schreiben eröffnet werden.	immer erkannt

READ_ERROR

Fehler	Fehlererkennung

- Beim Lesen einer Integerzahl oder Realzahl aus einer Textdatei (mit Read(f,v)) oder aus einem Zeichenkettenausdruck (mit Readstring(s,v)) gilt:
  - die eingelesene Zeichenfolge ist syntaktisch fehlerhaft oder
  - die eingelesene Zeichenfolge ergibt eine Zahl, die intern nicht mehr darstellbar ist. Bei Integerzahlen liegt der Wert außerhalb des Bereichs Long_Minint .. Long_Maxint und bei Realzahlen liegt der Wert außerhalb des Bereichs -Long_Maxreal .. Long_Maxreal.
	immer erkannt

MEMORY_ERROR

Fehler	Fehlererkennung
- Die Ausführung des Programms kann wegen fehlenden Speichers nicht fortgesetzt werden (z.B. bei Aufruf eines Unterprogramms oder bei New).	immer erkannt

ELAB_ERROR

Fehler	Fehlererkennung
- Die Initialisierung der Pakete eines Programms kann nicht fortgesetzt werden, da durch die Verwendung der vordefinierten Prozedur Elaborate Zyklen bei der Initialisierung entstehen.	immer erkannt

Sonstige Fehler (ohne zugeordneter Fehlernummer)

Die in der folgenden Tabelle angegebenen Fehler haben keine zugeordnete Fehlernummer. Das Auftreten eines solchen Fehlers wird nicht erkannt und führt i.a. zu Folgefehlern mit undefinierten Auswirkungen.

Fehler	Fehlererkennung
- In einer Zuweisung ist der Typ des Ausdrucks (rechte Seite) vom generischen Zeigertyp und der Zeigerwert des Ausdrucks verweist auf eine dynamische Variable, deren Typ vom Domänentyp des Typs der verallgemeinerten Variablen bzw. des Funktions-Bezeichners (linke Seite) verschieden ist.	nicht erkannt
- Bei Wertparameterübergabe ist der Typ des Aktualparameters vom generischen Zeigertyp und der Zeigerwert des Ausdrucks verweist auf eine dynamische Variable, deren Typ vom Domänentyp des Typs des Formalparameters veschieden ist.	nicht erkannt
- In einem Aggregat ist der Typ eines Zeigerwertes vom generischen Zeigertyp und der Zeigerwert des Ausdrucks verweist auf eine dynamische Variable, deren Typ vom Domänentyp des Typs der entsprechenden Aggregatkomponente verschieden ist.	nicht erkannt
- Die Länge einer String-Variablen wird verändert, obwohl noch eine Referenz auf eine Komponente der String-Variablen existiert.	nicht erkannt
- Bei Writestring(s,p1,...,pn) enthält einer der Schreibparameter p1,...,pn eine Referenz auf die String-Variable s.	nicht erkannt
- Bei Convert(x,t) repräsentiert die Speicherdarstellung von x keinen zulässigen Wert des Typs t.	nicht erkannt

Fortsetzung nächste Seite

Fehler	Fehlererkennung
- Die Variante einer Record-Variablen ist nicht für die Gesamtdauer jeglicher Referenz auf jede ihrer Komponenten aktiv.	nicht erkannt
- Der Dateizeiger einer Dateivariablen f wird (z.B. durch Lesen oder Schreiben) geändert, obwohl noch eine Referenz auf die Puffervariable f existiert.	nicht erkannt
- Bei einer Zeigerdereferenzierung ist der Wert der Variablen, der Konstanten oder des Funktionsergebnisses von einem Zeigertyp undefiniert.	nicht erkannt
- Mit Dispose(q) wird ein Verweiswert auf eine dynamische Variable entfernt, obwohl noch eine Referenz auf die dynamische Variable existiert.	nicht erkannt
- Beim Aufruf von Dispose(p) ist der Wert von p undefiniert.	nicht erkannt
- Vor dem Aufruf von Dispose(p) ist p↑ durch New(p,c1,...,cn) oder New(p,c1,...,cn,e) oder New(p,e) erzeugt worden.	nicht erkannt
- Vor dem Aufruf von Dispose(p,k1,...,km) ist die dynamische Variable p↑ durch New(p,c1,...,cn) erzeugt worden, wobei m ungleich n ist.	nicht erkannt
- Bei Dispose(p,k1,...,kn) oder Dispose(p,k1,...kn,e) sind in der dynamischen Variable p↑ andere Varianten eingestellt, als durch die Selektorkonstanten k1 bis kn angegeben.	nicht erkannt
- Vor dem Aufruf von Dispose(p,e) ist p↑ durch New(p,a) erzeugt worden, wobei a ungleich e ist. Analog gilt dies auch für Dispose(p,c1,...,cn,e) und New(p,k1,...kn,a).	nicht erkannt
- Bei Dispose(p,e) oder Dispose(p,c1,..., cn, e) liegt der Wert von e nicht im Wertebereich des Indextyps des entsprechenden ARRAY-Typs bzw. ist kleiner als 1 oder größer als die Maximallänge des entsprechenden String-Typs.	nicht erkannt
- Bei einem indizierten ARRAY- oder String-Objekt wurde das ARRAY- bzw. String-Objekt verkürzt durch den Aufruf von New(p,e) oder New(p,c1,...,cn,e) erzeugt und der Wert des Indexausdrucks im indizierten Objekt ist größer als e.	nicht erkannt
- In einem indizierten String-Objekt ist der Wert des String-Objekts nicht definiert (unabhängig davon, ob das indizierte Objekt in einem Ausdruck oder z.B. als verallgemeinerte Variable auf der linken Seite einer Zuweisung auftritt).	nicht erkannt

Fortsetzung nächste Seite

Fehler	Fehlererkennung
- In einer mit New(p,c1,...,cn) oder New(p,c1,...,cn,e) erzeugten dynamische Variable wird eine andere Variante eingestellt, als durch die Selektorkonstanten c1 bis cn angegeben wurde.	nicht erkannt
- Einer dynamischen String-Variablen, die mit New(p,e) erzeugt wurde, oder an die letzte Komponente einer dynamischen String-Variablen, die mit New(p,c1,...,cn,e) erzeugt wurde, wird eine Zeichenkette zugewiesen, die länger ist als e.	nicht erkannt
- Eine dynamische Variable, die durch New(p,e), New(p,c1,...,cn) oder New(p,c1,...,cn,e) erzeugt wurde, kommt als Ganzes in einem Ausdruck oder als linke Seite in einer Wertzuweisung vor, oder wird als Parameter übergeben.	nicht erkannt
- Bei New(p,e) oder New(p,c1,...,cn,e) liegt der Wert von e nicht im Wertebereich des Indextyps des entsprechenden ARRAY-Typs bzw. ist kleiner als 1 oder größer als die Maximallänge des entsprechenden String-Typs.	nicht erkannt
- Der beim Aufruf von Release(p) übergebene Verweis-Wert p wurde durch einen anderen Aufruf von Release(q) vernichtet.	nicht erkannt
- Vor dem Aufruf von Put(f), Put(f,c1,...,cn), Put(f,e) oder Put(f,c1,...,cn,e) ist die Puffer-Variable f↑ undefiniert.	nicht erkannt
- Bei der verkürzten Ausgabe eines Arrays mit Put(f,e) oder Put(f,c1,...,cn,e) liegt der Wert des Indexausdrucks e nicht im Wertebereich des Indextyps des Arrays.	nicht erkannt
- Bei der verkürzten Ausgabe einer Zeichenkette variabler Länge mit Put(f,e) oder Put(f,c1,...cn,e) ist der Wert des Indexausdrucks e kleiner als 1 oder größer als die aktuelle Länge der Zeichenkette.	nicht erkannt
- Eine verallgemeinerte Variable, die als Objekt in einem Ausdruck verwendet wird, hat zum Zeitpunkt der Auswertung des Ausdrucks einen undefinierten Wert.	nicht erkannt
- Das Ergebnis einer Funktion ist nach Ausführung des Funktionsblocks undefiniert, wenn dem Funktionsbezeichner kein Wert zugewiesen wurde.	nicht erkannt
- Bei Unpack(z,a,i) ist irgendeine Komponente des gepackten Arrays z undefiniert.	nicht erkannt
- Bei Pack(a,i,z) wird auf eine Komponente des ungepackten Arrays a zugegriffen, die undefiniert ist [D.27].	nicht erkannt
- Die Bezeichner der Programmparameter des Hauptprogramms und aller dazugehörigen Pakete sind, mit Ausnahme von Input und Output, nicht paarweise verschieden.	nicht erkannt
- Die Namen aller zu einem Programm gehörigen Pakete und der Name des Hauptprogramms sind nicht paarweise verschieden.	nicht erkannt

Systemfehlercodes

In den folgenden Tabellen werden für die vordefinierten Ausnahmen die möglichen Systemfehlercodes aufgelistet, wie sie durch die Funktion System_Code des vordefinierten Pakets ERRORS (siehe A.7) geliefert werden. Systemfehlercodes größer 9999 sind Systemfehlermeldungen und müssen erst in eine Hexadezimalzahl umgewandelt werden, um die entsprechenden Fehlertexte mit dem Systemkommando HELP abfragen zu können. Die Bedeutung der Unterbrechungsgewichte ist aus [6] zu entnehmen.

Ein Pascal-Programm liefert immer den System_Code 0, wenn eine der Ausnahmen Numeric_Error, Range_Error, Set_Error, String_Error, Index_Error, Pointer_Error, Variant_Error oder Case_Error aufgrund der eingeschalteten Check-Option erkannt wird.

SYSTEM_ERROR (-1)

System_Code	Bedeutung
1201	Aufruf eines 24 Bit Unterprogramms mit Parametern im 31-Bit-Bereich
1405	HEAPSUPPORT.select_heap wurde auf einen ungültigen Heapzeiger angewandt
1406	HEAPSUPPORT.release_heap wurde auf einen ungültigen Heapzeiger angewandt
1407	HEAPSUPPORT.release_heap wurde auf den augenblicklichen Heap (current_heap) angewandt
1408	HEAPSUPPORT.release_heap wurde auf den default_heap angewandt
1409	NEW und DISPOSE: Diese Prozeduren dürfen in UTM-Anwendungen nicht aufgerufen werden
1413	Fehler bei der Rückgabe von Hauptspeicher an das Betriebssystem
1414	Kein Speicher zum Einrichten einer Glue Page vorhanden
1503	Pascal-Filevariable muß vom Typ TEXT sein
2250	Fehler beim Laden des EDT
2251	Fehler beim Initialisieren des EDT
2252	Fehler bei SUBMIT_EDT
2254	Fehler beim Ausführen des EDT
2255	Fehler beim Lesen aus der virtuellen Datei
2256	SUBMIT_EDT: Zeilenlänge ist größer als 256
2257	Fehler beim Schreiben in die virtuelle Datei des EDT
3001	Fehler beim Aufruf des TABLE-Makros
4002	Fehler in der Liste der globalen Dateien
4003	Fehler in der Liste der lokalen Dateien
5001	Reraise: Es ist noch keine Ausnahmesituation aufgetreten
5002	Fehler in einem fremdsprachigen Unterprogramm aufgetreten
sonstige	Unterbrechungsgewicht (siehe [6]).

NUMERIC_ERROR (-2)

System_Code	Bedeutung
1001	EXP: Das Argument ist zu groß
1002	LN: Das Argument ist nicht positiv
1003	SIN bzw. COS: Der Absolutbetrag des Arguments ist größer oder gleich pi * 2**50
1004	SQRT: Das Argument ist negativ
sonstige	Unterbrechungsgewicht (siehe [6]).

STRING_ERROR (-5)

System_Code	Bedeutung
0	Fehler bei einer String-Zuweisung oder bei INSERT, DELETE oder SUBSTRING (erkannt aufgrund von {$CHECK=ON})
1012	WRITESTRING: Die Ausgabelänge ist ungültig
1013	WRITESTRING: Der String ist zu klein, um ein Zeichen aufzunehmen
1014	WRITESTRING: Der String ist zu klein, um einen Integerwert aufzunehmen
1015	WRITESTRING: Der String ist zu klein, um eine Realzahl aufzunehmen
1016	WRITESTRING: Der String ist zu klein, um einen String aufzunehmen
1017	WRITESTRING: Der String ist zu klein, um einen booleschen Wert aufzunehmen
1025	READSTRING: Das Ende der Stringvariablen ist erreicht
1030	READSTRING, READ: Die Stringvariable ist zu klein, um den Rest der Zeile aufzunehmen

POINTER_ERROR (-7)

System_Code	Bedeutung
0	Derefenzierung eines NIL-Zeigers (erkannt aufgrund von {$CHECK=ON}) oder eines nicht initialisierten Zeigers (erkannt aufgrund von {$INITIALIZE=ON, CHECK=ON})
1404	RELEASE: Der Zeigerwert wurde nicht mittels MARK erzeugt oder MARK wurde für einen anderen Heap aufgerufen.
1410	DISPOSE: Angabe eines ungültigen Verweiswertes
1411	DISPOSE: Angabe einer falschen Länge
1412	DISPOSE: Der freizugebende Speicher wird von einer dynamischen Variablen eines anderen Typs belegt.
sonstige	Unterbrechungsgewicht (siehe [6]) (von der Hardware erkannter Adressierungsfehler)

FILE_ERROR (-10)

System_Code	Bedeutung
1090	Pufferüberlauf bei der Text-File Ausgabe
1091	Pufferüberlauf bei der Text-File Eingabe [1)]
1095	WRITE: Die Ausgabelänge ist ungültig (wird auch bei Check=Off erkannt)
1096	PUT: Ungültige Längenangabe
1500	ASSIGNFILE: Die Länge der externen Beschreibung ist ungültig
1501	ASSIGNFILE: Syntaxfehler in der externen Beschreibung
1502	ASSIGNFILE: Ungültige Angaben in der externen Beschreibung
2010	WRITELN: Gerätefehler
2011	READLN: Gerätefehler

Fortsetzung nächste Seite

1) Eine eingelesene Zeile ist länger als der Laufzeitsystem-interne Puffer (siehe auch Open_Error).

System_Code	Bedeutung
2045	Ein-/Ausgabe auf eine nicht eröffnete Datei
2046	Die Datei ist für diesen Aufruf nicht zugelassen
2049	UPDATE: Aufruf mit ungültiger Satzlänge
2105	Bibliothekselement kann nicht geschlossen werden
2106	Bibliothek kann nicht geschlossen werden
2109	PLAM Zugriffs-Fehler
2201	Fehler beim Lesen einer EAM-Datei
2202	Fehler beim Schreiben in eine EAM-Datei
2203	Fehler beim Schließen einer EAM-Datei
4001	Fehler im Laufzeitsystem
sonstige	Fehlercodes des DVS

OPEN_ERROR (-12)

System_Code	Bedeutung
1091	Pufferüberlauf beim Eröffnen einer Textdatei [1)]
1603	Der Dateiname ist nicht bestimmbar
1604	RESET: Eröffnen einer nicht existierenden Datei
1605	Ungültiger Open-Modus für die Datei
1606	Kein Hauptspeicher für Block-/Satzpuffer mehr verfügbar
1607	RESET, REWRITE, REPLACE, EXTEND und CLOSE dürfen nicht auf die vordefinierten Textdateien Input und Output angewendet werden
1608	Falsche oder fehlende Parameter im FILE-Kommando (FCBTYPE, RECFORM, KEYLEN, KEYPOS, RECSIZE, BLKSIZE, SPACE)
1609	Es wurde versucht, eine Datei nach einem fehlerhaftem Assignfile zu eröffnen.
2100	Die Systembibliothek für den Zugriff auf PLAM-Bibliotheken ist nicht vorhanden.
2101	Falscher bzw. unbekannter Bibliothekstyp
2102	Bibliothek kann nicht eröffnet werden
2103	Bibliothekselement ist nicht vorhanden
2104	Bibliothekselement kann nicht eröffnet werden
2110	Unzulässige Versionsangabe bei PLAM-Bibliothekselementen
2200	Fehler beim Eröffnen einer EAM-Datei
sonstige	Fehlercodes des DVS

1) Beim Eröffnen einer Textdatei zum Lesen wird bereits die erste Zeile gelesen, die länger sein kann als der Laufzeitsystem-interne Puffer (siehe auch File_Error).

READ_ERROR (-13)

System_Code	Bedeutung
1092	Fehler beim Lesen einer Integer-Zahl
1094	Fehler beim Lesen einer Realzahl

MEMORY_ERROR (-14)

System_Code	Bedeutung
1401	Hauptspeicherüberlauf bei interner Anforderung
1402	Hauptspeicherüberlauf bei einer Stack-Anforderung
1403	Hauptspeicherüberlauf bei einer Heap-Anforderung
1601	Kein Hauptspeicher mehr vorhanden zum Einrichten eines FCB

Anhang

Gegenüberstellung von Pascal (BS2000) Version 3.x und Pascal-XT

Es werden alle Spracheigenschaften von Pascal (BS2000) Version 3.1B (kurz Version 3 genannt) und Pascal-XT einander gegenübergestellt. Der Vergleich erfolgt nur stichwortartig, Details sind den jeweiligen Sprachbeschreibungen zu entnehmen.

In den Tabellen sind alle Eigenschaften der Sprachen aufgelistet. Das Vorhandensein oder Fehlen einer Eigenschaft in einer Sprache wird angezeigt durch:

X Eigenschaft vorhanden
– Eigenschaft fehlt.

Die wesentlichen Unterschiede zwischen Pascal Version 3 und Pascal-XT sind:

– unterschiedliche Konzepte der getrennten Übersetzung
– unterschiedliche Konzepte für die programmierte Ausnahmebehandlung
– Pascal-XT umfaßt zusätzlich das Level 1 der ISO-Norm
– Aggregate (strukturierte Werte) in Pascal-XT
– Statische Ausdrücke als Konstanten in Pascal-XT
– Einflußnahme auf Speicherdarstellung in Pascal-XT
– Inline-Unterprogramme in Pascal-XT
– Unterschiedliche Standardprozeduren
– ISAM Dateien sind nicht in der Sprache Pascal-XT integriert, sondern werden in einem Paket angeboten
– Include-Mechanismus entfällt in Pascal-XT

Satzlängen bei Nicht-Textdateien

Ein Pascal-XT-Programm kann in gewissen Fällen Nicht-Textdateien, die von Version 3 erstellt wurden, nicht verarbeiten. Der Grund dafür ist, daß in Version 3 im Gegensatz zu Pascal-XT die Satzlänge immer auf ein ganzzahliges Vielfaches der Ausrichtung des Komponententyps der Datei aufgerundet wird. Zum Lesen einer solchen Datei muß der Komponententyp der Datei in Pascal-XT um ein Dummy-Feld erweitert werden, dessen Größe sich aus der Differenz der Satzlängen ergibt (siehe Beispiel).

Die Satzlänge wird folgendermaßen berechnet:

```
T   sei der Komponententyp eines File-Typs (FILE OF T)
L   Länge des Satzes der BS2000-Datei
LF  Größe des Satzlängenfelds  = 4  bei RECFORM = V
                               = 0  bei RECFORM = F
sizeof  (T) :  Größe des Komponententyps (in Byte)
alignof (T) :  Ausrichtung des Komponententyps (1, 2, 4 oder 8)

V3.x        :  L := ((sizeof (T) + LF) + alignof (T) - 1)
                          div alignof (T)  *  alignof (T)
               Die Größe des Satzes ist also stets das kleinste ganz-
               zahlig Vielfache des Alignments von T, das größer ist
               als (sizeof (T) + LF)

Pascal-XT   :  L := sizeof (T) + LF
               Die Satzgröße wird nicht aufgerundet
```

Beispiel

```
type
    satz = record            (* Definition des Satzes in Version 3 *)
             i: integer;
             c: char;
           end;
var
    datei: file of satz;

Für diese Datei gilt:
   sizeof  (satz) = 5
   alignof (satz) = 4
```

	Satzlänge bei RECFORM = V	Satzlänge bei RECFORM = F
V3.x	12	8
Pascal-XT	9	5

Zum Lesen einer solchen von der Version 3 erstellten Datei muß der Record-Typ satz in Pascal-XT um 3 Byte vergrößert werden:

```
type
    satz = record            (* Definition des Satzes in Pascal-XT *)
             i: integer;
             c: char;
             d: packed array [1..3] of char;  (* Dummy-Feld *)
           end;
```

Schlüsselwörter

Zusätzliche Schlüsselwörter in Pascal-XT sind:

```
BODY        ENTRY        EXIT       EXCEPTION        FROM
INLINE      PACKAGE      USE
```

Übersetzungseinheiten

Die Konzepte für getrennte Übersetzungen sind zwischen den beiden Sprachen nicht kompatibel. Version 3 kennt ein Modulkonzept ohne Überprüfung der Schnittstellen. In Pascal-XT sind Schnittstellen und Implementierung getrennt und die Überprüfung der Schnittstellen erfolgt zur Übersetzungszeit.

Übersetzungseinheit	Version 3	Pascal-XT
program	X	X
Prozedurmodul	X	werden durch das Paket-konzept abgedeckt
Funktionsmodul	X	
Modul	X	
Entry-Prozedur	X	
package	-	X
package body	-	X

Direktiven und Sprachanschlüsse

In Version 3 werden alle externen Unterprogramme durch die Direktive "external" gekennzeichnet. In Pascal-XT ist für Cobol-Unterprogramme die Direktive "cobol", für Fortran-Unterprogramme die Direktive "fortran" und für alle anderen Unterprogramme die Direktive "external" anzugeben.
Bei Sprachverknüpfungen sind in Pascal-XT beliebige Aufrufsequenzen ohne explizite Initialisierungen möglich, in Version 3 sind Initialisierungen der Laufzeitumgebung explizit anzugeben.
Entry-Prozeduren sind in Version 3 eine eigene Übersetzungseinheit, in Pascal-XT werden sie in Paketspezifikationen definiert (siehe Übersetzungseinheiten).

Direktive	Version 3	Pascal-XT
module	X	-
forward	X	X
internal	X	X
external	X	X
fortran	-	X
cobol	-	X

Pseudokommentare und Compileroptionen

Die Compileroptionen der beiden Compiler sind nicht verträglich. Etwa gleichwertige Optionen sind in der Tabelle einander gegenübergestellt, die aber i.a. nicht denselben Funktionsumfang abdecken.
Version 3 bietet zur bedingten Übersetzung von Programmteilen die Steueranweisungen SKIPON und SKIPOFF an. In Pascal-XT kann dies mit der IF-Anweisung simuliert werden, wenn die IF-Bedingung ein statischer Ausdruck ist. Hat die Bedingung den Wert "True", dann wird nur für den THEN-Part Code erzeugt, ansonsten nur für den ELSE-Part. Bei einer CASE-Anweisung mit statischem Fall-Index wird nur Code für das entsprechende Fall-Listenelement erzeugt.

Version 3	Pascal-XT
(*%MAP*)	} die beiden Steueranweisungen werden
(*%COPY*)	} durch das Paketkonzept abgedeckt
(*%SKIPON*)	siehe oben
(*%SKIPOFF*)	siehe oben
(*$...*)	(*$...}
R, E, K, F, H, S	-
J, Q, V, T, Z	-
O	OPTIMIZE = on | off
D	CHECK = on | off
G	GENERATE = on | off
N	automatisch, wenn mit PATH getestet wird
Y, U	DEBUG = on | off | restricted
L	LIST = on | off
L n	-
X	XREF = on | off
P	PAGE (ohne Titelangabe)
-	TITLE = '...' (ohne Seitenwechsel)
C	ASSEMBLER = on | off
W	STANDARD = on | off
I	MAP = on | off

Programmparameterliste

In Version 3 kann für externe Textdateien die Puffergröße hinter dem Dateibezeichner in der Programmparameterliste angegeben werden. In Pascal-XT ist die Puffergröße fest vorgegeben.

Konstantendefinitionen

In Pascal-XT können in Konstantendefinitionen Konstante beliebigen, auch strukturierten Typs definiert werden. Auf der rechten Seite der Definition können beliebige statische Ausdrücke stehen.

Konstantendefinition	Version 3	Pascal-XT
Zeichenliteral (char)	X	X
Aufzählungstyp (boolean)	X	X
Zahlen (integer, real)	X	X
Zeichenketten	X	X
Konstantenbezeichner	X	X
statischer Ausdruck	-	X
- arith. bzw. logischer Ausdruck	-	X
- variabler Zeichenkettenausdruck	-	X
- (qualifizierte) Mengenbildner	-	X
- Array-Aggregat	-	X
- Record-Aggregat	-	X

Vordefinierte Konstanten

Bezeichner / Schlüsselwort	Version 3	Pascal-XT
TRUE	X	X
FALSE	X	X
MAXINT	X	X
SHORT_MAXINT	-	X
LONG_MAXINT	-	X
MININT	-	X
SHORT_MININT	-	X
LONG_MININT	-	X
MINREAL	X	X
SHORT_MINREAL	-	X
LONG_MINREAL	-	X
MAXREAL	X	X
SHORT_MAXREAL	-	X
LONG_MAXREAL	-	X
NIL	X	X

Vordefinierte Typen und neue Typen

In Pascal-XT können Konstante in Typdefinitionen auch statische Ausdrücke sein. Zeichenkettentypen werden nach ISO-Norm durch packed array [1..n] of char definiert, variable Zeichenkettentypen durch string [n], wobei n die maximale Stringlänge angibt. In Version 3 muß ein Zeichenkettentyp nicht gepackt sein und die Indexuntergrenze darf von 1 verschieden sein. Pascal-XT hält sich streng an die ISO-Norm.

Typ	Version 3	Pascal-XT
CHAR	X	X
BOOLEAN	X	X
INTEGER	X	X
SHORT_INTEGER	-	X
LONG_INTEGER	-	X
REAL	X	X
SHORT_REAL	-	X
LONG_REAL	-	X
Aufzählungs-Typ	X	X
Teilbereichs-Typ	X	X
Zeigertyp	X	X
Generischer Zeigertyp	-	X
Privater Zeigertyp	-	X
Zeichenkettentyp	X	X (nach ISO-Norm)
Variabler Zeichenkettentyp	X	X
- Standardlänge =	252	254
- Längenfeld =	4	2
- Maximallänge =	$2^{16}-1$	$2^{15}-1$
Record-Typ	X	X
- Angaben zur Darstellung	-	X
- else-Teil im Variantteil	-	X
- Bereichsangaben bei den Selektorkonstanten	-	X
Array-Typ	X	X
File-Typ	X	X
- ISAM Dateien	X	-
Generischer Filetyp	-	X
Mengen-Typ	X	X
Corid-Typ	X	-

Variable, Ausdrücke und Operatoren

In Pascal-XT gibt es neben Komponenten von Variablen auch Komponenten von Werten eines strukturierten Typs.

Ausdrücke, die zur Übersetzungszeit ausgewertet werden können, heißen statisch. In Pascal-XT können statische Ausdrücke als Konstanten sowohl in Konstantendefinitionen als auch an Stellen eingesetzt werden, an denen die ISO-Norm eine Konstante fordert.

Operatoren

Operator	Version 3	Pascal-XT
Monadische Operatoren:		
+	X	X
-	X	X
Arithmetische Opertoren:		
+	X	X
-	X	X
*	X	X
/	X	X
** (Exponentation)	-	X
div	X	X
mod	X	X
Boolesche Operatoren:		
or	X	X
and	X	X
not	X	X
or else	-	X
and then	-	X
Mengenoperatoren:		
+	X	X
-	X	X
*	X	X
/ (symmetrische Differenz)	-	X
Vergleichsoperatoren:		
=	X	X
<>	X	X
<	X	X
>	X	X
<=	X	X
>=	X	X
in	X	X

Bei Vergleichen können Zeichenkettentypen in Version 3 unterschiedlich lang sein, in Pascal-XT müssen die Zeichenkettentypen gemäß ISO-Norm dieselbe Anzahl von Zeichen enthalten. Bei variablen Zeichenkettentypen ist in beiden Sprachen nur die aktuelle Länge relevant.

Anweisungen

Anweisung	Version 3	Pascal-XT
Zuweisung [1)]	X	X
goto	X	X
exit	-	X
return	X	X
Verbundanweisung	X	X
if	X	X
case	X	X
- Bereiche für Fallkonstanten	-	X
repeat	X	X
while	X	X
for	X	X
with	X	X
Prozeduraufrufe	X	X
Inline Prozeduraufrufe	-	X

1) In Version 3 kann einer Variablen eines Zeichenkettentyps auch eine kürzere Zeichenkette zugewiesen werden, in Pascal-XT muß gemäß der ISO-Norm der Zeichenkettenausdruck vom selben Typ wie die Zeichenkettenvariable sein.

Prozedur- und Funktionsdeklarationen und formale Parameter

In Pascal-XT darf die Formalparameterliste wiederholt werden.

Entry-Prozeduren

In Version 3 sind Entry-Prozeduren eigenständige Übersetzungseinheiten. In Pascal-XT können Entry-Prozeduren nur in Paketspezifikationen deklariert werden.

Inline-Unterprogramme

In Pascal-XT können Prozeduren und Funktionen als Inline deklariert werden.

Werteparameter

Bei Formalparametern eines Zeichenkettentyps kann in Version 3 der Aktualparameter auch ein Zeichenkettenausdruck mit kürzerer Länge sein. In Pascal-XT muß der Aktualparameter entsprechend der ISO-Norm denselben Typ haben.
Pascal-XT läßt zusätzlich Konformreihungsschemata zu.

Variablenparameter

Pascal-XT läßt zusätzlich Konformreihungsschemata zu.

Parameterprozeduren und -Funktionen

Keine Unterschiede.

Programmierte Ausnahmebehandlung

Die Konzepte zur programmierten Ausnahmebehandlung sind in beiden Sprachen gänzlich verschieden. In Version 3 gilt ein Ausnahmebehandler ab seiner Anmeldung so lange eingestellt, bis ein neuer angemeldet oder der Block beendet wird, der seine Anmeldungen unmittelbar enthält. In Pascal-XT kann ein Ausnahmebehandler für eine einzelne Verbundanweisung oder den gesamten Rumpf eines Unterprogramms definiert werden.

Standardprozeduren und Standardfunktionen

Prozedur/Funktion	Version 3	Pascal-XT
PACK	X	X
UNPACK	X	X
ABS	X	X
SQR	X	X
SIN	X	X
COS	X	X
EXP	X	X
LN	X	X
SQRT	X	X
ARCTAN	X	X
TRUNC	X	X
SHORT_TRUNC	-	X
LONG_TRUNC	-	X
ROUND	X	X
SHORT_ROUND	-	X
LONG_ROUND	-	X
LONG	-	X
ORD	X	X
CHR	X	X
SUCC	X	X
PRED	X	X
ODD	X	X

Prozeduren zur Ein-/Ausgabe

Prozedur/Funktion	Version 3	Pascal-XT	
RESET	X	X	
REWRITE	X	X	
GET	X	X	
PUT	X	X	
PUT (f,c1,...,cn,e)	X	X	
EOF	X	X	
READ	X	X	
READ (f,str:len)	X	-	
WRITE	X	X	
WRITE (f,str:len1:len2)	X	-	
WRITE (menge:len HEX)	X	-	
WRITE (f,expr HEX)	X	-	
READLN	X	X	
WRITELN	X	X	3)
EOLN	X	X	
PAGE	X	X	
PAGE (f,exp1,exp2,...)	X	-	
EXTEND	X		
REPLACE	X		
CLOSE	X		
ELIM	X		
ELIMKEY	X		
GETKEY	X		
GETBACK	X		
UPDATE	X		
INSERT	X		werden im vordefinierten
POS	X		Paket DMSIO zur Verfügung
POSKEY	X		gestellt
POSBACK	X		
MOVEKEY	X		
NOKEY	X		
LOCK	X		
RECLEN	X		
KEYPOS	-		
KEYLEN	-		
CLPAM	X	-	
OPPAM	X	-	
RDPAM	X	-	
WRPAM	X	-	
EAM	X	über lokale Dateien	

Prozeduren zur Haldenverwaltung

Prozedur/Funktion	Version 3	Pascal-XT	
NEW	X	X	
DISPOSE	X	X	
MARK	X	X	
RELEASE	X	X	
PUSHEAP	X	⎫	mit den Funktionen des
POPHEAP	X	⎬	vordefinierten Pakets
KILLHEAP	X	⎭	HEAPSUPPORT nachbildbar
RELMEM	X	⎫	werden im vordefinierten
REQMEM	X	⎬	Paket MEMORYMANAGER
GETMAP	X	⎭	zur Verfügung gestellt
CHPTR (p,e)	X	- 1)	
ASSPTR (p,q)	X	- 2)	

1) p := CONVERT (CONVERT (p,integer)+e, pointer)

2) p := CONVERT (q, pointer)

3) Zur Erzeugung einer Leerzeile muß in Pascal-XT mindestens ein Leerzeichen (Blank) ausgegeben werden.

Prozeduren für die programmierte Ausnahmebehandlung

Prozedur/Funktion	Version 3	Pascal-XT
RAISE	X	X
ERROREXIT	X	-
SYSERROR	X	-
ERROR_NUMBER	-	X

Unterprogramme für Zeichenkettenverarbeitung

Prozedur/Funktion	Version 3	Pascal-XT
DELETE	X	X
INSERTSTRING	X	X = INSERT
READSTRING	X	X
WRITESTRING	X	X
LENGTH	X	X
POSITION	X	X
CONCAT	X	X
SUBSTRING	X	X

Attributfunktionen

Funktion	Version 3	Pascal-XT
ALIGNOF	X	X [1]
SIZEOF	X	X [1]
BITSIZEOF	-	X [1]
CARD	X	X
SETMIN	-	X
SETMAX	-	X
FIRST	-	X
LAST	-	X
MAXLENGTH	-	X

[1] auf Variable und Typen anwendbar

Coroutinen

Coroutinen werden in Pascal-XT zur Zeit nicht unterstützt.

Prozedur/Funktion	Version 3	Pascal-XT
ALLOCATE	X	-
NEWCOR	X	-
TRANSFER	X	-
ENDCOR	X	-
USED	X	-

Sonstige Unterprogramme

Version 3	Pascal-XT
-	CONVERT (ungeprüfte Typkonvertierung)
-	ELABORATE (expl. Paketinitialisierung)
EDT	EDTADAPTER . call_edt
EDT (cmd)	EDTADAPTER . submit_edt (cmd)
EDP	-
EDOR	-
WRTRD	-
GETSWITCH	diese und weitere BS2000 Makroaufrufe
BREAK	werden im vordefinierten Paket
CMD	BS2000CALLS angeboten
DATE	werden vom vordefinierten Paket
TIME	CLOCK in modifizierter Form angeboten
COMPDATE	-
VERSION	-
ASSEXT	-
LINK	-
LINKCALL	-
STXIT	wird nicht benötigt
FINALIZE	wird nicht benötigt

Compiler-Listen

Für ein Beispielprogramm werden die verschiedenen Compiler-Listen erzeugt. Auf die Wiederholung des Prologs und der Übersetzungsliste wird bei der Assemblerliste, Querverweisliste und Adreßliste verzichtet. Die Fehlerliste entstand durch Einbau einiger Fehler in das Beispielprogramm.

Das Beispielprogramm benutzt das Paket TOOLS aus der PLAM-Bibliothek PLAM.SUPPORT. Die Funktion hex dieses Pakets liefert für eine Integerzahl den Hexadezimalwert als Zeichenkette zurück.

```
*** SOURCE LISTING ***    BS2000 SIEMENS PASCAL-XT COMPILER  V2.2A...


GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG       =    OFF          BY DEFAULT
GENERATE    =    ON           BY DEFAULT
MAP         =    OFF          BY DEFAULT
STANDARD    =    OFF          BY DEFAULT
XREF        =    OFF          BY DEFAULT


LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)

               ($USERID.PLAM.MANUAL,TOOLS.SPEC(*STD,S))



CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.MANUAL,BEISPIEL.PROG(*STD,S))


     1      with TOOLS;
     2
     3      program BEISPIEL (input,output);
     4
     5      const
     6            txt = 'Hexadezimalwert = #';
     7
     8      var
     9            i : integer;
    10
    11      begin
    12         repeat
    13            read    (input, i);
    14            writeln (output, txt, TOOLS.hex(i));
    15         until i = 0;
    16      end (* BEISPIEL *).


************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :        0                     *
*  WARNINGS                 :        0                     *
*  NOTES                    :        0                     *
*  SIZE OF CODE MODULE      :      540 BYTES               *
*  SIZE OF DATA MODULE      :      104 BYTES               *
*  COMPILATION TIME         :    0.142 SEC                 *
************************************************************
```

```
*** SOURCE LISTING ***    BS2000 SIEMENS PASCAL-XT COMPILER  V2.2A...


GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT


CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.MANUAL,FEHLER.PROG(*STD,S))

     1      {$Check=Of}
           ──────────────1
>>> 1: NOTE       3: INVALID ARGUMENT NAME OF OPTION
>>>    NOTE       5: OPTION IGNORED

     2      prgram FEHLER (input);
           ─────────────1
>>> 1: ERROR    723: REPLACED BY "PROGRAM"

     3
     4      const
     5         zugross = maxint + 1;
           ──────────────────────────────1
>>> 1: ERROR    404: NUMERIC ERROR IN COMPUTATION

     6
     7      var
     8         i,j : integer;
           ─────────────────1
>>> 1: NOTE      50: IDENTIFIER IS DECLARED BUT NOT USED

     9
    10      begin
    11         for i := 1 to 10 do
    12            writeln (output, 2 ** i);
           ────────────────────1─────────2
>>> 1: ERROR    361: STANDARD FILE MUST BE PROGRAM PARAMETER
>>> 2: ERROR    302: UNDECLARED OR INVISIBLE IDENTIFIER

    13         i := maxint + 1;
           ──────────────────────────1
>>> 1: WARNING 3404: NUMERIC ERROR IN COMPUTATION

    14      end (* FEHLER *).


************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :        4                     *
*  WARNINGS                 :        1                     *
*  NOTES                    :        3                     *
*  SIZE OF CODE MODULE      :        0 BYTES               *
*  SIZE OF DATA MODULE      :        0 BYTES               *
*  COMPILATION TIME         :    0.070 SEC                 *
************************************************************
```

```
** ASSEMBLER LISTING *    BS2000 SIEMENS PASCAL-XT COMPILER  V2.0A...

     PROCEDURE: BEISPIEL

     LOCTN   OBJECT CODE     ADDR1  ADDR2    STMNT    ASSEMBLY CODE

     000098  C2C5C9E2D7C9C5D3                   1            DC    C'BEISPIEL'
     0000A0  00000088                           2            DC    F'136'
     0000A4  00000238                           3            DC    A(START+568)
     0000A8  0000018E                           4            DC    F'398'
     0000AC  00000000                           5            DC    F'0'
     0000B0  00000000                           6            DC    A(START+0)
     0000B4  50 C0 B030                         7            ST    12,48(0,11)
     0000B8  50 90 B0AC                         8            ST    9,172(0,11)
     0000BC  50 A0 B0B0                         9            ST    10,176(0,11)
     0000C0  50 B0 B0B4                        10            ST    11,180(0,11)
     0000C4  50 D0 B034                        11            ST    13,52(0,11)
     0000C8  58 30 A018      0000B0            12            L     3,24(0,10)
     0000CC  58 10 308C                        13            L     1,140(0,3)
     0000D0  58 F0 1028                        14            L     15,40(0,1)
     0000D4  58 10 D05C                        15            L     1,92(0,13)
     0000D8  50 10 B068                        16            ST    1,104(0,11)
     0000DC  58 D0 D05C                        17            L     13,92(0,13)
     0000E0  45 E0 9030                        18            BAL   14,48(0,9)
     0000E4  58 D0 B034                        19            L     13,52(0,11)
     0000E8  58 30 A018      0000B0            20            L     3,24(0,10)
     0000EC  58 10 3090                        21            L     1,144(0,3)
     0000F0  58 F0 1028                        22            L     15,40(0,1)
     0000F4  58 20 D060                        23            L     2,96(0,13)
     0000F8  50 20 B06C                        24            ST    2,108(0,11)
     0000FC  58 D0 D060                        25            L     13,96(0,13)
     000100  45 E0 9030                        26            BAL   14,48(0,9)
     000104  58 D0 B034                        27            L     13,52(0,11)
     000108  58 30 A018      0000B0            28            L     3,24(0,10)
     00010C  58 00 D030                        29            L     0,48(0,13)
     000110  5A 00 90C8                        30            A     0,200(0,9)
     000114  50 00 D030                        31            ST    0,48(0,13)
     000118  5B 00 90C0                        32            S     0,192(0,9)
     00011C  50 00 B048                        33            ST    0,72(0,11)
     000120  58 30 3094                        34            L     3,148(0,3)
     000124  50 30 B07C                        35            ST    3,124(0,11)
     000128  58 F0 3028                        36            L     15,40(0,3)
     00012C  58 D0 D064                        37            L     13,100(0,13)
     000130  05 E9                             38            BALR  14,9
     000132  58 D0 B034                        39            L     13,52(0,11)
     000136  58 30 A018      0000B0            40            L     3,24(0,10)
     00013A  58 10 B068                        41            L     1,104(0,11)
     00013E  58 20 B06C                        42            L     2,108(0,11)
     000142  41 00 1034                        43            LA    0,52(0,1)
     000146  50 00 B080                        44            ST    0,128(0,11)
     00014A  58 10 3080                        45            L     1,128(0,3)
     00014E  50 10 B070                        46            ST    1,112(0,11)
     000152  41 20 2034                        47            LA    2,52(0,2)
     000156  50 20 B074                        48            ST    2,116(0,11)
     00015A  41 30 302C                        49            LA    3,44(0,3)
```

```
00015E  50 30 B078                                  50            ST    3,120(0,11)
                                                    51    ***** LINE 13
000162  50 00 B048                                  52            ST    0,72(0,11)
000166  58 F0 1038                                  53            L     15,56(0,1)
00016A  58 D0 D050                                  54            L     13,80(0,13)
00016E  45 E0 9030                                  55            BAL   14,48(0,9)
000172  58 D0 B034                                  56            L     13,52(0,11)
000176  58 10 B070                                  57            L     1,112(0,11)
00017A  58 20 B074                                  58            L     2,116(0,11)
00017E  58 30 B078                                  59            L     3,120(0,11)
000182  41 00 0013      000013                     60            LA    0,19(0,0)
000186  58 40 B04C                                  61            L     4,76(0,11)
00018A  50 40 D034                                  62            ST    4,52(0,13)
                                                    63    ***** LINE 14
00018E  50 20 B048                                  64            ST    2,72(0,11)
000192  50 30 B04C                                  65            ST    3,76(0,11)
000196  50 00 B050                                  66            ST    0,80(0,11)
00019A  50 00 B054                                  67            ST    0,84(0,11)
00019E  58 F0 1058                                  68            L     15,88(0,1)
0001A2  58 D0 D050                                  69            L     13,80(0,13)
0001A6  45 E0 9030                                  70            BAL   14,48(0,9)
0001AA  58 D0 B034                                  71            L     13,52(0,11)
0001AE  58 30 B07C                                  72            L     3,124(0,11)
0001B2  58 20 D034                                  73            L     2,52(0,13)
0001B6  41 10 B060                                  74            LA    1,96(0,11)
0001BA  58 F0 302C                                  75            L     15,44(0,3)
0001BE  58 D0 D064                                  76            L     13,100(0,13)
0001C2  05 E9                                       77            BALR  14,9
0001C4  58 D0 B034                                  78            L     13,52(0,11)
0001C8  58 10 B070                                  79            L     1,112(0,11)
0001CC  58 20 B074                                  80            L     2,116(0,11)
0001D0  41 00 B060                                  81            LA    0,96(0,11)
0001D4  41 30 0008      000008                      82            LA    3,8(0,0)
0001D8  50 20 B048                                  83            ST    2,72(0,11)
0001DC  50 00 B04C                                  84            ST    0,76(0,11)
0001E0  50 30 B050                                  85            ST    3,80(0,11)
0001E4  50 30 B054                                  86            ST    3,84(0,11)
0001E8  58 F0 1058                                  87            L     15,88(0,1)
0001EC  58 D0 D050                                  88            L     13,80(0,13)
0001F0  45 E0 9030                                  89            BAL   14,48(0,9)
0001F4  58 D0 B034                                  90            L     13,52(0,11)
0001F8  58 10 B070                                  91            L     1,112(0,11)
0001FC  58 20 B074                                  92            L     2,116(0,11)
000200  50 20 B048                                  93            ST    2,72(0,11)
000204  58 F0 105C                                  94            L     15,92(0,1)
000208  58 D0 D050                                  95            L     13,80(0,13)
00020C  45 E0 9030                                  96            BAL   14,48(0,9)
000210  58 D0 B034                                  97            L     13,52(0,11)
000214  58 00 B080                                  98            L     0,128(0,11)
000218  58 10 B070                                  99            L     1,112(0,11)
00021C  58 20 D034                                 100            L     2,52(0,13)
000220  12 22                                     101            LTR   2,2
000222  47 70 A0CA      000162                     102            BC    7,202(0,10)
000226  58 E0 B038                                 103            L     14,56(0,11)
00022A  58 90 B024                                 104            L     9,36(0,11)
00022E  58 A0 B028                                 105            L     10,40(0,11)
000232  58 B0 B02C                                 106            L     11,44(0,11)
000236  07 FE                                     107            BCR   15,14
```

```
*** XREF LISTING ***      BS2000 SIEMENS PASCAL-XT COMPILER  V2.0A...

hex
   FUNCTION ... ARRAY ........................... AT     7 IN TOOLS

       14

i
   VARIABLE ... INTEGER ......................... AT     9

       13    14    15

integer
   TYPE ....... INTEGER .... 4 .................. PREDEFINED

        9

read
   PROCEDURE .................................... PREDEFINED

       13

TOOLS
   PACKAGE ...................................... AT     1

       14

txt
   CONSTANT ... ARRAY ........................... AT     6

       14

writeln
   PROCEDURE .................................... PREDEFINED

       14


*** MAP LISTING ***       BS2000 SIEMENS PASCAL-XT COMPILER  V2.0A...

                    PROCEDURE ENTRY VECTOR

       PEV-ADDRESS      MODULE-OFFSET        PROCEDURE / FUNCTION
      40 (00000028)       0 (00000000)   INITIAL PROCEDURE

                    GLOBAL CONSTANTS OF THE UNIT

     MODULE-OFFSET     TYPE       NAME          VALUE
      44 (0000002C)   STRING   txt             'Hexadezimalwert = #'

                    GLOBAL VARIABLES OF THE UNIT

      52 (00000034)   i
```

Vordefinierte Pakete

Zusammen mit dem Pascal-XT Programmiersystem werden dem Benutzer eine Reihe von vordefinierten Paketen angeboten, die häufig benutzte Funktionen und Prozeduren zur Verfügung stellen. Diese Pakete können vom Anwender wie selbst geschriebene Pakete verwendet werden.

Die Spezifikationen der vordefinierten Pakete werden in der PLAM-Bibliothek **PASSUP-XT** ausgeliefert, der Objektcode der Pakete ist in der Bibliothek des Laufzeitsystems PASLIB-XT enthalten. Die vordefinierten Pakete für UTM-Anwendungen sind in [11] beschrieben.

Hinweise

Die vordefinierten Pakete sind nicht portabel.

Das vordefinierte Paket CLOCK ist im Sprachmanual [1] beschrieben, da es von allen Pascal-XT Implementierungen zur Verfügung gestellt wird. Die Spezifikation des Pakets ist in der Bibliothek PASSUP-XT, die Implementierung in der Bibliothek des Laufzeitsystems PASLIB-XT enthalten.

Die Paketnamen der vordefinierten Pakete dürfen in einem Anwenderprogramm nicht als Programm- bzw. Paketnamen verwendet werden, wenn diese vordefinierten Pakete verwendet werden sollen.

Damit die vordefinierten Pakete benutzt werden können, müssen Sie in der eingestellten Projektdatei bekannt gemacht werden. Das geschieht einfach durch Übersetzung der Paket-Spezifikationen der gewünschten Pakete. Jede Paket-Spezifikation ist in einem Bibliothekselement (von PASSUP-XT) abgespeichert, dessen Elementname aus dem Paketnamen und dem Suffix ".SPEC" gebildet wird.

Beispiel

```
/EXEC $PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//DEFINE DIRECTORY
//MC ($PASSUP-XT,), *DUMMY
//C (,DMSIO.SPEC)
//C (,BS2000CALLS.SPEC)
```

BS2000CALLS

Das Paket BS2000CALLS stellt BS2000 spezifische Makroaufrufe zur Verfügung.

```
(************************************************************************
*                                                                       *
*       COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990       *
*                     ALL RIGHTS RESERVED                               *
*                                                                       *
************************************************************************)

{$DEBUG=ON}
package BS2000CALLS;


            (*********************************************)
            (*  The body of this package is part of the     *)
            (*  Pascal-XT runtime system.                   *)
            (*                                              *)
            (*  Required Pascal-XT Runtime System: >= 2.1A  *)
            (*  Available in V2.1A and higher:              *)
            (*     CREATE_CMD_BUFFER                        *)
            (*     RELEASE_CMD_BUFFER                       *)
            (*     RESET_CMD_BUFFER                         *)
            (*     CMDX                                     *)
            (*     GET_CMD_LINE                             *)
            (*********************************************)

const

   batch_task    =  0;


type

   switches      = set of 0 .. 31;
   ret_code      = packed array [ 1..4 ] of char;
   byte          = 0..255;
   cmdstring     = string [254];
   name8         = packed array [ 1..8 ] of char;

   unpacked_decimal = packed array [ 1..4 ] of char;

   taskinfo = packed record
      tasktype    (0):  byte;
      bufsize     (1):  short_integer;
      priority    (3):  byte;
      tsn         (4):  unpacked_decimal;
      user_id     (8):  name8;
      account    (16):  name8;
      time       (24):  long_integer;
      privilege  (28):  byte;
      line_length(29):  byte;
      station    (30):  byte;
      prog_name  (32):  name8;
      logon_name (40):  name8;
      end;


function tmode : taskinfo;
   (* returns attributes of the user process                        *)

procedure break;
   (* interrupts the program currently running                      *)

procedure cmd (     command: string;
               var errors: boolean );
   (* the (BS2000-) command is executed; errors returns true, if an *)
   (* error occurred, otherwise false.                              *)
```

```
function get_switches : switches;
   (* returns the currently set process switches                    *)

procedure set_switches     ( s: switches );
   (* switch i is set if element (31 - i) is set in s               *)

procedure reset_switches   ( s: switches );
   (* switch i is reset if element (31 - i) is set in s             *)

procedure set_return_code  ( c: ret_code );
   (* return code c will be set at program termination              *)

function  return_code: ret_code;
   (* returns the return code set by the last call of               *)
   (* set_return_code                                               *)

procedure create_cmd_buffer ( var buffer : pointer );
   (* creates a new buffer of 32 KB and returns the pointer         *)
   (* of this buffer; returns nil, if allocation failed             *)

procedure release_cmd_buffer ( var buffer : pointer );
   (* returns the buffer of 32 KB to the BS2000-System              *)
   (* buffer is set to NIL                                          *)

procedure reset_cmd_buffer ( buffer : pointer );
   (* after this procedure was called the buffer@ can be read       *)
   (* again from the beginning                                      *)

procedure cmdx (     command: string;
                     sysout:  boolean;
                     buffer:  pointer;
                var errors:  boolean );
   (* Precondition:
   (* command contains the bs2000-command; sysout contains true, if *)
   (* the output of the command shall be directed both to SYSOUT    *)
   (* and buffer@, sysout contains false, if the output shall be    *)
   (* directed only to buffer@. The information (strings) in        *)
   (* buffer@ can be read by means of procedure GET_CMD_LINE;       *)
   (* Postcondition:                                                *)
   (* error = false  the command was executed and the return        *)
   (*                information is available in buffer@            *)
   (*       = true   an error occurred, the contents of buffer@ is  *)
   (*                undefined.                                     *)

procedure get_cmd_line (     buffer: pointer;
                         var line:   cmdstring;
                         var more:   boolean );
   (* Precondition:                                                 *)
   (* buffer@ contains the value returned by a previous call of CMDX*)
   (* Postcondition:                                                *)
   (* the first call of GET_CMD_LINE after a call of CMDX returns   *)
   (* the first line of buffer@, the following calls of GET_CMD_LINE*)
   (* return the next lines of buffer@ until the end of the         *)
   (* information in buffer@ is reached. In this case the empty     *)
   (* string is returned. The next call of GET_CMD_LINE again       *)
   (* returns the first line of buffer@. More is true, if the       *)
   (* buffer's line is longer than cmdstring. With following calls  *)
   (* of GET_CMD_LINE the remainder of the line can be read.        *)

end (* package BS2000CALLS *).
```

TMODE

liefert Statusinformationen über den Benutzerprozeß zurück.

BREAK

bewirkt eine Programmunterbrechung und einen Übergang in den BS2000-Kommandomodus. Die Ausführung des Programms kann mit dem BS2000-Kommando RESUME fortgesetzt werden.

CMD (command, errors)

ermöglicht die Ausführung eines BS2000-Kommandos von einem Pascal-Programm aus. Der in command angegebene String wird als BS2000-Kommando interpretiert (Makro-Aufruf CMD). Die boolesche Variable errors liefert den Wert "True" zurück, wenn das Kommando fehlerhaft ausgeführt wurde, ansonsten hat errors den Wert "False". In command können Groß- und Kleinbuchstaben angegeben werden.

Vorsicht ist geboten bei Kommandos wie EXEC, die das Pascal-Programm überladen und damit das Programm beenden.

GET_SWITCHES

liefert die Menge der momentan gesetzten Prozeßschalter. Die Reihenfolge der Schalter ist gegenüber den Beschreibungen in [6] und [7] vertauscht, so daß Schalter i gesetzt ist, wenn (31-i) in der zurückgegebenen Menge gesetzt ist.

SET_SWITCHES (s)

setzt die in der Menge s enthaltenen Prozeßschalter. Die Schalter sind von 0 bis 31 durchnumeriert. Die Reihenfolge ist gegenüber den Beschreibungen in [6] und [7] vertauscht, so daß Schalter i gesetzt wird, wenn (31-i) in s gilt.

RESET_SWITCHES (s)

setzt die in der Menge s enthaltenen Prozeßschalter zurück. Die Schalter sind von 0 bis 31 durchnumeriert. Die Reihenfolge ist gegenüber den Beschreibungen in [6] und [7] vertauscht, so daß Schalter i zurückgesetzt wird, wenn (31-i) in s gilt.

SET_RETURN_CODE (code)

übernimmt code als Beendigungscode für den beim Programmende auszuführenden TERM-Makro (siehe [6]). Werden in einem Programm mehrere Aufrufe von SET_RETURN_CODE ausgeführt, so wird der im letzten Aufruf angegebene Code als Beendigungscode im TERM-Makro verwendet. Wurde beim Aufruf des Programms eine Jobvariable zur Programmüberwachung angegeben, dann wird code beim Beenden dieses Programms in diese Jobvariable übernommen.

RETURN_CODE

liefert den Beendigungscode, der zuletzt mit SET_RETURN_CODE gesetzt wurde.

CREATE_CMD_BUFFER (buffer)

Diese Prozedur fordert Speicher in der Größe von 32 KB an. Dies ist die max. Größe des Ergebnisbereiches für das SYSOUT-Protokoll beim Aufruf des CMD-Makros.

buffer Enthält nach Ausführung von CREATE_CMD_BUFFER die Adresse des Ergebnisbereiches. Kann die Speicheranforderung nicht befriedigt werden, so enthält buffer den Wert NIL.

RELEASE_CMD_BUFFER (buffer)

Bei Aufruf dieser Prozedur wird der mit CREATE_CMD_BUFFER angeforderte Speicherbereich an das Betriebssystem zurückgegeben.

buffer Enthält die Adressse des zurückzugebenden Ergebnisbereiches und wird anschließend auf NIL gesetzt.

RESET_CMD_BUFFER (buffer)

Mit dieser Prozedur wird auf den Anfang des Puffers positioniert, so daß die Zeilen erneut ausgelesen werden können.

buffer Enthält die Adresse des Ergebnisbereiches.

CMDX (command, sysout, buffer, errors)

Diese Prozedur ermöglicht das Absetzen eines BS2000-Kommandos mit anschließender Übernahme des SYSOUT-Protokolls in den mit CREATE_CMD_BUFFER angeforderten Puffer. Das SYSOUT-Protokoll wird in einer nicht aufbereiteten Form zurückgegeben, deshalb sollte der Zugriff auf diese Informationen ausschließlich über die Prozedur GET_CMD_LINE erfolgen.

command
: Enthält das abzusetzende BS2000-Kommando.

sysout
: Ist sysout TRUE, so erfolgt eine zusätzliche Ausgabe des Protokolls (durch das Systemmakro) auf die Systemdatei SYSOUT.

buffer
: Puffer, in der die vom Systemmakro cmd gelieferten Informationen abgelegt werden. Dieser Puffer muß zuvor mit CREATE_CMD_BUFFER beschafft werden.

errors
: Ist TRUE, wenn beim Absetzen des Kommandos ein Fehler auftrat, ansonsten FALSE.

GET_CMD_LINE (buffer, line, more)

Mit dieser Prozedur können die Zeilen aus dem Puffer zeilenweise ausgelesen werden, ohne die Betriebssystem-spezifische Block-/Satz-Struktur kennen zu müssen. Nach einem Aufruf von CMDX liefert jeder Aufruf von GET_CMD_LINE die nächste Zeile aus dem Puffer, d.h.

- der erste GET_CMD_LINE-Aufruf liefert die erste Zeile
- der zweite GET_CMD_LINE-Aufruf liefert die zweite Zeile

usw.

Ist keine Zeile mehr vorhanden, dann liefert GET_CMD_LINE den Leerstring zurück und positioniert wieder an den Pufferanfang, so daß mit dem nächsten GET_CMD_LINE wieder die erste Zeile gelesen werden kann. Ist eine vom Systemmakro cmd gelieferte Zeile länger als Maxlength (line), dann enthält line den ersten Teil der Zeile und die Variable more wird auf TRUE gesetzt.

Der folgende Aufruf von GET_CMD_LINE liefert den nächsten Teil der Zeile. more bekommt den Wert FALSE, wenn line den Rest der Zeile enthält.

buffer Ist der Puffer, der bei einem vorherigen CMDX-Aufruf angegeben wurde.

line Enthält nach Aufruf von GET_CMD_LINE die nächste Zeile des Puffers (als Pascal-String).

more = TRUE die eingelesene Zeile passt nicht vollständig in line. Der Rest kann mit weiteren GET_CMD_LINE Aufrufen gelesen werden.
= FALSE Zeile vollständig in line enthalten.

DMSIO

Das Paket DMSIO stellt zusätzliche Funktionen zum Eröffnen und Schließen von Dateien und eine Reihe von Zugriffsfunktionen für ISAM-Dateien (index-sequentielle Dateien) zur Verfügung.

Beim Eröffnen einer Datei zum Lesen übernimmt das Pascal-System die aktuellen Dateiattribute aus dem Katalogeintrag (bei nicht existierender Datei wird ein OPEN_ERROR erzeugt). Wurde vor dem Eröffnen ein FILE-Kommando mit Angabe des Linknamens und der Dateiattribute abgesetzt, dann wird ein OPEN_ERROR erzeugt, wenn diese Attribute nicht mit denen im Katalog übereinstimmen (siehe auch 5.3).

Beim Eröffnen einer Datei zum Schreiben richtet das Pascal-System standardmäßig eine SAM-Datei mit variabler Satzlänge ein. Soll die aktuelle Datei eine SAM-Datei mit fester Satzlänge oder eine ISAM-Datei sein, dann muß vor dem Eröffnen der Datei ein FILE-Kommando mit den gewünschten Dateiattributen abgesetzt werden. Im FILE-Kommando muß außerdem der Name der Pascal-Datei als Linkname angegeben werden, damit die Dateiattribute in den Katalogeintrag übernommen werden (siehe auch 5.3). Die mit "(*)" gekennzeichneten Attribute müssen für eine ISAM-Datei unbedingt im FILE-Kommando angegeben werden, für die anderen Attribute werden die Standardwerte (siehe [7]) eingesetzt. Aus Gründen der Zuverlässigkeit und der Dokumentation sollte auch das Attribut RECFORM stets angegeben werden, da der Wert von KEYPOS von der Satzform abhängig ist.

FCBTYPE (*) Als Zugriffsmethode ist ISAM anzugeben.

KEYPOS (*) Gibt die Position des ersten Zeichens vom Satzschlüssel innerhalb des Satzes an. In einer Pascal-Datei entspricht der Satzschlüssel einem Recordfeld im Basistyp des Dateityps. Dem relativen Abstand n des Recordfeldes zum Recordanfang entspricht bei fester Satzlänge (RECFORM=F) die Position n+1, bei variabler Satzlänge (RECFORM=V) die Position n+5, da hier noch das 4 Byte lange Satzlängenfeld zu berücksichtigen ist.

Der relative Abstand des Recordfeldes, das den Schlüssel enthält, sollte mit der vordefinierten Funktion OFFSETOF bestimmt werden (siehe auch Beispiele).

KEYLEN (*) Gibt die Länge des Satzschlüssels in Byte an. Die Länge des Recordfeldes, das den Satzschlüssel definiert, kann mit der Standardfunktion SIZEOF ermittelt werden.

RECFORM	Legt fest, ob die Sätze mit fester oder variabler Länge verarbeitet werden. Variabel lange Sätze werden automatisch mit einem 4 Byte Satzlängenfeld versehen.
RECSIZE	Gibt die Satzlänge (inclusive Längenfeld) in Bytes an.
BLKSIZE	Definiert die Pufferlänge für die Ein-/Ausgabe.
DUPEKY	Legt fest, ob doppelte Schlüssel zulässig sind. Dieses Attribut wird nicht im Katalog abgespeichert und gilt nur solange, wie das zugehörige FILE-Kommando Gültigkeit hat.
SHARUPD	Legt fest, ob eine Datei von mehreren Prozessen gleichzeitig geändert werden kann, wenn die Datei mit Replace eröffnet wurde. Näheres über die Zugriffs-Synchronisation ist in [7,8] enthalten. Dieses Attribut wird nicht im Katalog abgespeichert und gilt nur solange, wie das zugehörige FILE-Kommando Gültigkeit hat.

```
(*************************************************************************
*                                                                        *
*       COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990        *
*                     ALL RIGHTS RESERVED                                *
*                                                                        *
*************************************************************************)

 {$DEBUG=ON}
 package DMSIO;

             (*************************************)
             (*  The body of this package is part   *)
             (*  of the Pascal-XT runtime system    *)
             (*************************************)

 procedure extend     (var f: any_file);
    (* same as rewrite but the file's content is not lost *)

 procedure replace    (var f: any_file);
    (* opens the file for reading and writing *)

 procedure close      (var f: any_file);
    (* closes the file *)

 procedure elim       (var f: any_file);
    (* deletes the last read record in the file *)

 procedure elimkey    (var f: any_file);
    (* deletes the record with the given key *)

 procedure getkey     (var f: any_file);
    (* reads the record with the specified key *)

 procedure getback    (var f: any_file);
    (* reads backwards the next record *)

 procedure update     (var f: any_file);
    (* writes a previously read record back into the file *)

 procedure pos        (var f: any_file);
    (* positions to the beginning of the file *)

 procedure poskey     (var f: any_file);
    (* positions to the record with the specified key *)

 procedure posback    (var f: any_file);
    (* positions to the file's end *)

 procedure movekey    (var f: any_file; var stv: string);
    (* returns the key of the last read record *)

 function  bof        (var f: any_file): boolean;
    (* returns begin of file *)

 function  nokey      (var f: any_file): boolean;
    (* returns false if the specified record is in the file *)

 function  lock       (var f: any_file): boolean;
    (* is true if the specified record is locked *)

 function  reclen     (var f: any_file): integer;
    (* returns the length of the last read record *)

 function  keypos     (var f: any_file): integer;
    (* returns the file key's position *)

 function  keylen     (var f: any_file): integer;
    (* returns the file's keylength *)

 end (* package DMSIO *).
```

Eröffnungsmodi für Dateien

CLOSE (f)

Bei Programmende werden automatisch alle von Pascal eröffneten Dateien geschlossen. Der Benutzer kann zusätzlich eine Datei durch Close(f) selbst schließen. Bei einer Textdatei wird bei einem Close(f) ein implizites Writeln (f) ausgeführt, wenn die letzte Zeile noch nicht mittels Writeln (f) abgeschlossen wurde. Bei Nicht-Textdateien geht der Inhalt eines nicht ausgegebenen Dateipuffers verloren. Wird Close auf die vordefinierte Datei Input oder Output angewendet, dann tritt ein Open_Error mit dem Systemfehlercode 1607 auf.

EXTEND (f)

eröffnet die Datei zum Schreiben. Der bisherige Inhalt der Datei geht nicht verloren. Der Puffer f↑ ist undefiniert. Mittels Put können weitere Sätze am Ende der Datei abgelegt werden. Extend kann nur auf SAM- und ISAM-Dateien und die Standard-Dateien *SYSOUT, *SYSLST und *DUMMY angewandt werden. Ein Aufruf von Extend mit Angabe von Input oder Output führt zu einem Open_Error mit dem Systemfehlercode 1607.

REPLACE (f)

eröffnet die Datei zum Lesen und Schreiben, so daß Sätze geändert werden können. Existiert die Datei nicht, dann wird sie neu eingerichtet. Wenn die Datei existiert, dann wird der erste Satz in den Dateipuffer gelesen und ist damit bei ISAM-Dateien mit SHAREUPD=YES gesperrt. Die Sperre kann durch einen Aufruf von Pos (f) aufgehoben werden. Replace kann nur auf SAM- und ISAM-Dateien angewandt werden. Ein Aufruf von Replace mit Angabe von Input oder Output führt zu einem Open_Error mit dem Systemfehlercode 1607.

Allgemeine Funktionen

BOF (f)

zeigt an, ob der Anfang einer Datei f erreicht wurde. Diese Funktion verhält sich symmetrisch zu Eof. Wird Bof nach einem Aufruf von Getback oder Eof True, dann ist ein weiteres Getback verboten, während ein Get erlaubt ist.

POS (f)

positioniert auf den Dateianfang, so daß mit einem darauffolgenden Get(f) der erste Satz der Datei gelesen wird. Nach Pos(f) ist Bof(f) True. Eof(f) ist bei einer leeren Datei True, sonst False.

POSBACK (f)

positioniert an das Dateiende, so daß mit einem darauffolgenden Getback(f) der letzte Satz der Datei gelesen werden kann. Nach Posback(f) ist Eof(f) True. Bof(f) ist bei einer leeren Datei True, sonst False.

RECLEN (f)

Reclen liefert als Integer-Ergebnis die Satzlänge des zuletzt gelesenen oder geschriebenen Satzes der Datei f. Bei undefiniertem Dateistatus wird der Wert -1 zurückgeliefert.

UPDATE (f)

schreibt einen mit Get(f), Getkey(f) oder Getback(f) bereitgestellten Satz wieder in die Datei zurück. Dabei dürfen die Satzlänge und bei ISAM-Dateien das Schlüsselfeld nicht verändert werden, wenn die Datei f mit Replace eröffnet wurde. Die Sperrung des Satzes wird aufgehoben. Update kann nur angewendet werden, wenn die Datei f mit Replace eröffnet wurde.

Bearbeitung von ISAM-Dateien

ELIM (f)

löscht den zuletzt mit Get(f) gelesenen Satz aus einer ISAM-Datei und hinterläßt f↑ undefiniert. War dies der letzte Satz der Datei, so wird Eof(f) True, war es der erste, so bleibt Bof(f) True.

ELIMKEY (f)

löscht den Satz, dessen Schlüssel im Schlüsselfeld steht, aus einer ISAM-Datei. Falls dieser Satz gesperrt ist, wird er nicht gelöscht, sondern Lock(f) ist True. Falls so ein Satz nicht vorhanden ist, ist nach dem Aufruf Nokey(f) True. Falls es mehrere Sätze mit dem gleichen Schlüssel gibt, wird der erste gelöscht. Nach Elimkey (f) sind Eof(f) und Bof(f) False.

GETBACK (f)

liest den nächsten Satz in Richtung Dateianfang aus der Datei. War der zuvor gelesene Satz der erste in der Datei, so ist f↑ undefiniert und Bof(f) wird True. Ist der Satz gesperrt, d.h. Lock (f) True, dann ist f↑ undefiniert und ein erneutes Getback versucht den gesperrten Satz nochmals zu lesen.

GETKEY (f)

liest den Satz, dessen Schlüssel im Schlüsselfeld angegeben ist. Falls dieser Satz gesperrt ist, ist f↑ undefiniert und Lock(f) True. Falls so ein Satz nicht vorhanden ist, ist nach dem Aufruf Nokey(f) True. Bei mehreren Sätzen mit dem gleichen Schlüssel wird der erste dieser Sätze gelesen und die anderen Sätze können mit Get gelesen werden. Falls die Datei mit Replace eröffnet wurde, ist nach einem erfolgreichen Getkey(f) der Satz gesperrt, bei Reset nicht. Getkey(f) setzt Eof(f) und Bof(f) auf False.

KEYLEN (f)

gibt für eine ISAM-Datei die Länge des Satzschlüssels in Byte an.

KEYPOS (f)

gibt für eine ISAM-Datei die Position des ersten Zeichens vom Satzschlüssel an.

LOCK (f)

ist True und f↑ ist undefiniert, wenn der gewünschte Satz gerade gesperrt ist. Bei Wiederholung der Zugriffsfunktion wird wieder auf den gesperrten Satz zugegriffen. Bei ISAM-Dateien mit gleichen Schlüsseln (DUPEKY=YES) kann nicht sichergestellt werden, daß wieder auf denselben Satz zugegriffen wird (wird durch das Verhalten des Retry-Makros bestimmt).

MOVEKEY (f,stv)

liefert bei lesendem Zugriff auf ISAM-Textdateien den zuletzt gelesenen Satzschlüssel der Datei f in die Stringvariable stv. Bei sequentiellen Dateien oder undefiniertem Dateizustand wird ein leerer String zurückgeliefert.

NOKEY (f)

ist True, wenn der gewünschte Satz in der Datei f nicht vorhanden ist.

Hinweis

Nach Poskey(f) ist Nokey auch dann False, wenn der Satz nicht vorhanden ist. Ist die Datei mit Replace eröffnet worden, dann ist vor dem Aufruf von Nokey zuerst abzufragen, ob der Satz gesperrt ist (Aufruf von Lock).

POSKEY (f)

positioniert auf den Satz, dessen Schlüssel im Schlüsselfeld steht, so daß dieser mit dem nächsten Get(f) gelesen werden kann. Falls es so einen Satz nicht gibt, wird auf die Stelle positioniert, an der so ein Satz stehen müßte. Mit einem darauffolgenden Get(f) wird der erste Satz mit dem nächsthöheren Schlüssel gelesen. Nach Poskey(f) sind Eof(f), Bof(f), Lock(f) und Nokey(f) False.

Erweiterungen der Standard-Prozeduren

Zusätzlich zu diesen Prozeduren und Funktionen sind die in Pascal-XT definierten Standard-Prozeduren und -Funktionen auf ISAM-Dateien anwendbar.
Die Prozeduren Put, Get und die Funktion Eof werden für ISAM-Dateien folgendermaßen erweitert:

GET (f)

liest den nächsten Satz in Richtung Dateiende aus der Datei. Falls er gesperrt ist, ist Lock(f) True und f↑ undefiniert. Falls die Datei mit Replace eröffnet ist, wird ein Satz bei erfolgreichem Get gesperrt, bei Reset nicht. Falls kein Satz mehr vorhanden ist, ist f↑ undefiniert und Eof(f) True.

PUT (f)

hängt bei einer SAM-Datei einen Satz an das Ende der Datei an. Bei einer ISAM-Datei, die mit Rewrite oder Extend eröffnet wurde, wird ein Satz ebenfalls am Ende der Datei angehängt. Der Satzschlüssel muß allerdings größer sein als die Schlüssel aller bereits geschriebenen Sätze. Wurde die ISAM-Datei mit Replace eröffnet, dann wird der Satz entsprechend seinem Schlüssel abgelegt. Existiert bereits ein Satz mit dem gleichen Schlüssel, dann wird er bei DUPEKY=NO überschrieben, sonst an die Liste der Sätze mit dem gleichen Schlüssel angehängt. Ist der Satz gesperrt, in den geschrieben werden soll, dann wird keine Aktion ausgeführt und Lock(f) ist True.

EOF (f)

zeigt an, ob das Ende der Datei f erreicht wurde. Wird Eof nach einem Aufruf von Pospack bzw. Get true, dann ist ein weiteres Get verboten, während Getback erlaubt ist.

In der folgenden Tabelle sind alle Aktionsaufrufe bezüglich ihrer Zulässigkeit in Abhängigkeit vom Eröffnungsmodus und Dateiart zusammengefaßt.

Openmodus →	RESET		REWRITE		EXTEND		REPLACE	
Dateiart →	SAM	ISAM	SAM	ISAM	SAM	ISAM	SAM	ISAM
Aktionsaufruf ↓								
ELIM								x
ELIMKEY								x
GET	x	x					x	x
GETKEY		x						x
GETBACK		x						x
PUT			x	(1)	x	(1)		x
UPDATE							x	x
POS	x	x					x	x
POSKEY		x						x
POSBACK	x	x					x	x

x der Aktionsaufruf ist erlaubt

(1) am Ende der Datei erlaubt, d.h. nur mit aufsteigenden Schlüsseln

Die nachfolgende Tabelle bietet einen Überblick über die Beeinflussung der Funktionsergebnisse von Bof, Eof, Nokey und Lock sowie der Pufferinhalte von f↑ durch die Prozeduren.

	BOF	EOF	NOKEY	LOCK	Pufferinhalte
RESET	(1)	(1)	FALSE	FALSE	(A)
REWRITE	-	TRUE	FALSE	FALSE	(U)
EXTEND	-	TRUE	FALSE	FALSE	(U)
REPLACE	(1)	(1)	FALSE	(2)	(A,B)
CLOSE	-	-	FALSE	FALSE	(U)
ELIM	-	-	FALSE	(2)	(U)
ELIMKEY	FALSE	FALSE	(3)	(2)	(U)
GET	FALSE	(4)	FALSE	(2)	(A)
GETBACK	(5)	FALSE	FALSE	(2)	(A,B)
GETKEY	FALSE	FALSE	(3)	(2)	(B)
POS	TRUE	(1)	FALSE	FALSE	(U)
POSBACK	(1)	TRUE	FALSE	FALSE	(U)
POSKEY	FALSE	FALSE	FALSE	FALSE	(U)
PUT	FALSE	-	FALSE	(2)	(U)
UPDATE	-	-	FALSE	FALSE	(U)

– unverändert, nicht relevant.

(1) True, wenn die Datei leer ist, sonst False.

(2) True, falls Satz gesperrt, sonst False.

(3) True, falls Satz nicht existiert, sonst False.

(4) True, falls versucht wird, hinter dem Dateiende zu lesen.

(5) True, falls versucht wird, vor dem Beginn der Datei zu lesen.

(A) undefiniert, falls Eof True, sonst definiert.

(B) undefiniert, falls Lock oder Nokey True, sonst definiert.

(U) undefiniert.

Beispiel 1

Das Beispielprogramm erzeugt eine ISAM-Datei, schreibt Sätze in diese Datei und liest anschließend wieder einen bestimmten Satz. Die Zuweisung der aktuellen Datei an die Pascal-Datei erfolgt vor Ausführung des Programms durch das FILE-Kommando. Beim ersten Aufruf des Programms werden Sätze fester Länge, beim zweiten Aufruf Sätze variabler Länge verarbeitet.

```
/EXEC $USERID.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//D DIRECTORY
//C ($USERID.PASSUP-XT, DMSIO.SPEC), *D
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C (PLAM.EXAMPLE, ISAM1.PROG), *SYSOUT, *OMF(Y)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT

LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)

               ($USERID.PASSUP-XT,DMSIO.SPEC(*STD,S))


CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,ISAM1.PROG(*STD,S))

     1      (*$TITLE = 'Erzeugen einer ISAM-Datei'*)
     2
     3      with DMSIO;
     4
     5      program ISAM1 (output, isamfile);
     6
     7      const
     8         keypos   = 4;
     9
    10      type
    11         satztyp  = record
    12                       c (0)      : char;
    13                       i (keypos) : long_integer;
    14                       end;
    15         dateityp = file of satztyp;
    16
    17      var
    18        isamfile : dateityp;
    19
    20      procedure erzeuge_datei;
    21      begin
    22         rewrite (isamfile);
```

```
    23          isamfile↑.i := 1;
    24          isamfile↑.c := 'A';
    25          put (isamfile);
    26          isamfile↑.i := 2;
    27          isamfile↑.c := 'B';
    28          put (isamfile);
    29       end (* erzeuge_datei *) ;
    30
    31       procedure drucke_satz (key : long_integer);
    32       begin
    33          reset (isamfile);
    34          isamfile↑.i := key;
    35          DMSIO.getkey (isamfile);
    36          if DMSIO.nokey (isamfile) then
    37             writeln ('Satz nicht gefunden')
    38          else
    39             writeln (isamfile↑.i:1, isamfile↑.c:3);
    40       end (* drucke_satz *) ;
    41
    42       begin
    43          erzeuge_datei;
    44          drucke_satz (2);
    45       end (* ISAM1 *).


************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED         :        0                      *
*  WARNINGS                :        0                      *
*  NOTES                   :        0                      *
*  SIZE OF CODE MODULE     :      968 BYTES                *
*  SIZE OF DATA MODULE     :      472 BYTES                *
*  COMPILATION TIME        :    0.210 SEC                  *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//SY FILE DATEI-1,LINK=ISAMFILE,FCBTYPE=ISAM,KEYLEN=4,KEYPOS=5,RECFORM=F  (01)
//RUN                                                                     (02)
2  B                                                                     (03)
//SY FSTAT DATEI-1,ALL                                                  (04)
0000003 :V:$USERID.DATEI-1
  FCBTYPE = ISAM             VSNTYPE = PUB
  LASTPG  = 0000002          2ND ALLO= 00003
  SHARE   = NO               ACCESS  = WRITE
  ACL     = NO               AUDIT   = NONE            DESTROY = NO
  CRDATE  = 1990-11-27       EXDATE  = 1990-11-27      LADATE  = 1990-11-27
  RDPASS  = NONE             WRPASS  = NONE            EXPASS  = NONE
  ACCESS# = 002              VERSION = 001
  LARGE   = NO               BACKUP  = A               MIGRATE = ALLOWED
  BLKTYPE = STD              BLKSIZE = 002048          BLKCTRL = PAMKEY
  RECFORM = (F,N)            RECSIZE = 000008
  KEYLEN  = 004              KEYPOS  = 00005
  VSN/DEV/EXT  =    PUBV02 / D3480    / 001
  EXTCNT  =   1
:V:    PUBLIC:     1  FILE  RES=        3  FREE=        1  REL=        0 PAGES
//SY FILE DATEI-2,LINK=ISAMFILE,FCBTYPE=ISAM,KEYLEN=4,KEYPOS=9,RECFORM=V  (05)
//RUN
2  B
```

```
//SY FSTAT DATEI-2,ALL ──────────────────────────────────────────────── (06)
0000003 :V:$USERID.DATEI-2
  FCBTYPE = ISAM             VSNTYPE = PUB
  LASTPG  = 0000002          2ND ALLO= 00003
  SHARE   = NO               ACCESS  = WRITE
  ACL     = NO               AUDIT   = NONE            DESTROY = NO
  CRDATE  = 1990-11-27       EXDATE  = 1990-11-27      LADATE  = 1990-11-27
  RDPASS  = NONE             WRPASS  = NONE            EXPASS  = NONE
  ACCESS# = 002              VERSION = 001
  LARGE   = NO               BACKUP  = A               MIGRATE = ALLOWED
  BLKTYPE = STD              BLKSIZE = 002048          BLKCTRL = PAMKEY
  RECFORM = (V,N)            RECSIZE = 000012
  KEYLEN  = 004              KEYPOS  = 00009
  VSN/DEV/EXT  =    PUBV04 / D3480    / 001
  EXTCNT  =   1
:V:    PUBLIC:     1  FILE  RES=        3  FREE=        1  REL=        0 PAGES
//
```

(01) Vor Ausführung des Programms wird mit dem FILE-Kommando eine Datei mit Angabe des Linknamens und der Dateiattribute eingestellt. In diesem Fall werden Sätze fester Länge verarbeitet. KEYPOS muß demzufolge den Wert 5 haben.

(02) Ausführen des Programms. Der Name der Pascal-Datei wird als Linkname angenommen.

(03) Meldung des Programms.

(04) Ausgabe der Informationen aus dem Dateikatalog für DATEI-1.

(05) Mit dem FILE-Kommando wird eine neue Datei eingestellt. KEYPOS bekommt den WERT 9, da diesmal Sätze variabler Länge verarbeitet werden.

(06) Ausgabe der Informationen aus dem Dateikatalog für DATEI-2.

Beispiel 2

Das Programm schreibt Sätze in eine Datei, die paarweise denselben Schlüssel besitzen. Anschließend werden alle Sätze eines bestimmten Schlüssels ausgegeben. Der Name der aktuellen Datei wird vom Programm eingelesen. Im Programm wird das FILE-Kommando aufgerufen, um die Parameter der ISAM-Datei einzustellen. Die Zuweisung der aktuellen Datei an die Pascal-Datei erfolgt durch den Aufruf von ASSIGNFILE.

```
/EXEC $USERID.PASCAL-XT
%  BLS0500 PROGRAM 'PASCALXT', VERSION '22A00' OF ...  LOADED<
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG.1990. ALL RIGHT
S RESERVED
//D DIRECTORY
//C ($USERID.PASSUP-XT, DMSIO.SPEC), *D
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C ($USERID.PASSUP-XT, BS2000CALLS.SPEC), *D
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//C (PLAM.EXAMPLE, ISAM2.PROG), *SYSOUT, *OMF(Y)
*** SOURCE LISTING ***    BS2000 PASCAL-XT COMPILER  V2.2A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT

LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)

               ($USERID.PASSUP-XT,BS2000CALLS.SPEC(*STD,S))

               ($USERID.PASSUP-XT,DMSIO.SPEC(*STD,S))


CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.EXAMPLE,ISAM2.PROG(*STD,S))

     1      with BS2000CALLS, DMSIO;
     2
     3      program ISAM2 (input, output, isamfile);
     4
     5      type
     6         keytyp   = long_integer;
     7         satztyp  = record
     8                       c   : char;
     9                       key : keytyp;
    10                       end;
    11         dateityp = file of satztyp;
    12
    13      var
    14        isamfile  : dateityp;
    15        attr      : string;
    16        dateiname : string;
    17        error     : boolean;
    18
```

```
19      procedure init;
20      begin
21         (* Aufbau des FILE-Kommandos ohne Angabe einer Datei *)
22         writestring (attr, '/FILE ',
23                             ',LINK=ISAMFILE',
24                             ',FCBTYPE=ISAM',
25                             ',KEYPOS=', (offsetof (satztyp, key) + 5):1,
26                             ',KEYLEN=', sizeof (keytyp):1,
27                             ',RECFORM=V',
28                             ',DUPEKY=YES');
29         writeln ('Bitte den Dateinamen eingeben:');
30         readln; read (dateiname);
31         BS2000CALLS.cmd (attr, error);
32         if error then raise (1);
33         assignfile (isamfile, dateiname);
34      end (* init *) ;
35
36      procedure erzeuge_datei;
37      begin
38         rewrite (isamfile);
39         isamfile↑.key := 1; isamfile↑.c := 'A'; put (isamfile);
40         isamfile↑.key := 1; isamfile↑.c := 'a'; put (isamfile);
41         isamfile↑.key := 2; isamfile↑.c := 'B'; put (isamfile);
42         isamfile↑.key := 2; isamfile↑.c := 'b'; put (isamfile);
43         isamfile↑.key := 3; isamfile↑.c := 'C'; put (isamfile);
44         isamfile↑.key := 3; isamfile↑.c := 'c'; put (isamfile);
45      end (* erzeuge_datei *) ;
46
47      procedure drucke_satz (value : keytyp);
48      begin
49         reset (isamfile);
50         isamfile↑.key := value;
51         DMSIO.getkey (isamfile);
52         if DMSIO.nokey (isamfile) then
53            writeln ('Satz nicht gefunden')
54         else
55            repeat
56               writeln (isamfile↑.key:1, isamfile↑.c:3);
57               get (isamfile);
58            until (isamfile↑.key <> value) or eof (isamfile);
59      end (* drucke_satz *) ;
60
61      begin
62         init;
63         erzeuge_datei;
64         drucke_satz (2);
65      end (* ISAM2 *).
```

```
************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :         0                     *
*  WARNINGS                 :         0                     *
*  NOTES                    :         0                     *
*  SIZE OF CODE MODULE      :      2368 BYTES               *
*  SIZE OF DATA MODULE      :       988 BYTES               *
*  COMPILATION TIME         :     0.326 SEC                 *
************************************************************
   >>> COMPILATION SUCCESSFUL (WARNINGS: 0;  NOTES: 0)
//RUN
Bitte den Dateinamen eingeben:
TESTDATEI
2  B
2  b
//SY FSTAT TESTDATEI,ALL
0000033 :V:$USERID.TESTDATEI
  FCBTYPE = ISAM             VSNTYPE = PUB
  LASTPG  = 0000002          2ND ALLO= 00032
  SHARE   = NO               ACCESS  = WRITE
  ACL     = NO               AUDIT   = NONE             DESTROY = NO
  CRDATE  = 1990-11-27       EXDATE  = 1990-11-27       LADATE  = 1990-11-27
  RDPASS  = NONE             WRPASS  = NONE             EXPASS  = NONE
  ACCESS# = 002              VERSION = 001
  LARGE   = NO               BACKUP  = A                MIGRATE = ALLOWED
  BLKTYPE = STD              BLKSIZE = 002048           BLKCTRL = PAMKEY
  RECFORM = (V,N)            RECSIZE = 000012
  KEYLEN  = 004              KEYPOS  = 00009
  VSN/DEV/EXT  =    PUBV03 / D3480    / 001
  EXTCNT  =   1
:V:    PUBLIC:     1  FILE  RES=       33  FREE=       31  REL=       30 PAGES
//
```

EDTADAPTER

Das Paket EDTADAPTER ermöglicht einem Programm den Aufruf des Editors EDT bzw. das Senden von Daten und EDT-Kommandos an den EDT.

```
(************************************************************************
*                                                                       *
*       COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990       *
*                     ALL RIGHTS RESERVED                               *
*                                                                       *
************************************************************************)

{$DEBUG=ON}
package EDTADAPTER;


          (*********************************************)
          (*  The body of this package is part of the  *)
          (*  Pascal-XT runtime system.                *)
          (*                                           *)
          (*  Required EDT Version:  >= 16.1           *)
          (*  Required Pascal-XT Runtime System: >= 2.1A  *)
          (*  Available in V2.1A and higher:           *)
          (*     EDT_VERSION                           *)
          (*     EDT_INTERFACE_VERSION                 *)
          (*     SET_EDT_INTERFACE_VERSION             *)
          (*********************************************)


procedure call_edt;
   (* calls the editor EDT for interactive use *)

procedure submit_edt (command: string);
   (* sends a string or an edt-command to the editor edt *)


const
                                (* Constant values defined by EDT *)
   command_string_len = 296;    (* must be equal to EUPCMDM *)
   message_string_len =  80;    (* must be equal to EUPMSGM *)
   workfile_name_len  =   8;    (* must be equal to L'EGLFILE *)
   version_number_len =  12;    (* must be equal to EGLVERSL *)

type
   command_string     = string [ command_string_len ];
   message_string     = string [ message_string_len ];
   workfile_name      = string [ workfile_name_len ];
   version_number     = string [ version_number_len ];


procedure edt_command (     command  : command_string;
                            message  : message_string;
                       var result   : message_string;
                       var workfile : workfile_name );
   (* this procedure executes F-mode commands as specified in      *)
   (* operand 'command'; the string passed in operand 'message'    *)
   (* is displayed in line 23 of the EDT; in case of an error the  *)
   (* EDT's error message is returned in 'result'; the name of the *)
   (* current active workfile is returned in operand 'workfile'.   *)

function edt_version : version_number;
   (* returns the EDT version-number *)
   (* format: 'EDT V16.xAxx'         *)

function edt_interface_version : integer;
   (* returns the EDT subprogram interface version *)
   (* which is currently supported, e.g. 1 or 2    *)
```

```
procedure set_edt_interface_version (    vers   : integer;
                                     var errors : boolean  );
   (* this procedure sets the EDT subprogram         *)
   (* interface version                              *)
   (* 'vers'  : specifies the interface version as   *)
   (*           described in [1] (subprogram control *)
   (*           block EDTUPCB, field EUPVERS)        *)
   (* 'error' : result of the operation              *)
   (*           = true  : an error occured           *)
   (*           = false : no error                   *)

end (* package EDTADAPTER *).
```

CALL_EDT

setzt die Ausführung des Pascal-Programms aus und ruft den EDT auf. Durch die EDT-Kommandos HALT oder @RETURN oder Drücken der K1-Taste wird der EDT verlassen und die Ausführung des Pascal-Programms fortgesetzt. Die Inhalte der EDT-Arbeitsbereiche bleiben erhalten.

SUBMIT_EDT (line)

übergibt den durch line bezeichneten String an den EDT. Die eingestellte EDT-Unterprogramm-Schnittstellenversion legt fest, wie der String zu interpretieren ist (als Daten oder als EDT-Kommando). Nach Bearbeitung des Strings durch den EDT wird zum Aufrufer zurückgekehrt.

EDT_COMMAND (command, message, result, workfile)

Diese Prozedur bedient die neue Unterprogramm-Schnittstelle des EDT ab Version 16.1 (siehe [13]). Diese Schnittstelle ermöglicht die Übergabe mehrerer EDT-Kommandos und den Aufruf des EDT. Zusätzlich kann ein Text mit übergeben werden, der vom EDT in der Meldungszeile ausgegeben wird. Bei Auftreten eines Fehlers wird der Fehlermeldungstext des EDT zurückgeliefert. Bei der Rückkehr aus dem EDT wird auch der Name des verlassenen EDT-Arbeitsbereichs angegeben. In command und message können Groß- und Kleinbuchstaben enthalten sein.

command Enthält eine Folge von F-Modus Kommandos, getrennt durch Strichpunkte, die der Reihe nach abgearbeitet werden. Die erlaubten Kommandos sind im EDT Handbuch beschrieben. Umschaltung des EDT Arbeitsbereiches durch: '$0' ,..., '$9' Aufruf des EDT durch: 'DIALOG'

message Der angegebene String wird in Zeile 23 des EDT ausgegeben. Bei Angabe des Leerstrings ('') erfolgt eine Standard-Ausgabe durch den EDT.

result Treten beim Abarbeiten von command Fehler auf, dann enthält result entweder den Fehlertext des EDT oder - falls dieser nicht vorhanden ist - den Main-Returncode des EDT in der Form
" PASCAL-XT : EDT MAIN-RETURNCODE xxxx", wobei xxxx für den hexadezimalen Main-Returncode steht. Bei fehlerfreiem Ablauf wird der Leerstring geliefert.

workfile Liefert immer den aktuell eingestellten Arbeitsbereich des EDT (in der Form '$0 ', '$1 ',..., '$9 ').

Die Unterprogramme call_edt und submit_edt benutzen ebenfalls die neue Schnittstelle des EDT 16.1 und haben folgende Wirkung:

call_edt: edt_command('DIALOG', '', dummy, dummy)

submit_edt(cmd): edt_command(cmd, '', dummy, dummy)

wobei "dummy" für Stringvariable steht.

EDT_VERSION

Diese Funktion liefert die Versionsnummer des im System befindlichen EDT in der Form 'EDT V16.xAxx' in einem String der Länge 12 zurück. 'x' steht für eine EDT-spezifische nähere Bezeichnung der Version.

EDT_INTERFACE_VERSION

Diese Funktion liefert die Schnittstellenversion der EDT-Unterprogramm-Schnittstelle in einer Variablen vom Typ Integer zurück. Mögliche Werte für die Schnittstellenversion und deren Bedeutung sind der Beschreibung der EDT-Unterprogramm-Schnittstelle [13] zu entnehmen (derzeit kennzeichnet der Wert 1 den Anweisungsvorrat der EDT-Version 16.1, der Wert 2 den Anweisungsvorrat der EDT-Version 16.2).

SET_EDT_INTERFACE_VERSION (vers, errors)

Das Pascal-XT-Laufzeitsystem übernimmt bei der Initialisierung der Schnittstellenversion die Voreinstellung des Makros IEDTUPCB (vgl. [13]). Mit der Prozedur SET_EDT_INTERFACE_VERSION kann die Schnittstellenversion geändert werden. Die Einstellung ist nur vor der Initialisierung des EDT möglich, d. h. es darf zuvor außer EDT_INTERFACE_VERSION, EDT_VERSION oder dieser Prozedur selbst keine andere Prozedur des Pakets EDTADAPTER aufgerufen worden sein. Alle anderen Aufrufe führen bereits zur Initialisierung des EDT. Ebenso ist nach der Verwendung eines EDT-Prozedurbereiches als Pascal-Datei oder nach Aufruf des EDT in der Programmierumgebung bzw. der Testhilfe PATH eine Änderung der Schnittstellenversion nicht mehr möglich.

vers Spezifiziert die gewünschte Schnittstellenversion, die als Ganzzahl (1,2,...) angegeben werden muß. Der Parameter wird von der Prozedur nicht auf Gültigkeit überprüft.

errors Enthält das Ergebnis der ausgeführten Operation.
= FALSE bei fehlerfreiem Ablauf
= TRUE wenn eine Inkonsistenz bei der EDT-Schnittstellenversion auftritt (z.B. EDT V16.1 wird mit Schnittstellenversion 2 aufgerufen), oder wenn bereits ein Kommando an den EDT abgesetzt wurde und eine Schnittstellenänderung nicht mehr möglich ist.

Hinweise

- In der vordefinierten Prozedur ASSIGNFILE können die verschiedenen Arbeitsbereiche angesprochen werden. Neben *EDT (aktuell eingestellter Arbeitsbereich) kann in Klammern noch die Nummer des Arbeitsbereichs (*EDT(1) ,..., *EDT(9)) angegeben werden (siehe 5.2.1).
- Wird der EDT von einem Pascal-XT Programm aus aufgerufen, dann kann mit dem EDT-Kommando @RUN kein Pascal-XT-Programm ausgeführt werden. Der Grund liegt darin, daß beide Programme dasselbe Laufzeitsystem verwenden, was bei der Beendigung des mit @RUN gestarteten Programms zu einem Fehler führt. Nicht-Pascal-XT Programme können problemlos ausgeführt werden.
- Das Programmiersystem und die Testhilfe setzen ebenfalls über die Unterprogramm-Schnittstelle Kommandos an den EDT ab. Ein mit //RUN gestartetes Programm kann dann die Schnittstellenversion nicht mehr ändern, wenn im Programmiersystem oder in der Testhilfe bereits der EDT aufgerufen wurde.

ERRORS

Das Paket ERRORS liefert bei einer Ausnahme zusätzliche Informationen zur näheren Klassifizierung des Fehlers. Die Bedeutung der Systemfehlercodes ist im Abschnitt 10.4 beschrieben.

```
(*************************************************************************
*                                                                        *
*       COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990        *
*                     ALL RIGHTS RESERVED                                *
*                                                                        *
*************************************************************************)

{$DEBUG=ON}
package ERRORS;


            (***********************************************)
            (*  The body of this package is part of the     *)
            (*  PASCAL-XT runtime system.                   *)
            (*                                              *)
            (*  Required Pascal-XT Runtime System: >= 2.1A  *)
            (*  Available in V2.1A and higher:              *)
            (*     INTR_MESSAGE                             *)
            (*     PRINT_ERROR_INFO                         *)
            (*     RERAISE                                  *)
            (***********************************************)


function system_code: integer;
   (* returns the system error code as additional information about *)
   (* the last error. 0 is returned if there is no system error     *)
   (* code.                                                         *)

function interrupt_address: integer;
   (* returns the interrupt address of the last exception. 0 is     *)
   (* returned if there is no error.                                *)

function file_info: string;
   (* file_info returns file information if an exception was raised *)
   (* by a file error. For other errors the empty string is         *)
   (* returned.                                                     *)

function intr_message: string;
   (* returns the most recent /INTR message if an exception was     *)
   (* raised by /INTR; if /INTR was sent without message or no      *)
   (* /INTR at all, the empty string is returned.                   *)

procedure print_error_info;
   (* Lists the dynamic stack chain as it existed at the instant    *)
   (* of the most recent exception. The format is the same as       *)
   (* in the case of an unhandled exception. If no exception has    *)
   (* occurred yet, nothing is listed.                              *)

inline procedure reraise;
begin raise(0) end;
   (* Resumes the raise of the most recently raised exception.      *)
   (* Raises system_error with system_code 5001, if no exception    *)
   (* has occurred yet.                                             *)

end (* package ERRORS *).
```

SYSTEM_CODE

liefert den Systemfehlercode des zuletzt aufgetretenen Fehlers (siehe 10.4). Für nicht vordefinierte Ausnahmen oder bei fehlerfreiem Ablauf ist der Funktionswert undefiniert.

INTERRUPT_ADDRESS

liefert die Unterbrechungs-Adresse der letzten Unterbrechnung oder des letzten Aufrufs der Standardprozedur RAISE. Bei fehlerfreiem Ablauf ist der Funktionswert undefiniert.

FILE_INFO

Bei einem Dateizugriffsfehler werden der Name der Pascal-Datei, die in der Standardprozedur ASSIGNFILE angegebene externe Beschreibung und der Name der aktuellen (nicht temporären) BS2000-Datei zurückgeliefert. Bei anderen Fehlern wird der Leerstring zurückgegeben.

PRINT_ERROR_INFO

Diese Prozedur gibt die dynamische Aufrufkette, wie sie zum Zeitpunkt des Auftretens der letzten Ausnahme bestanden hat, auf SYSOUT und SYSLST aus, ohne daß diese Ausnahme neuerlich ausgelöst oder propagiert wird. Das Ausgabeformat ist das gleiche wie im Fall einer unbehandelten Ausnahme. PRINT_ERROR_INFO kann jederzeit in- und außerhalb von Ausnahmebehandlern aufgerufen werden. Ist noch keine Ausnahme aufgetreten, wird nichts ausgegeben.

INTR_MESSAGE

Diese Funktion liefert den Text der bei der letzten (K2)/INTR-Unterbrechung angegebenen Meldung. Es wird der Leerstring zurückgeliefert, wenn beim letzten /INTR-Kommando kein Text angegeben bzw. während des Programmlaufs noch kein /INTR-Kommando angegeben wurde.

RERAISE

Diese (Inline-)Prozedur setzt die Propagierung der zuletzt aufgetretenen, von einem Ausnahmebehandler bereits behandelten Ausnahme fort. Ist kein weiterer Ausnahmebehandler vorhanden, wird die dynamische Aufrufkette, wie sie zum Zeitpunkt des Auftretens der Ausnahme bestanden hat, auf SYSOUT und SYSLST ausgegeben.

RERAISE kann in- und außerhalb von Ausnahmebehandlern aufgerufen werden. Ist noch keine Ausnahme aufgetreten, führt ein Aufruf zu einem Laufzeitfehler (SYSTEM_ERROR, System_Code=5001).

HEAPSUPPORT

Das Paket HEAPSUPPORT ermöglicht dem Benutzer die Benutzung mehrere Heaps (Halden). Dynamisch erzeugte Objekte werden immer im aktuell eingestellten Heap allokiert. Objekte in einem Heap können auch auf Objekte in anderen Heaps verweisen. Wird dieses Paket nicht verwendet, oder wurde kein Heap eingestellt, dann werden Objekte in dem vom Laufzeitsystem standardmäßig eingestellten Default-Heap allokiert.

Mit RELEASE(p) wird derjenige Haldenpegel zurückgesetzt, der zuvor mit dem zugehörigen MARK(p) markiert wurde.

Dieses Heapkonzept ermöglicht die Zusammenfassung logisch zusammengehöriger Objekte, oder von Objekten mit etwa gleich langer Lebensdauer, in einem Heap. Die Freigabe dieser Objekte erfolgt einfach und schnell durch Freigabe dieses Heaps.

```
(*********************************************************************
*                                                                    *
*       COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990    *
*                     ALL RIGHTS RESERVED                            *
*                                                                    *
*********************************************************************)

 {$DEBUG=ON}
 package HEAPSUPPORT;

             (*************************************)
             (*  The body of this package is part   *)
             (*  of the Pascal-XT runtime system    *)
             (*************************************)

 type

    heap  = heap_descriptor;   (* private *)

 procedure create_heap  ( var h : heap );
    (* Creates a new heap and returns the pointer of the heapdescriptor  *)
    (* for subsequent use by select_heap and release_heap.               *)
    (* Raises memory_error with system_code 1403 if creation of a new    *)
    (* heap is not possible.                                             *)
    (* Note : create_heap does not select the new heap.                  *)

 procedure select_heap  ( h : heap );
    (* Selects the heap h as the current heap. All subsequent calls to   *)
    (* the standard procedure new will allocate memory on this heap.     *)
    (* select_heap(default_heap) is allowed, select_heap(current_heap)   *)
    (* has no effect.                                                    *)
    (* select_heap on an already realeased heap raises system_error with *)
    (* system_code 1404.                                                 *)

 function  current_heap  : heap;
    (* Returns the pointer of the heap descriptor of the current heap.   *)
    (* Note : this may be the default heap.                              *)

 function  default_heap  : heap;
    (* Returns the pointer of the heap descriptor of the default heap.   *)
    (* Note : The Initialisation routine of the Runtime-System creates   *)
    (*        and selects a heap before transferring control to the      *)
    (*        user program. This heap is called the "default heap".      *)
```

```
procedure release_heap  ( h : heap );
   (* All dynamic variables allocated in the heap h will be "disposed"  *)
   (* and their memory together with the heap descriptor returned to    *)
   (* the operating system.                                             *)
   (* release_heap(current_heap) raises system_error, system_code 1405  *)
   (* release_heap(default_heap) raises system_error, system_code 1406  *)

end (* package HEAPSUPPORT *).
```

CREATE_HEAP (h)

legt einen Deskriptor für einen neuen Heap an und gibt einen Zeiger auf diesen zurück, der als Parameter für die Prozeduren select_heap und release_heap dient. Auf einen neu angelegten Heap muß explizit (durch die Prozedur select_heap) umgeschaltet werden, bevor er verwendet werden kann.

DEFAULT_HEAP

liefert einen Zeiger auf den vom Laufzeitsystem standardmäßig eingestellten Heap zurück. Die Freigabe des default_heaps ist nicht möglich und führt zu einem Laufzeitfehler (SYSTEM_ERROR, System_Code = 1408).

CURRENT_HEAP

liefert einen Verweis auf den Heap zurück, der zuletzt durch die Prozedur select_heap eingestellt wurde. Falls select_heap noch nicht verwendet wurde, gilt current_heap = default_heap. Die Freigabe des current_heap ist nicht möglich und führt zu einem Laufzeitfehler (SYSTEM_ERROR, System_Code = 1407).

SELECT_HEAP (h)

Es wird auf den durch h angegebenen Heap umgeschaltet. Dieser Heap muß zuvor mittels create_heap erzeugt worden sein, wenn es nicht der default heap ist. Nachfolgende Aufrufe der Standardprozedur NEW erzeugen Objekte auf diesem Heap.
Der Aufruf select_heap (current_heap) hat keine Wirkung, während der Aufruf select_heap (default_heap) auf den default_heap umschaltet, sofern nicht schon current_heap = default_heap gilt.

RELEASE_HEAP (h)

Es wird der durch h angegebene Heap freigegeben und der belegte Hauptspeicher an das Betriebssystem zurückgegeben. Die Freigabe des aktuellen Heaps oder des default_heaps führt zu einem Laufzeitfehler.

Beispiel

Das Paket HEAPSTACK simuliert die Funktionen pushheap, popheap und killheap von Pascal (BS2000) Version 3.

```
*** SOURCE LISTING ***    BS2000 SIEMENS PASCAL-XT COMPILER  V2.0A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.MANUAL,HEAPSTACK.SPEC(*STD,S))

     1      (*$TITLE = 'Simulation der Heapkellerung von Pascal Version 3'*)
     2      package HEAPSTACK;
     3
     4      procedure pushheap;
     5      procedure popheap;
     6      procedure killheap;
     7
     8      end (* package HEAPSTACK *).

************************************************************
*                   COMPILATION SUMMARY                    *
************************************************************
*  ERRORS DETECTED         :        0                      *
*  WARNINGS                :        0                      *
*  NOTES                   :        0                      *
*  SIZE OF CODE MODULE     :        0 BYTES                *
*  SIZE OF DATA MODULE     :        0 BYTES                *
*  COMPILATION TIME        :    0.071 SEC                  *
************************************************************

*** SOURCE LISTING ***    BS2000 SIEMENS PASCAL-XT COMPILER  V2.0A00...

GLOBAL OPTIONS FOR THIS COMPILATION

DEBUG      =    OFF          BY DEFAULT
GENERATE   =    ON           BY DEFAULT
MAP        =    OFF          BY DEFAULT
STANDARD   =    OFF          BY DEFAULT
XREF       =    OFF          BY DEFAULT

LIST OF RECOMPILED PACKAGE SPECIFICATIONS (SOURCE FILES)

               ($PASSUP-XT,HEAPSUPPORT.SPEC(*STD,S))

               ($USERID.PLAM.MANUAL,HEAPSTACK.SPEC(*STD,S))

CURRENT COMPILATION UNIT (SOURCE FILE)

               ($USERID.PLAM.MANUAL,HEAPSTACK.BODY(*STD,S))
```

```
 1      with HEAPSUPPORT;
 2      from HEAPSUPPORT use heap       , default_heap,
 3                           create_heap, select_heap,  release_heap;
 4
 5      package body HEAPSTACK;
 6
 7      type
 8         heapstack     = ↑heapctrlblock;
 9         heapctrlblock = record
10                            curr : heap;
11                            rest : heapstack;
12                            end;
13
14      var
15         hs       : heapstack;
16         lastheap : heap;
17
18      procedure pushheap;
19      var
20         hsnew   : heapstack;
21         newheap : heap;
22
23      begin
24         create_heap (newheap);
25         new (hsnew);
26         hsnew↑.curr := newheap;
27         hsnew↑.rest := hs;
28         hs          := hsnew;
29         lastheap    := nil;
30         select_heap (newheap);
31      end (* pusheap *) ;
32
33      procedure popheap;
34      begin
35         if hs↑.curr <> default heap_then begin
36            lastheap := hs↑.curr;
37            hs       := hs↑.rest;
38            select_heap (hs↑.curr);
39            end;
40      end (* popheap *) ;
41
42      procedure killheap;
43      begin
44         if lastheap <> nil then begin
45            release_heap (lastheap);
46            lastheap := nil;
47            end;
48      end (* killheap *) ;
49
50      begin
51         new (hs);
52         hs↑.curr := default_heap;
53         hs↑.rest := nil;
54         lastheap := nil;
55      end (* package HEAPSTACK *).
```

```
************************************************************
*                    COMPILATION SUMMARY                   *
************************************************************
*  ERRORS DETECTED          :        0                     *
*  WARNINGS                 :        0                     *
*  NOTES                    :        0                     *
*  SIZE OF CODE MODULE      :      896 BYTES               *
*  SIZE OF DATA MODULE      :      112 BYTES               *
*  COMPILATION TIME         :    0.245 SEC                 *
************************************************************
```

MEMORYMANAGER

Die in dem Paket MEMORYMANAGER zur Verfügung gestellten Prozeduren realisieren die Funktionen der BS2000 Makro-Aufrufe GTMAP, REQM und RELM (siehe [8]). Im Normalfall sollte ein Pascal-Programm zur Speicherverwaltung nur die vordefinierten Prozeduren NEW, DISPOSE, MARK und RELEASE verwenden. Zur Information über die momentane Speicherbelegung ist ein Aufruf von MEMORYMANAGER. GETMAP sinnvoll. Aufrufe von MEMORYMANAGER.REQMEM und MEMORY MANAGER.RELMEM sind nur Spezialanwendungen vorbehalten. Insbesondere ist es **nicht erlaubt**, RELMEM für Speicherseiten aufzurufen, die nicht zuvor durch expliziten Aufruf von MEMORYMANAGER.REQMEM beschafft wurden. Eine fehlerhafte Anwendung führt im allgemeinen zu einem Programmabsturz.

Es ist zu beachten, daß das Laufzeitsystem selbst eine Speicherverwaltung durchführt und nicht in jedem Fall frei gewordene Seiten an das Betriebssystem zurückgibt. Dies wird insbesondere dann gemacht, wenn damit zu rechnen ist, daß die frei gewordenen Seiten bald wieder gebraucht werden.

```
(*********************************************************************
*                                                                    *
*       COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990    *
*                     ALL RIGHTS RESERVED                            *
*                                                                    *
*********************************************************************)

 {$DEBUG=ON}
 package MEMORYMANAGER;

             (*************************************)
             (*  The body of this package is part   *)
             (*  of the Pascal-XT runtime system    *)
             (*************************************)

 const

      max_page             = 2047;
      max_pages_per_reqmem = 1792; (* BS2000 V7.5 and higher *)
      any_page             = 0;    (* code for non-specific page request *)

 type

      page_range           = 0 .. max_page;
      reqmem_page_range    = 1 .. max_pages_per_reqmem;

      page_map             = set of 0..max_page;

 procedure getmap  ( var map  : page_map  );
    (* returns a map of the used pages of the user's memory *)

 procedure reqmem  (     number_of_pages : reqmem_page_range;
                         page_number     : page_range; (* or "any_page" *)
                   var first_page        : pointer );
    (* request a contiguous memory area of pages *)

 procedure relmem  ( number_of_pages : page_range;
                     first_page      : pointer);
    (* releases the specified number of pages starting at "first_page" *)

 end (* package MEMORYMANAGER *).
```

GETMAP (map)

informiert über die Belegung der Seiten der ersten 16 MB des Benutzerspeichers. Eine Seite i ist genau dann frei, wenn "i in map" gilt (vgl. auch [7]). Diese Prozedur liefert keine Informationen über die Belegung der Seiten im oberen Adressraum (> 16 MB).

REQMEM (number_of_pages, page_number, first_page)

fordert einen zusammenhängenden Speicherbereich von number_of_pages Seiten an, beginnend mit der Seite page_number. Ist page_number = any_page, so wird mit einer beliebigen, freien Seite begonnen. Kann die Anforderung vom BS2000 erfüllt werden, dann wird ein Zeiger auf den zugewiesenen Speicherbereich im Ergebnisparameter first_page übergeben, ansonsten wird der Wert NIL zurückgegeben.

RELMEM (number_of_pages, first_page)

gibt die in number_of_pages angegebene Anzahl von Speicher-Seiten frei, beginnend mit der durch first_page angegebene Seite.

Kompatibilitätsprobleme zwischen Pascal-XT V2.2 und V3.0

In der nächsten Pascal-XT-Version löst eine neue Sprache den bisherigen Sprachumfang ab. Diese umfaßt den in der ISO-Norm 10206 definierten Sprachumfang von Extended Pascal mit einigen Erweiterungen. Um die Migrationsprobleme möglichst gering zu halten, wird ein Tool angeboten, das Pascal-XT-Quellprogramme der Version V2.2 transformiert in Quellprogramme der Version V3.0. Einige Sprachkonstrukte bereiten hier aber Probleme. Deshalb gibt bereits der Pascal-XT-Compiler V2.2A an diesen Stellen Hinweise (Notes) aus, die den Anwender auf diese Probleme aufmerksam machen. Bei Sprachkonstrukten, die das Migrationstool problemlos in äquivalente Konstrukte von Pascal-XT V3.0 transformieren kann, gibt der Compiler keine Hinweise aus.

Die Hinweise des Compilers betreffen folgende Sprachkonstrukte:

Standardprozedur Elaborate(p)
(siehe Sprachbeschreibung [1], 15.12)

Diese Prozedur, mit der der Anwender die Reihenfolge beeinflussen kann, in der Pakete initialisiert werden, wird in Pascal-XT V3.0 nicht mehr unterstützt. Bei zyklischen Paketreferenzen dürfen deshalb die Paket-Blöcke keine Anweisungen mehr enthalten.

Domänentyp von Zeigern
(siehe Sprachbeschreibung [1], 6.4)

Der Domänentyp eines Zeigertyps und der Zeigertyp müssen in Pascal-XT V3.0 im selben Deklarationsteil definiert werden.

Auf dieses Problem kann der Compiler nicht in jedem Fall exakt hinweisen.

Prozeduren und Funktionen mit der Direktive Forward
(siehe Sprachbeschreibung [1], 8.6)

Wird in Pascal-XT V3.0 eine Prozedur bzw. Funktion mit der Direktive "FORWARD" deklariert, müssen Kopf und Block der Prozedur bzw. Funktion im selben Deklarationsteil deklariert werden. Das heißt, es dürfen dazwischen keine Deklarationen von Variablen, Typen oder Konstanten stehen.

Qualifizierte Mengenbildner
(siehe Sprachbeschreibung [1], 9.4)

In Pascal-XT müssen die Ordinalwerte des Basistyps eines nicht qualifizierten Mengenbildners im Bereich 0..2047 liegen (siehe 4.2., Nr.11). Über qualifizierte Mengenbildner ist es möglich, Mengen außerhalb dieses Bereichs zu konstruieren. In Pascal-XT V3.0 wird der Basistyp eines Mengenbildners auf die Ordinalwerte im Bereich 0..2047 festgelegt. Es ist nicht mehr möglich, Mengen außerhalb dieses Bereichs zu konstruieren.

Indizierung und Selektion von Aggregat-Elementen
(siehe Sprachbeschreibung [1], 9.5)

In Pascal-XT V3.0 ist es nicht mehr möglich, in einem Ausdruck ein Aggregat anzugeben, in dem durch Indizierung bzw. Selektion ein Element ausgewählt wird. Bei Bedarf kann das Aggregat als Konstante oder bei nicht statischen Elementen als Variable definiert werden.

Kleinste darstellbare Realzahl
(Siehe 4.2, Nr.1 und 2)

Die Konstante MINREAL steht für die kleinste auf einer gegebenen Maschine darstellbare Realzahl. In Pascal-XT V3.0 entspricht die Konstante der kleinsten normalisierten Realzahl und hat damit einen etwas größeren Wert.

Literatur

[1] **Pascal-XT (BS2000)**
Sprachbeschreibung

Zielgruppe
Pascal-XT-Anwender im BS2000
Inhalt
- Struktur und Elemente eines normgemäßen Pascal-Programms
- Erweiterungen von Pascal-XT gegenüber der Norm

[2] BS2000
Einführung in die Dialogschnittstelle SDF
Benutzerhandbuch

Zielgruppe
BS2000-Anwender
Inhalt
- Die verschiedenen Eingabemöglichkeiten im Systembetrieb mit SDF
- Bedienungshinweise und Beispiele mit der optionalen Benutzerführung über Menüs

Einsatz
Allgemein

[3] **LMS** (BS2000)
ISP-Format
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Beschreibung der Anweisungen zum Erstellen und Verwalten von PLAM-Bibliotheken und darin enthaltenen Elementen.
Häufige Anwendungsfälle werden an Hand von Beispielen erklärt.

[4] BS2000
Dienstprogramme
Benutzerhandbuch

Zielgruppe
BS2000-Anwender (nicht privilegiert)
Inhalt
Dienstprogramme für den nicht privilegierten Benutzer des BS2000
Einsatz
BS2000-Teilnehmerbetrieb

[5a] BS2000
Systemanwendung Teil 1
Taschenbuch

Zielgruppe
Erfahrene BS2000-Anwender
Inhalt
Eine Zusammenstellung der Kommandos und Makros im BS2000, der wichtigsten Tabellen und Register, der Code-Tabellen und Systemkonventionen
Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb

[5b] BS2000
Systemanwendung Teil 2
Taschenbuch

Zielgruppe
Erfahrene BS2000-Anwender
Inhalt
Eine Zusammenstellung der
- Assemblerbefehle und Assembleranweisungen
- Anweisungen der Softwareprodukte und Dienstprogramme EDT, SORT, ARCHIVE, PERCON, LEASY, TSOSLNK, DCAT, PASSWORD, FDEXIM, FDRIVE, DPAGE, SODUMP, $PCCNTRL, TPCOMP2, PRSERVE

Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb

[6] BS2000
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

Zielgruppe
- BS2000-Assembler-Programmierer (nicht privilegiert)
- Systemverwalter

Inhalt
- Alle Makroaufrufe an den Ablaufteil in lexikalischer Reihenfolge mit Hinweisen und Beispielen, einschließlich ausgewählter Makroaufrufe für das DVS und für TIAM
- Zusammenstellung der Makroaufrufe nach Anwendungsgebieten
- Ausführlicher Lernteil über Ereignissteuerung, Serialisation, Inter-Task-Kommunikation, Contingencies

Einsatz
BS2000-Anwendungsprogramme

[7] BS2000
Benutzerkommandos (ISP-Format)
Benutzerhandbuch

Zielgruppe
BS2000-Anwender (nicht privilegiert)
Inhalt
Alle BS2000-Systemkommandos in lexikalischer Reihenfolge mit Hinweisen und Beispielen.
Folgende Liefereinheiten sind berücksichtigt:
BS2000-GA, MSCF, JV, FT, TIAM
Einsatz
BS2000-Dialogbetrieb, -Prozeduren, -Stapelbetrieb

[7a] BS2000
Benutzer-Kommandos (SDF-Format)
Benutzerhandbuch

Zielgruppe
BS2000-Anwender
Inhalt
Benutzer-Kommandos des BS2000 in der Syntax der Dialogschnittstelle SDF (System Dialog Facility)
Einsatz
- BS2000-Dialogbetrieb
- BS2000-Stapelbetrieb mit SDF

[8] BS2000
Jobvariablen
Benutzerhandbuch

Zielgruppe
BS2000-Benutzer
Inhalt
- Anwendungsmöglichkeiten für Jobvariablen zur Steuerung und Überwachung von Aufträgen und Programmläufen
- Bedingungsabhängige Auftragssteuerung

– Alle erforderlichen Kommandos und Makroaufrufe
– Anwendungsbeispiele

Einsatz

BS2000-Teilnehmerbetrieb

[9] **UTM** (TRANSDATA, BS2000)
Planen und entwerfen
Benutzerhandbuch

Zielgruppe

– Organisatoren
– Einsatzplaner
– Programmierer

Inhalt

– Einführung in UTM, Erläuterung des Programm-Speicher- und Schnittstellenkonzeptes sowie des Zugriffs auf Daten und Dateien und der Zusammenarbeit mit Datenbanken
– Hinweise zu Design, Optimierung und Performance von UTM-Anwendungen sowie Datenschutz und Anwendungsverbund.

Einsatz

BS2000-Transaktionsbetrieb

[10] **UTM** (TRANSDATA, BS2000)
Anwendungen generieren und administrieren
Benutzerhandbuch

Zielgruppe

– Systemverwalter
– Administratoren

Inhalt

– Aufbau, Generierung und Betrieb von UTM-Anwendungen
– Arbeiten mit UTM-Meldungen und Fehlercodes

Einsatz

BS2000-Transaktionsbetrieb

[11a] **UTM** (TRANSDATA)
Ergänzung für Pascal-XT
Benutzerhandbuch

Zielgruppe

Programmierer von UTM-Pascal-XT-Anwendungen

Inhalt

– Umsetzung der Programmschnittstelle KDCS in die Sprache Pascal-XT
– Alle Informationen, die der Programmierer von UTM-Pascal-XT-Anwendungen benötigt

Einsatz
BS2000-Transaktionsbetrieb

[11b] **UTM** (TRANSDATA)
Anwendungen programmieren
Benutzerhandbuch

Zielgruppe
Programmierer von UTM-Anwendungen
Inhalt
- Sprachunabhängige Beschreibung der Programmschnittstelle KDCS,
- Aufbau von UTM-Programmen
- KDCS-Aufrufe
- Testen von UTM-Anwendungen
- Alle Informationen, die der Programmierer von UTM-Anwendungen benötigt

Einsatz
BS2000-Transaktionsbetrieb

[12] **Assemblerbefehle** (BS2000)
Beschreibung

Zielgruppe
BS2000-Assembler-Programmierer
Inhalt
Beschrieben sind alle Assemblerbefehle (nicht privilegiert) der vom BS2000 unterstützten Zentraleinheiten in alphabetischer Reihenfolge
Bei jedem Befehl sind dargestellt:
- seine Funktion
- sein Assemblerformat, d.h. seine Schreibweise in Assemblersprache
- sein Maschinenformat, d.h. seine Darstellung in der Zentraleinheit
- sein Ablauf im Detail
- etwaige von ihm gesetzte Werte der Anzeige
- die bei seinem Ablauf möglichen Programmunterbrechungen
- Programmierhinweise
- ein oder mehrere Beispiele

Einsatz
BS2000-Assembler-Anwendungsprogrammierer

[13a] **EDT** (BS2000)
Anweisungen
Benutzerhandbuch

Zielgruppe
- EDT-Einsteiger
- EDT-Anwender

Inhalt
- Bearbeiten von SAM- und ISAM-Dateien und Elementen aus Programm-Bibliotheken
- Einstieg in die Arbeitsweise des EDT und Beschreibung der Arbeitsmodi
- Erstellen von EDT-Prozeduren
- Beschreibung aller EDT-Anweisungen. Häufige Anwendungsfälle werden anhand zahlreicher Beispiele erklärt;

Einsatz
Aufbereiten von Dateien

[13b] **EDT** (BS2000)
Unterprogrammschnittstellen
Benutzerhandbuch

Zielgruppe
Erfahrene EDT-Anwender und Programmierer
Inhalt
- Bearbeiten von SAM- und ISAM-Dateien und Elementen aus Programm-Bibliotheken
- Einbinden der EDT-Funktionalität in eigene Programme
- Beschreibung der Unterprogrammschnittstelle des EDT
- Aufruf des EDT als Unterprogramm
- Funktionen der UP-Schnittstelle
- Aufbau und Generierung der Kontrollblöcke
- Erstellen von externen Anweisungsroutinen
- Aufruf eines Benutzerprogramms aus dem EDT

Einsatz
Programmentwicklung

[14] BS2000
DVS Einführung und Kommandoschnittstelle
Benutzerhandbuch

Zielgruppe
Nicht privilegierte BS2000-Anwender
Inhalt
- Funktionen des Datenverwaltungssystems im BS2000
- Verarbeitung von Platten- und Banddateien
- Zugriffsmethoden UPAM, SAM, BTAM, EAM, ISAM
- DVS-Kommandos

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

Zielgruppe

Assembler-Anwender im BS2000
Inhalt
- Aufruf und Steuerung des ASSEMBH
- Übersetzen, Binden, Laden und Starten
- Eingabequellen und Ausgaben des ASSEMBH
- Laufzeitsystem, Listenausgabe der strukturierten Programmierung
- Sprachverknüpfungen
- Assembler-Diagnoseprogramm ASSDIAG
- Dialogtesthilfe AID
- Meldungen des ASSEMBH
- Format der Assemblerbefehle

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an eine Geschäftsstelle unseres Hauses.

Stichwörter

C