```
**********************************
**********************************
***                            ***
*** D E M A N D - B E T R I E B ***
***                            ***
**********************************
**********************************
```

Inhaltsverzeichnis

1. Einführung

Dieses Kapitel führt in den Demand-Betrieb der SPERRY 1100/60 ein. Im **Demand-Betrieb** (syn. Dialog-Betrieb) arbeitet der Benutzer an einem Terminal interaktiv mit der SPERRY 1100/60: Jeder Eingabe (von der Tastatur) folgt eine Antwort der SPERRY 1100/60 (auf dem Bildschirm). Im Batch-Betrieb dagegen werden alle Eingaben, die zur Bearbeitung einer Aufgabe nötig sind, als Ganzes an die SPERRY 1100/60 übergeben (z.B. in einem File), die Antworten werden gesammelt und nach Bearbeitung der gesamten Aufgabe an einem Drucker oder Terminal ausgegeben (vgl. BATCH-BETRIEB).

Eine Aufgabe (IBM, DEC: Job) wird auf der SPERRY 1100/60 als **Run** bezeichnet, die Eingaben des Runs heißen **Runstream**. Ein Run, der interaktiv bearbeitet wird, heißt **Demand Run** (IBM: Session, DEC: Interaktiver Job); ein Run, der als Ganzes bearbeitet wird, heißt **Batch Run** (IBM, DEC: Batch Job). Im Demand-Betrieb werden **Demand Sessions** (IBM, DEC: -) geführt; innerhalb einer Demand Session können sowohl Demand Runs geführt werden, als auch Runstreams für Batch Runs ein- oder Output Files von Batch oder Demand Runs ausgegeben werden.

Innerhalb eines Runs werden Leistungen der SPERRY 1100/60 mit Hilfe einer Kommandosprache angefordert. Zur Zeit stehen zwei Kommandosprachen zur Verfügung: **ECL** (Executive Control Language) und **CTS** (Conversational Time Sharing). ECL ist eine veraltete für Batch Runs konzipierte Kommandosprache (z.B. ist Programmieren auf Kommandosprachenebene nur ansatzweise möglich), die das interaktive Arbeiten nur unzureichend unterstützt (z.B. gibt es kein "Help"). **ECL Control Statements** werden direkt von der **EXEC** (Executive System), dem Kern des Betriebssystems OS 1100, interpretiert. Der Zustand eines Runs, in dem ECL Control Statements eingegeben werden können, wird daher **EXEC Mode** (syn. Control Mode) genannt. Das Dialogsystem CTS ist auf ECL aufgesetzt; viele **CTS Commands** werden durch ECL Control Statements realisiert. CTS unterstützt das interaktive Arbeiten besser als ECL, jedoch können einige Leistungen der SPERRY 1100/60 (z.B. Elemente von Libraries umbenennen) nicht mit CTS Commands angefordert werden. Darüber hinaus kann die Programmentwicklung innerhalb CTS (insbesondere das Binden) sehr unübersichtlich werden, so daß immer wieder auf ECL Control Statements zurückgegriffen werden muß. Dazu kann entweder ein ECL Control Statement von CTS an die EXEC durchgereicht werden (PXQT), oder es kann zwischen EXEC Mode und CTS hin- und hergeschaltet werden (@CTS,XCTS). Dem Anfänger wird empfohlen, in Demand Runs zunächst mit CTS zu arbeiten.

Dokumentationen zu ECL und CTS:

UP-8118	Time Sharing Guide
UP-4144	Executive System Programmer Reference
UP-7940	Conversational Time Sharing Programmer Reference
UP-7824	Executive System Software Summary
UP-7946	Conversational Time Sharing Summary

Der Time Sharing Guide ist als Einführungsliteratur konzipiert; er beschreibt das Arbeiten mit CTS (Editor, Files, Programmentwicklung) in gut verständlicher Form. Die Programmer References eignen sich zum Nachschlagen von Leistung und Format eines ECL Control Statements bzw. CTS Commands. Die beiden Summaries werden lediglich als Gedächtnisstütze bei der Arbeit am Terminal empfohlen.

Gebrauchsanweisung: Es wird empfohlen, zunächst die Abschnitte 2 bis 5 zu lesen und dann das erste Beispiel in Abschnitt 7 am Terminal nachzuvollziehen. Danach sollte versucht werden, die in den Abschnitten 2 bis 5 erläuterten Funktionen spielerisch anzuwenden. Wer größere Programmsysteme an der SPERRY 1100/60 entwickeln will, sollte auch noch den Abschnitt 6 und das zweite Beispiel in Abschnitt 7 lesen (Programmentwicklung mit ECL); dies fördert zudem das Verständnis für das Arbeiten mit CTS. Auch der Time Sharing Guide ist hilfreich beim Einarbeiten in CTS.

Das Drücken einer Taste wird dargestellt durch <Taste> (z.B. <XMIT> für Drücken der XMIT Taste). Beispiele für Eingaben sind ***kursiv und fett*** gedruckt und haben die Form >... <XMIT>; Antworten der SPERRY 1100/60 sind i.a. groß geschrieben. In den Erläuterungen zu den Beispielen sind Bezeichnungen wie z.B. Filenames oder Command Names groß geschrieben. Zur Darstellung von Formaten sind wählbare Bezeichnungen klein, obligate groß geschrieben; Angaben in ekkigen Klammern sind optional.

Alphanumerische Zeichen sind Buchstaben (ohne Unterscheidung groß/klein), Ziffern und die Sonderzeichen $ und -; das Sonderzeichen $ darf nicht in Bezeichnungen verwendet werden, die vom Benutzer gewählt werden.

Übersicht: Abschnitt 2 beschreibt Eigenschaften von verschiedenen Terminals. Abschnitt 3 beschreibt die Funktionen für das Arbeiten mit der SPERRY 1100/60 im Demand-Betrieb; mit diesen Funktionen hat das Arbeiten im Demand-Betrieb folgende grundsätzliche Struktur:

```
Verbindung aufbauen ($$SON, vgl. 3.1.1)
   Demand Session beginnen ($$OPEN DEMAND, vgl. 3.1.1)
      .
      .
      Demand Run beginnen  (@RUN, vgl. 3.1.2)
         .
         .
         ECL Control Statement ausführen (vgl. 3.1.3)
         .
         .
         Programm ausführen (@XQT, vgl. 6.5)
         .
         .
         CTS beginnen (@CTS, vgl. 3.1.2)
            .
            .
            CTS Command ausführen (vgl. 3.1.3)
            .
            .
            Programm ausführen (XQT, vgl. 5.3)
            .
            .
            CTS Subroutine ausführen
            (CALL, vgl. KOMMANDOSPRACHEN)
            .
            .
         CTS beenden (XCTS, vgl. 3.1.3)
         .
         .
      Demand Run beenden (@FIN, vgl. 3.1.4)
      .
      .
      Batch Run eingeben (@RUN/B, vgl. BATCH-BETRIEB)
      .
      .
      Output File ausgeben (@@SEND, vgl. 3.5.3)
      .
      .
   Demand Session beenden ($$CLOSE, vgl. 3.1.5)
Verbindung abbauen ($$SOFF, vgl. 3.1.5)
```

Abschnitt 4 beschreibt das Arbeiten mit Files und Libraries. Abschnitt 5 behandelt die Programmentwicklung (Edieren, Compilieren, Binden, Starten) in FORTRAN innerhalb von CTS; Abschnitt 6 beschreibt, wie die gleichen Funktionen mit ECL Control Statements zu realisieren sind. Abschnitt 7 enthält Beispiele für Demand Runs zum Edieren, Compilieren, Binden und Starten eines FORTRAN Programms mit Hilfe von CTS bzw. ECL.

2. Terminals

Demand Runs können an Terminals vom Typ

- SPERRY UTS20 — Standard Terminal
- EUROBEE FT20 } DEVELNET Terminals
- QUME QVT201 }
- DEC VT220 }
- ... }

geführt werden. Jedes Terminal trägt einen Aufkleber mit Typ (z.B. UTS20) und Anschlußbezeichnung (z.B. M14110). Das Terminal UTS20 ist als Standard Terminal anzusehen; seine Eigenschaften werden im Abschnitt 2.2 erläutert; zu den abweichenden Eigenschaften der anderen Terminals vgl. 2.3. Zu allen Terminals gibt es **Bedienungsanleitungen** des HRZ (sie liegen neben den Terminals). An die Terminals können Matrixdrucker angeschlossen sein.

Microcomputer/PC's mit VT100 Emulation können ebenfalls über das DEVELNET auf die SPERRY 1100/60 zugreifen (vgl. 2.3). Für folgende PC's hat das HRZ die entsprechenden Bedienungsanleitungen erstellt:

- IBM PC AT
- Olivetti M24

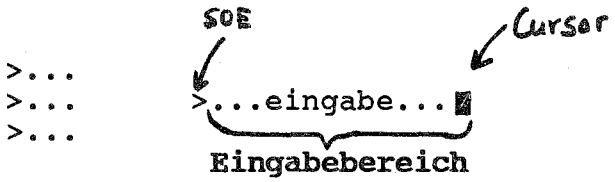
2.1 Datenübertragung

Alle Terminals sind entweder direkt an den Datenübertragungsrechner DCP/40 angeschlossen oder können über das DEVELNET mit ihm verbunden werden. Alle Terminal Ein/Ausgaben werden über den **DCP/40** zu/von der SPERRY 1100/60 übertragen.

Jedes eingegebene Zeichen wird am Bildschirm auf der Cursor Position angezeigt. Der Bildschirm hat in der Regel keine schreibgeschützten Bereiche, es kann an beliebiger Stelle geschrieben werden. Übertragen werden kann ein beliebiger Bereich des Bildschirms; in der Regel dürfen Eingaben jedoch höchstens 80 Zeichen lang sein. Der Anfang des Eingabebereichs ist durch ein **SOE** (▸, Start of Entry, wird in der Regel als Prompt von der SPERRY 1100/60 gesetzt), das Ende mit dem Cursor zu markieren; wenn mehrere SOEs auf dem Bildschirm stehen, beginnt der Eingabebereich hinter dem SOE, das als nächstes vor dem Cursor steht (s.u.); das Zeichen auf Cursor Position wird immer mit übertragen. Zur Darstellung des SOEs wird statt ▸ im folgenden das Zeichen > verwendet. Die Übertragung erfolgt durch

<XMIT>.

Eine Eingabe kann - solange sie noch auf dem Bildschirm steht - ggf. korrigiert und erneut übertragen werden.



2.2 SPERRY Terminal UTS20

Das SPERRY Terminal **UTS20** wurde vom HRZ als **Standard Terminal** der SPERRY 1100/60 festgelegt. Neben dem hier beschriebenen Standard Terminal gibt es im HRZ noch einige Varianten (UTS20R, UTS20RW), deren Eigenschaften in der Bedienungsanleitung aufgeführt sind. Alle UTS20 sind direkt über den DCP/40 an die SPERRY 1100/60 angeschlossen.

Jedes UTS20 hat entweder eine oder zwei **Site-id's** (Anschlußbezeichnung des Terminals); sie stehen an dem Gerät (Aufkleber). Ein UTS20 mit zwei Site-id's hat zwei **logische Bildschirme** (umschalten mit <DISP 1-2>); an jedem logischen Bildschirm kann eine Demand Session (vgl. 3.1) geführt werden.

Die Bildschirmzeile unter der durchgezogenen Linie heißt **Kontrollzeile**; sie zeigt neben der Nummer des aktiven logischen Bildschirms (1 oder 2) und der Cursor Position ggf. einen besonderen Zustand des Terminals an:

WAIT Vom bzw. zum UTS20 werden Daten übertragen; Tastatur ist gesperrt, freigeben mit <KBOARD UNLOCK> möglich
MSGW (message wait) Message von einem anderen Terminal liegt vor; Ausgabe mit $$SEND
AUXB (auxiliary busy) Daten werden auf den am Terminal angeschlossenen Matrixdrucker übertragen
AXER (auxiliary error) bei der Datenübertragung zum angeschlossenen Matrixdrucker trat ein Fehler auf
POLL DCP/40 fordert Terminal zur Datenübertragung auf

Standard **Tastatur** am UTS20 ist das UTS400-Format Keyboard, vgl. Abb. 2-1. Die Tastatur besteht aus Zeichentasten (dunkelgrau) und Funktionstasten (schwarz, blau, rot). Jede Taste realisiert 1 bis 2 Zeichen bzw. Funktionen. Umgeschaltet wird innerhalb der Zeichentasten durch gleichzeitiges Drükken von SHIFT, innerhalb der Funktionstasten durch gleichzeitiges Drücken von UPPER FUNCTION. Es gibt im HRZ auch andere Tastaturen, die Zuordnung zu den in Tabelle 2-1 beschriebenen Funktionen finden Sie in der Bedienungsanleitung des Terminals.

Taste	Funktion
<SHIFT>	Umschalten (Zeichen)
<UPPER FUNCTION>	Umschalten (Funktionen)
<SOE>	>, Start of Entry, Markierung des Anfangs des Eingabebereichs
<XMIT>	Übertragen des Eingabebereichs
<KBOARD UNLOCK>	Tastatur freigeben, falls gesperrt (WAIT in Kontrollzeile)
<MSG WAIT>	Ausgabe unterbrechen
<DISP 1-2>	Umschalten auf den anderen logischen Bildschirm
<←><↓><↑><→>	Cursor positionieren
<CURSOR TO HOME>	Cursor in linke obere Ecke des Bildschirms positionieren
<RETURN>	Cursor an den Anfang der nächsten Zeile positionieren
<LINE DUP>	Zeile, in der der Cursor steht, in die nächste Zeile kopieren
<INSERT IN DIS>	Blank einfügen auf Cursor Position (restliche Zeichen des Bildschirms verschieben)
<INSERT LINE>	Leerzeile einfügen auf Cursor Zeilenposition
<INSERT IN LINE>	Blank in Zeile einfügen auf Cursor Position (restliche Zeichen der Zeile verschieben)
<ERASE DIS>	Bildschirm ab Cursor Position löschen
<DELETE IN DIS>	Zeichen auf Cursor Position löschen (restliche Zeichen des Bildschirms verschieben)
<DELETE LINE>	Zeile löschen, in der der Cursor steht
<DELETE IN LINE>	Zeichen auf Cursor Position löschen (restliche Zeichen der Zeile verschieben)
<F1> bis <F22>	spezielle vom Anwendungsprogramm zu interpretierende Funktionen

Tabelle 2-1: wichtige Funktionen des UTS20

Zeichenvorrat: Vom und zum Terminal werden 7-bit Zeichen gesendet. Diese Zeichen werden von der SPERRY 1100/60 rechtsbündig in 9-bit Zeichen (Viertelwörter) gespeichert.

Eingabe: Alle ASCII Schriftzeichen (oktale Werte 40-126, vgl. CODES im Teil ALLGEMEINES) sind über die Tastatur eingebbar, die übrigen (9-bit) Zeichen sind nur mit Hilfe des ED Prozessors eingebbar (mit dem ED Command EXCH können Ersatzzeichen festgelegt werden; z.B. ersetzt der ED Prozessor nach EXCH & 0377 jedes eingegebene & durch das 9-bit Zeichen (Viertelwort) mit oktalem Wert 377). Im Eingabebereich haben folgende Zeichen besondere Bedeutungen (vgl. 2.1 und 3.1.3):

> SOE, markiert den Anfang des Eingabebereichs

$$ in Pos. 1-2 : Network Control Statement
@@ in Pos. 1-2 : Demand Symbiont oder Transparent Control Statement
@ in Pos. 1 : ECL Control Statement

Ausgabe: Auf dem Bildschirm darstellbar sind neben den ASCII Schriftzeichen die Zeichen

▛ (blinkend, oktaler Wert 34)
▜ (blinkend, oktaler Wert 35)
> (SOE , oktaler Wert 36)
▚ (DEL , oktaler Wert 177)

Alle weiteren (9-bit) Zeichen (insbesondere mit oktalem Wert ≥ 200) sollten nicht auf das Terminal ausgegeben werden. (Sie werden ggf. als Steuerzeichen für das Terminal oder für die Datenübertragung interpretiert.)

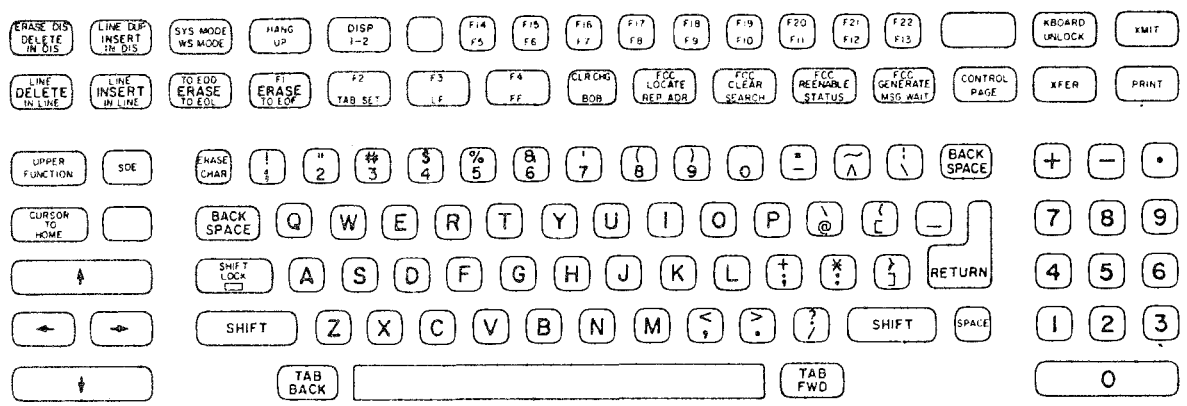


Abb. 2-1: Standard Tastatur

2.3 Terminals am DEVELNET

Ein Terminal FT20, QVT201, VT220 am DEVELNET kann wie das Standard Terminal UTS20 an der SPERRY 1100/60 benutzt werden. Das DEVELNET verbindet das Terminal über einen Protokollkonverter mit der SPERRY 1100/60 (genauer: mit dem DCP/40). Terminals am DEVELNET müssen sich gegenüber dem Protokollkonverter wie ein DEC Terminal VT100 verhalten. Der Protokollkonverter emuliert UTS20 Funktionen mit Hilfe von VT100 Funktionen, vgl. UTS20 Emulation in der Bedienungsanleitung des Terminals.

3. Arbeiten im Demand-Betrieb

Die Benutzung der SPERRY 1100/60 muß beantragt werden, vgl. BENUTZUNG im Teil ALLGEMEINES. Das HRZ teilt dem Benutzer eine **User-id** mit **Paßwort** KEINES zu; dieses Paßwort sollte schnellstmöglich geändert werden, vgl. 3.4 und BENUTZER-BERECHTIGUNGEN.

Zum Kennenlernen können potentielle Benutzer die folgende User-id/Paßwort Kombination verwenden (das Paßwort kann nicht geändert werden):

FAUST/$EINES (Account-id: MAGIE)

3.1 Demand-Betrieb

Die grundsätzliche Struktur für das Arbeiten im Demand-Betrieb finden Sie in Abschnitt 1.

3.1.1 Verbindung aufbauen, Demand Session beginnen

UTS20: Nach dem Einschalten müssen zunächst die Ausgaben eines Tests abgewartet werden (TERMINAL READY); in der Kontrollzeile muß POLL stehen. Durch **<XMIT>** wird ggf. eine (logische) Verbindung zum DCP/40 aufgebaut (Sign-on mit der Site-id des Terminals: $$SON site-id) und implizit eine Demand Session begonnen ($$OPEN DEMAND).

DEVELNET Terminals: Der Zugang zum DEVELNET wird in der Bedienungsanleitung des Terminals beschrieben. Im Auswahlmenü des DEVELNETs ist als Ziel das Kürzel für die SPERRY 1100/60 mit Standard Terminal UTS20 anzugeben (die TTY Schnittstelle ist nur für Filetransfer sinnvoll und wird hier nicht beschrieben). Das DEVELNET sucht einen freien Port des Protokollkonverters (Connected); durch **<XMIT>** wird eine Verbindung zum DCP/40 aufgebaut (Sign-on mit der Site-id des Ports) und implizit eine Demand Session begonnen ($$OPEN DEMAND).

Eine **Demand Session** kann sich in einem der folgenden 5 Zustände (Modes) befinden (Abb. 3-1 beschreibt die möglichen Zustandsänderungen):

- Run Initiation Mode: Es kann ein (Demand oder Batch) Run begonnen werden; Anfangszustand jeder Demand Session; gekennzeichnet durch die Aufforderung (vgl. 3.1.2): ENTER USERID/PASSWORD:
- Demand Mode: Es wird ein Demand Run geführt
- Remote Batch Input Mode: Es wird der Runstream eines Batch Runs eingegeben, vgl. BATCH-BETRIEB.
- Remote Batch Output Mode: Es werden Output Files ausgegeben, vgl. 3.5.3.
- Inactive Mode: z.B. nach Beendigung eines Demand Runs (Ausgabe: *TERMINAL INACTIVE*)

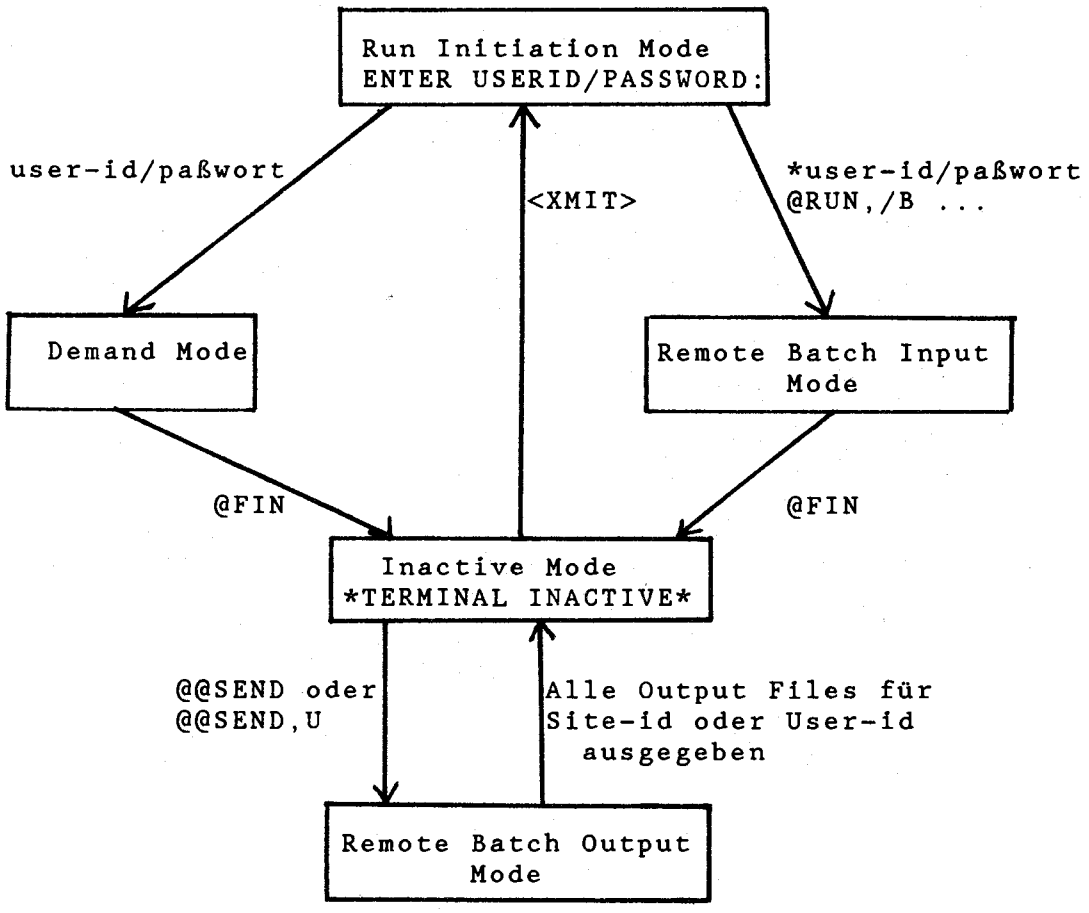


Abb. 3-1: Zustände (Modes) einer Demand Session

3.1.2 Demand Run beginnen

Ein **Demand Run** kann nur im Run Initiation Mode einer Demand Session (vgl. Abb. 3-1) begonnen werden. Nach der Aufforderung

>ENTER USERID/PASSWORD:

sind **User-id** (z.B. FAUST) und **Paßwort** (z.B. $EINES) durch / getrennt einzugeben:

>*faust/$eines* <XMIT>

User-id und Paßwort werden vom Terminal Security System TSS geprüft. Falls eine falsche User-id oder ein falsches Paßwort eingegeben wurde, gibt TSS die Meldung ID NOT ACCEPTED aus. Andernfalls wird ein Demand Run begonnen, und zwar mit Hilfe eines vom TSS generierten **@RUN Control Statements**, das **Run-id** (Name des Runs; identisch mit den ersten 6 Zeichen der User-id, im Beispiel: FAUST), **Account-id** (Name des Massenspeicherbereichs zur User-id, im Beispiel: MAGIE) und **Project-id** (Default Qualifier in File References; i.a. identisch mit Account-id, im Beispiel: MAGIE) des Runs festgelegt; das @RUN Control Statement kann auch vom Benutzer eingegeben werden, vgl. BENUTZER-BERECHTIGUNGEN. Zusätzlich werden vom TSS die Nachrichten des HRZ ausgegeben und die Ausführung eines ersten ECL Control Statements angestoßen (Input Generation, vgl. BENUTZER-BERECHTIGUNGEN) und zwar standardmäßig

@CTS,I

zur **Initialisierung von CTS**. Dabei wird

- eine leere Workarea (für das Edieren von Texten, Quellprogrammen) bereitgestellt
- der Save File (permanenter File zum Sichern von Workarea-Inhalten und CTS Subroutines) bereitgestellt. Sein Filename ist identisch mit der Run-id (im Beispiel: FAUST; der File wird ggf. kreiert und assigniert).
- die CTS Subroutine **USER$** des Save Files (falls vorhanden) ausgeführt; in das symbolische Element USER$ (IBM: PROFILE EXEC, DEC: LOGIN.COM) sollten Sie z.B. alle ASSUME Commands eintragen, die Sie zur Programmentwicklung benötigen (vgl. 5.); zum Arbeiten mit CTS Subroutines vgl. KOMMANDOSPRACHEN.
- ein Recovery File kreiert (temporärer oder permanenter File zum Sichern der CTS Umgebung; im Beispiel: CTS$FAUST).

3.1.3 Commands/Control Statements eingeben

Eingaben werden in der Regel angefordert, und zwar durch Ausgabe eines **Prompts** auf dem Bildschirm. Ein Prompt hat eines der beiden Formate

```
>
oder
>zeichenfolge>
```

wobei der Cursor direkt hinter (dem letzten) > steht.

CTS Prompts:

```
>->          CTS fordert Command an
>130 >       CTS fordert Zeile an (Number Mode)
>>>          Prescanner (vgl. 5.1) fordert Command an
```

EXEC Prompts:

```
>            EXEC fordert Control Statement an
```

FORTRAN Prompts:

```
>            Programm fordert Daten an (mit READ(5,...)...)
>->          Post Mortem Dump (FTN PMD) fordert Command an
```

Korrigieren im Eingabebereich: Vor dem <XMIT> kann im Eingabebereich mit Hilfe der in 2.2 beschriebenen Terminal Funktionen (z.B. <INSERT IN LINE>, <DELETE IN LINE>) beliebig korrigiert werden; der Cursor muß vor dem <XMIT> ans Ende des Eingabebereichs positioniert werden.

Tasten belegen mit Commands/Control Statements ist nicht möglich.

Zurückrufen von Commands/Control Statements (DEC: RECALL, IBM: RETRIEVE) ist nicht möglich; jedoch können Eingaben, die noch auf dem Bildschirm stehen, ggf. korrigiert und erneut übertragen werden. Tip: die unterste Bildschirmzeile (Zeile unter dem Prompt) wird nie durch Ausgaben überschrieben; sie ist vom "Scrolling" ausgenommen; ein Command/Control Statement, das in dieser Zeile steht, kann dadurch mehrfach übertragen werden.

Groß/Kleinschreibung: kleine Buchstaben werden in allen Commands/Control Statements wie große interpretiert.

CTS Commands: Nach der Initialisierung von (bzw. Rückkehr in) CTS kann hinter dem Prompt >-> ein CTS Command eingegeben werden, z.B.

>->*new prog* <XMIT>

dabei muß der Command Name (im Beispiel NEW) unmittelbar hinter dem > stehen.

Erst nach Beendigung der Ausführung kann ein neues CTS Command eingegeben werden; die Beendigung wird oft nur durch ein neues Prompt angezeigt. Eine Liste aller CTS Commands erhalten Sie mit HELP, vgl. 3.3.1:

>->*help* <XMIT>
>...>*commands* <XMIT>

Abkürzungen: In CTS Commands können alle Schlüsselwörter (Command Names, Optionen, ...) i.a. durch die ersten 3 Zeichen abgekürzt werden, vgl. KOMMANDOSPRACHEN.

Fortsetzung: nicht möglich

ECL Control Statements (@...): Sie können (bis auf einige Ausnahmen, wie z.B. @ADD, @FIN) nur im EXEC Mode eingegeben werden. Das **CTS Command XCTS** sichert die CTS Umgebung (Workarea, ASSUMEs, Variable) im Recovery File und schaltet in den EXEC Mode um.

>->*xcts* <XMIT>
>IN EXEC MODE
>

Jetzt kann ein ECL Control Statement eingegeben werden, z.B.

>@*delete,s faust.user$* <XMIT>

dabei muß @ unmittelbar hinter > stehen. Erst nach Beendigung der Ausführung kann ein neues ECL Control Statement eingegeben werden; bei der Beendigung wird in der Regel eine Endemeldung, immer aber ein neues Prompt ausgegeben. (Das neue Prompt kann jedoch auch Daten oder Direktiven für die Ausführung anfordern, z.B. @FTN, @MAP.) Eine Liste von ECL Control Statements steht in UP-4144, Appendix A; zur Ergänzung vgl. Text im symbolischen Element SYS$*LIB$.INHALT.

Abkürzungen: nicht zulässig.

Fortsetzung: Zur Eingabe eines ECL Control Statements können mehrere Eingabezeilen benutzt werden; das Zeichen ; wird am Ende der Eingabezeile als Fortsetzungszeichen interpretiert. Es darf nur hinter dem Zeichen , oder / verwendet werden, z.B.:

>@*ed,u anw$*beispiel.d-exec-add,;* <XMIT>
> *lib.comp-listing* <XMIT>

Das **ECL Control Statement @CTS** schaltet zurück in die zuvor verlassene CTS Umgebung (Vorsicht: nur möglich, falls von CTS assignierte Files nicht freigegeben, gelöscht, gepackt (vgl. 4.1.1) oder überschrieben wurden; falls nicht möglich: CTS mit @CTS,I erneut initialisieren, vgl. 3.1.2).

>*@cts* <XMIT>
>->

Neben den ECL Control Statements gibt es **weitere Control Statements:**

- **Network Control Statements** ($$ ...)
  eingebbar solange Verbindung aufgebaut, z.B.
  >*$$open demand* <XMIT>
  zum expliziten Beginnen einer Demand Session. Eine Liste aller Network Control Statements gibt es nicht; es wird empfohlen, nur die Network Control Statements zu verwenden, die im Benutzerhandbuch beschrieben sind.

- **Demand Symbiont Control Statements** (@@ ...)
  sind jeweils nur in einem spezifischen Zustand der Demand Session eingebbar, z.B. kann
  >*@@send* <XMIT>
  nur im Inactive Mode eingegeben werden. Eine Liste aller Demand Symbiont Control Statements steht in UP-4144, Appendix A.

- **Transparent Control Statements** (@@ ...)
  (spezielle ECL Control Statements mit @@ statt @) sind nur in Demand Runs eingebbar, z.B.
  >*@@asg,u perm-file* <XMIT>
  zum Kreieren eines permanenten Files während der Ausführung eines Programms, das zu einem späteren Zeitpunkt auf ihn zugreift (schlechter Arbeitsstil!). Eine Liste aller Transparent Control Statements steht in UP-4144, Appendix A.

Obige Control Statements können nur am Terminal eingegeben werden (sie dürfen nicht in einem ADD File/Element stehen); ihre Eingabe ist auch ohne Prompt möglich, vgl. Tabelle 3-1 (Ausgabe ggf. vorher mit <MSG WAIT> unterbrechen). Bei Eingabe mit Prompt wird nach Beendigung der Ausführung ein neues Prompt > ausgegeben; dieses Prompt wird jedoch oft vor die Ausgaben des Control Statements gesetzt (Synchronisationsproblem).

Abkürzungen: nicht zulässig
Fortsetzung: nicht möglich

Die Tabelle 3-1 beschreibt den Weg eines Commands/Control Statements durch die verschiedenen Softwareschichten von DCP/40 (TELCON) und SPERRY 1100/60 (Symbiont, EXEC, CTS), die es entweder bearbeiten oder weiterreichen.

Zustand Demand / Eingabe Run	CTS Prompt >-->	EXEC Prompt >	kein Prompt
$$ ... Network Control Statement	TELCON Komponente in DCP/40: Ausführung falls nicht möglich: >$$ERROR ...		
@@ ... Demand Symbiont Control Statement	Symbiont: Ausführung		
oder- Transparent Control Statement	Symbiont/EXEC: Ausführung; falls -nicht möglich: >-@@ERROR ... -beendet: >-@@COMPLETE		
@ADD ...	Symbiont: Standard Read File Zeiger umsetzen, @ADD ... durch 1. Zeile aus File/Element ersetzen		Symbiont: >**WAIT LAST INPUT IGNORED**
@EOF	Symbiont/CTS: >@EOF IGNORED >-->	EXEC: >@EOF IGNORED -IN CONTROL MODE >	
@FIN	CTS/EXEC: CTS verlassen Run beenden	EXEC: Run beenden	
@ ... ECL Control Statement	Symbiont: >STATEMENT NOT IN CLIST >	EXEC: Ausführung	
... CTS Command	CTS [/EXEC]: Ausführung	EXEC: >DATA IGNORED -IN CONTROL MODE >	

Tabelle 3-1: Bearbeitung von Terminal Eingaben

3.1.4 Demand Run beenden

Ein Demand Run wird sowohl in CTS als auch im EXEC Mode mit dem **ECL Control Statement @FIN** beendet:

CTS:
>->*@fin* <XMIT>

EXEC Mode:
>*@fin* <XMIT>

CTS führt ein implizites XCTS Command (vgl. 3.1.3) aus, bevor der Run von der EXEC beendet wird. Nach der Ausgabe der Abschlußinformation des Runs ist die Demand Session im Inactive Mode (*TERMINAL INACTIVE*).

3.1.5 Demand Session beenden, Verbindung abbauen

Die Verbindung muß immer dann abgebaut werden, wenn das Terminal längere Zeit nicht benutzt wird oder eine andere Verbindung aufgebaut werden soll. Der Abbau (Sign-off) erfolgt durch

UTS20: >*$$soff* <XMIT>

DEVELNET Terminals: <BREAK> oder Ausschalten des Terminals, vgl. Bedienungsanleitung.

Beim Abbau der Verbindung wird die Demand Session ggf. implizit beendet. (Sie kann mit $$CLOSE auch explizit beendet werden.)

3.2 Unterbrechen, Fortsetzen, Abbrechen

Das Arbeiten mit diesen Funktionen ist zweistufig; sie können entweder auf Ein/Ausgaben oder auf die Programme angewendet werden, die die Ein/Ausgaben anfordern/erzeugen:

```
Programm beginnen
    .
    .
    Programm unterbrechen
    Programm fortsetzen
    .
    .
    Eingabe beginnen
        .
        .
    Eingabe beenden/abbrechen
    .
    .
    Ausgabe beginnen
        .
        .
        Ausgabe unterbrechen
        Ausgabe fortsetzen/abbrechen
        .
        .
    Ausgabe beenden
    .
    .
Programm beenden/abbrechen
```

Dies gilt nicht nur für Programme , sondern ganz analog auch für CTS Commands, ECL Control Statements und CTS Subroutines.

3.2.1 Programme, CTS Commands, CTS Subroutines, ECL Control Statements

Die Ausführung eines Programms, eines CTS Commands, einer CTS Subroutine oder eines ECL Control Statements wird

- abgebrochen mit >**@@x** *t* <XMIT>
  oder >**@@x** *tio* <XMIT>

Mit dem **Demand Symbiont Control Statement @@X TIO** (T: Termination, I: Input, O: Output) werden gleichzeitig anstehende Terminal Ein/Ausgaben abgebrochen. Dagegen ist

- Unterbrechen und
- Fortsetzen i.a. nicht möglich

Eine Ausnahme hiervon bilden dumpfähige FORTRAN Programme (mindestens eine Programmeinheit wurde mit @FTN Option F compiliert). Ein dumpfähiges FORTRAN Programm wird

- unterbrochen mit >@@x *c* <XMIT>

(c: contingency interrupt). Bei der Unterbrechung wird der interaktive FTN PMD (FORTRAN Post Mortem Dump) aufgerufen; es können FTN PMD Commands eingegeben werden (Prompt:>->, vgl. 5.3). Das unterbrochene Programm wird

- fortgesetzt mit >->*exit* <XMIT>

3.2.2 Ein/Ausgaben

Ausgaben auf das Terminal (z.B. von CTS Command LIST) erfolgen in der Regel fortlaufend (Scrolling) und meistens schneller als sie gelesen werden können. Die Ausgabe von Zeilen auf das Terminal wird

- unterbrochen mit <MSG WAIT>
- fortgesetzt mit >@@*cont* <XMIT>
- abgebrochen mit >@@x *o* <XMIT>

Die **Eingabe** von Zeilen wird

- abgebrochen mit >**manual* <XMIT> im CTS Number Mode
  mit > <XMIT> im ED INPUT Mode
  bzw.>@*eof* <XMIT> sonst

Das **ECL Control Statement @EOF** liefert beim Lesen vom Terminal im FORTRAN Programm die END Bedingung (z.B. READ (5,100,END=99)...).

3.2.3 Normieren des Terminals

Falls auch nach <KBOARD UNLOCK> keine Terminal Eingaben möglich sind oder aber Terminal Eingaben offensichtlich falsch ankommen, so ist das Normieren des Terminals i.a. nur durch Abbau der Verbindung möglich. Dazu reicht bei DEVELNET Terminals das Aus- und wieder Einschalten aus, an einem UTS20 muß nach dem Einschalten noch $$SOFF eingegeben werden:

>*$$soff* <XMIT>

3.3 Informieren

3.3.1 Leistung und Format von Commands/Control Statements

CTS Commands: Das **CTS Command HELP** gibt Informationen über Leistung und Format von CTS Commands am Terminal aus. HELP hat keine Parameter:

```
>->help <XMIT>
```

gibt auf Wunsch (TEACH) eine Anleitung zum Arbeiten mit HELP aus. Beispiel: Erläuterung von Leistung und Format des CTS Commands RUN:

```
>TEACH? - TYPE YES or TYPE A HELP COMMAND>explain run <XMIT>
.
. Leistung von RUN
.
>MORE EXPLANATION?>no <XMIT>
>MORE HELP?>syntax run <XMIT>
.
. Format von RUN
.
>MORE HELP?>exit <XMIT>
>RETURN TO CTS
```

Control Statements: Informieren am Terminal nicht möglich.

3.3.2 Verbrauchte CPU-Zeit

Es gibt kein Command/Control Statement (außer @FIN, vgl. 3.1.4), das die vom Run verbrauchte CPU-Zeit liefert. Es gibt lediglich das FORTRAN Unterprogramm SUP (in ANW$*RLIB) zur Abfrage der vom Run verbrauchten SUP's, wobei die CPU-SUP's die verbrauchte CPU-Zeit angeben.

3.3.3 Belegter Massenspeicher

Der QUIP Prozessor gibt neben Benutzer-Berechtigungen auch die Belegung des Massenspeicherbereichs zur Account-id des Runs am Terminal aus, vgl. 3.3.5.

3.3.4 Files des Massenspeicherbereichs

Das **ECL Control Statement @MFD**

>***@mfd,a*** <XMIT>

listet (im EXEC Mode) alle permanenten Files des Massenspeicherbereichs zur Account-id des Runs am Terminal auf (dauert einige Minuten).

3.3.5 Benutzer-Berechtigungen

Der QUIP Prozessor listet die wichtigsten Benutzer-Berechtigungen für die User-id (z.B. FAUST) und Account-id (z.B. MAGIE) am Terminal auf, vgl. BENUTZER-BERECHTIGUNGEN. Er wird (im EXEC Mode) mit dem **ECL Control Statement @QUIP** aufgerufen, z.B.

>***@quip*** <XMIT>
>***read account magie userid faust*** <XMIT>
.
. Benutzer-Berechtigungen
.
>***end*** <XMIT>

Weitere Benutzer-Berechtigungen (z.B. Input Generation) können mit @TSS abgefragt werden (LIST Direktive, vgl. BENUTZER-BERECHTIGUNGEN).

3.3.6 Nachrichten, Mitteilungen

Die **Nachrichten** des HRZ werden beim Beginn eines Demand Runs am Terminal ausgegeben; es ist nicht möglich, dies innerhalb eines Runs zu wiederholen.

Die **Mitteilungen** des HRZ können (in CTS) mit Hilfe der CTS Subroutines NEU und INFO am Terminal ausgegeben werden (vgl. Notiz NACHRICHTEN/MITTEILUNGEN); erforderliche Angaben werden mit den entsprechenden Erläuterungen angefordert:

>->***call neu*** <XMIT>
>->***call info*** <XMIT>

3.4 Paßwort ändern

Das Paßwort kann innerhalb eines Demand Runs mit Hilfe des **Transparent Control Statements @@PASSWD** pw-alt/pw-neu geändert werden, z.B.

>->***@@passwd keines/geheim*** <XMIT>
>PASSWORD REPLACED

Ein Paßwort darf aus max. 6 alphanumerischen Zeichen bestehen; Paßwörter, die mit $ beginnen, können vom Benutzer nicht geändert werden. Das Paßwort kann auch beim Beginn eines Demand Runs geändert werden:

>ENTER USERID/PASSWORD:
>***user-id/pw-alt/pw-neu*** <XMIT>

3.5 Drucken

Eine Liste aller Drucker (mit Standort und Bezeichnung) finden Sie in der Notiz DRUCKEN.

3.5.1 Bildschirminhalt drucken

Dies ist nur an dem an das Terminal angeschlossenen Matrixdrucker möglich.

UTS20/Matrixdrucker: Das Drucken eines Bildschirminhalts ist umständlich (z.B. müssen alle > per Hand gelöscht werden); es wird daher empfohlen, die Demand Session zu protokollieren, vgl. 3.5.2.

DEVELNET Terminal/Matrixdrucker: Drucker ein- und auf ON LINE schalten, dann <PRINT> bzw. äquivalente Taste drücken, vgl. VT100 Beschreibung in der Bedienungsanleitung.

3.5.2 Protokollieren einer Demand Session

Wenn an das Terminal ein Matrixdrucker angeschlossen ist, so können die Ein- und Ausgaben einer Demand Session gleichzeitig auf diesen Drucker ausgegeben werden; dies wird im folgenden **protokollieren** genannt.

Zunächst ist sicherzustellen, daß der Drucker eingeschaltet und auf ON LINE geschaltet ist. Werden an einem UTS20 zwei Demand Sessions geführt (nur möglich an UTS20 mit zwei Site-id's, vgl. 2.2), so kann jeweils nur eine davon protokolliert werden. Das Protokollieren wird

- begonnen mit >***@@prnt*** <XMIT>
- beendet mit >***@@nopr*** <XMIT>

3.5.3 File-Inhalt drucken

Im folgenden wird beschrieben, wie standardmäßig auf Drucker der SPERRY 1100/60 zugegriffen wird. Darüber hinaus kann mit Hilfe des Filetransfers (vgl. FILETRANSFER im Teil ALLGEMEINES) oder über ein Textverarbeitungssystem (vgl. z.B. Notiz ANW$*TUSTEP und Einzelschrift TUSTEP-Ausgabegeräte des HRZ) auf weitere Drucker des HRZ zugegriffen werden.

CTS

Gedruckt werden kann der Inhalt von DATA Files (z.B. DATEN) oder symbolischen Elementen (z.B. LIB.QUELLPROG). Breakpoint Files (vgl. 4.2.2) sollten nicht mit CTS gedruckt werden, da CTS die Vorschubinformation ignoriert. Der Inhalt muß zunächst mit OLD in die Workarea kopiert werden:

```
>->old daten. <XMIT>
bzw.
>->old lib.quellprog <XMIT>
```

und kann dann mit dem **CTS Command SITE** gedruckt werden:

```
>->site <XMIT>
>HDG?>irgendeine Kopfzeile <XMIT>
>RTN?>irgendeine Schlußzeile <XMIT>
>COPY?>1 <XMIT>
>SITE?>faust/u <XMIT>
```

Dabei wird angefordert mit

- HDG? Text für die Kopfzeilen (Headings) der Druckseiten
- RTN? Text für die Schlußseite (Return-to Message, wird vom Operator nicht ausgeführt)
- COPY? Anzahl der Kopien (Copies)
- SITE? Bezeichnung eines Ausgabegeräts; folgende Angaben sind möglich:

PR	Zeilendrucker im Rechnerraum
SHRZA1	Zeilendrucker der Außenstation 1
site-id	Site-id eines UTS20
user-id/U	User-id

Die auszugebenden Zeilen werden zunächst in einem Output File gepuffert. Er wird in die Queue für das Gerät bzw. für die User-id eingereiht; die Ausgabe auf Zeilendrucker wird vom Operator gesteuert; zur Steuerung der Ausgabe auf Matrixdrucker s.u.

EXEC

Gedruckt werden kann der Inhalt von speziellen permanenten Files, wie z.B. DATA Files oder Breakpoint Files, nicht aber der Inhalt von symbolischen Elementen. Das **ECL Control Statement @SYM** druckt den Inhalt des Files am angegebenen Gerät (site) aus. Es hat das Format

@SYM[,U] f,copy,site

mit

U	File nicht löschen
f	File Reference des permanenten Files
copy	Anzahl der Kopien
site	Bezeichnung des Ausgabegeräts; mögliche Angaben s. CTS

Im Gegensatz zum CTS Command SITE wird der File nicht kopiert, sondern lediglich die File Reference in die Queue eingetragen; der Inhalt sollte danach nicht mehr geändert werden. Ohne U Option wird der File nach dem Drucken gelöscht. Beispiel: Drucken des permanenten DATA Files DATEN am Zeilendrucker PR im Rechnerraum

>*@sym,u daten,1,pr* <XMIT>

Die Ausgabe auf Zeilendrucker wird vom Operator gesteuert, die Steuerung der Matrixdrucker wird im folgenden beschrieben.

Steuerung der Matrixdrucker an Terminals

Die EXEC verwaltet viele Queues für Output Files, insbesondere eine Queue für jede Site-id sowie eine Queue für jede User-id. Für die Ausgabe an einem UTS20 mit Matrixdrucker wird empfohlen, als SITE die zweite Site-id des Terminals (falls vorhanden) anzugeben; während des Druckens kann dann am ersten logischen Bildschirm weitergearbeitet werden. Für DEVELNET Terminals ist die Angabe einer Site-id unsinnig (vgl. 3.1.1), hier wird empfohlen als SITE eine Userid anzugeben; die Ausgabe des Output Files kann dann an einem beliebigen Terminal (mit Matrixdrucker) erfolgen.

Site-id: Output Files in einer Queue für eine Site-id können am Terminal mit dieser Site-id ausgegeben werden. Die Ausgabe kann nur im Inactive Mode einer Demand Session angestoßen werden (mit @@SEND; ggf. nach <XMIT> user-id/paßwort und @FIN eingeben); sie kann ggf. am angeschlossenen Matrixdrukker protokolliert werden (@@PRNT, vgl. 3.5.2), z.B.

```
.
.
>@fin <XMIT>
.
.
>*TERMINAL INACTIVE - OUTPUT FILE AVAIL*
>@@prnt <XMIT>
>@@send <XMIT>
.
.  Ausgabe aller Output Files aus Queue
.  für Site-id des Terminals
.
>*TERMINAL INACTIVE*
```

User-id/U: Output Files in einer Queue für eine User-id können unmittelbar nach Beendigung eines Demand Runs mit dieser User-id am Terminal mit @@SEND,U ausgegeben werden; die Ausgabe kann ggf. am angeschlossenen Matrixdrucker protokolliert werden (@@PRNT, vgl. 3.5.2), z.B.

```
>ENTER USERID/PASSWORD:
>faust/$eines <XMIT>
.
.
>@fin <XMIT>
.
.
>*TERMINAL INACTIVE*
>@@prnt   <XMIT>
>@@send,u <XMIT>
.
.  Ausgabe aller Output Files aus Queue
.  für User-id FAUST
.
>*TERMINAL INACTIVE*
```

4. Arbeiten mit Files

4.1 Massenspeicher-Files

Programme und Daten werden in Massenspeicher-Files gehalten; wenn Sie Programmentwicklung an der SPERRY 1100/60 machen wollen, müssen Sie einiges über Files und Massenspeicherbereiche wissen. Das Wichtigste ist im folgenden beschrieben, weitergehende Fragen behandeln die Kapitel FILES und LIBRARIES.

Jeder Benutzer (User-id) besitzt genau einen **Massenspeicherbereich** (oder ist zugriffsberechtigt zu einem); seine Account-id ist der Name dieses Massenspeicherbereichs. In diesem Massenspeicherbereich hält er seine permanenten Files. Für die Dauer des Runs kann er darüber hinaus temporäre Files kreieren.

File Reference: In Commands/Control Statements kann in der Regel das folgende vereinfachte Format einer File Reference zur Bezeichnung eines Files verwendet werden:

[qualifier*]filename

Der **Qualifier** ist für temporäre Files lediglich eine Verlängerung des Filenames, für permanente Files gibt er den Massenspeicherbereich an; Default ist im Demand Run die Account-id des Runs, also der eigene Massenspeicherbereich; ein Qualifier muß also nur angegeben werden, wenn auf permanente Files anderer Benutzer zugegriffen wird. Der **Filename** ist frei wählbar (max. 12 alphanumerische Zeichen).

In fast allen Commands/Control Statements kann die File Reference durch das Zeichen . abgeschlossen werden (z.B. LIST FILE, @ASG), in einigen muß sie durch das Zeichen . abgeschlossen werden (z.B. LIST SAVED, @DELETE).

Folgende Besonderheiten sind im **File-Handling** an der SPERRY 1100/60 zu beachten:

- **Kreation, Lebensdauer, Fileart:** Files müssen i.a. explizit kreiert werden; dabei wird festgelegt, ob der File permanent (Lebensdauer unabhängig von der des Runs) oder temporär (implizite Löschung am Ende des Runs) ist. Die Fileart (z.B. Library oder DATA File) wird erst beim ersten Schreiben in den File festgelegt.
- **Zugriffskoordination** für permanente Files erfolgt beim Assignieren/Freigeben; dies wird in der Regel implizit ausgeführt, kann aber auch explizit ausgeführt werden (@ASG, @FREE).
- **Zugriffsschutz** für permanente Files erfolgt über die File Merkmale private (nur Eigentümer kann zugreifen), public (alle können zugreifen), Leseschlüssel, Schreibschlüssel; read-only, write-only; ändern mit @CHG möglich.

Commands/Control Statements zum **Arbeiten mit Files:**

- **Kreieren** eines Files:

	CTS Command	ECL Control Statement
temporär :	CREATE,T f[,F70M, .]	@ASG,T f[,F70M, .]
permanent:	CREATE,U f[,F70M, .]	@ASG,U f[,F70M, .]

- **Informieren** über File:

CTS Command	ECL Control Statement
LIST FILE f.	@PRT,F f.

- **Umbenennen** eines Files ist nicht möglich
- **Kopieren** eines Files: Beide Files (f-i: input, f-o: output) müssen existieren.

CTS Command	ECL Control Statement
COPY f-i.,f-o.	@COPY f-i.,f-o.

- **Löschen** eines Files:

	CTS Command	ECL Control Statement
temporär :	PURGE f	@FREE f
permanent:	PURGE f	@DELETE f.

- **File-Bearbeitung:**

CTS Command	ECL Control Statement
USE, vgl. 5.2	@USE, vgl. 6.4

Dabei sind f, f-i, f-o durch die File References der entsprechenden Files zu ersetzen.

4.1.1 Files mit Programmen (Libraries)

Programme werden auf der SPERRY 1100/60 ausschließlich in **Libraries** (Program Files) gehalten. Eine Library besteht aus **Elementen**; jedes Element kann ein oder mehrere Programme oder einen Text enthalten; jedes Element hat einen **Typ**, der von der Art des Inhalts abhängt:

- Quellprogramme, Text: **symbolisches Element**, Typ S
- verschiebliche Programme (vom Compiler erzeugt): **verschiebliches Element**, Typ R (wie relocatable)
- ausführbares Programm (vom Binder erzeugt): **absolutes Element**, Typ A
- z.B. CTS Subroutine: **omnibus Element**, Typ O

Die vollständige Bezeichnung eines Elements einer Library hat in Commands/Control Statements das Format

[File Reference.]Element Reference

wobei die **Element Reference** in der Regel nur aus dem **Elementname** (max. 12 alphanumerische Zeichen) besteht. Elemente verschiedenen Typs können gleiche Elementnames haben. Nach dem Edieren, Compilieren und Binden eines Programms enthält die Library i.a. ein symbolisches, ein verschiebliches und ein absolutes Element mit gleichem Elementname. Der Default für die File Reference hängt vom Command/Control Statement ab.

Beim Löschen und Ersetzen von Elementen entstehen "Leichen" in der Library; die Elemente werden nämlich nur logisch gelöscht und belegen nach wie vor den Speicherplatz. Das ECL Control Statement @PRT,T informiert u.a. wieviel Speicherplatz die "Leichen" (syn. DELETED TEXT) belegen. Libraries, die laufend geändert werden, müssen daher in regelmäßigen Abständen **gepackt** werden (Neuaufbau; Entfernung der "Leichen"). Dies erfolgt in CTS mit dem **PACK Command** und im EXEC Mode mit dem **@PACK Control Statement** (s.u.).

Commands/Control Statements zum **Arbeiten mit Libraries und Elementen:**

- **packen** (Library):	
CTS Command	ECL Control Statement
PACK f.	@PACK f.
- **informieren** (Library):	
CTS Command	ECL Control Statement
LIST SAVED f.	@PRT,T f.
- **kreieren** (Element):	
CTS Command	ECL Control Statement
SAVE, vgl. 5.1	implizit, vgl. 6.
- **ersetzen** (Element):	
CTS Command	ECL Control Statement
REPLACE, vgl. 5.1	implizit, vgl. 6.
- **umbenennen** (Element):	
CTS Command	ECL Control Statement
-	@CHG,t f.e-a,.e-n
- **kopieren** (Element)	
CTS Command	ECL Control Statement
COPY,t f-i.e-i,f-o.e-o	@COPY,t f-i.e-i,f-o.e-o
- **löschen** (Element):	
CTS Command	ECL Control Statement
UNSAVE f.e	@DELETE,t f.e

Klein geschriebene Bezeichnungen sind wie folgt zu ersetzen:

t	Typ (S,R,A oder O) des Elements
f,f-i,f-o	File References der Libraries
e,e-a,e-n,e-i,e-o	entsprechende Element References

4.1.2 Files mit Daten (DATA Files)

Daten, die z.B. von FORTRAN oder SPSS Programmen gelesen werden sollen, werden in der Regel in **DATA Files** gehalten. DATA Files werden mit Hilfe eines Editors erzeugt (CTS, @ED, vgl. 5.1, 6.1). DATA Files können von FORTRAN Programmen sequentiell formatiert gelesen werden (BACKSPACE ist nicht möglich). Umgekehrt können Files, die von FORTRAN Programmen sequentiell formatiert geschrieben wurden, vom Editor (CTS, @ED) wie DATA Files gelesen werden.

Files, die von FORTRAN Programmen mit ACCESS='DIRECT' (in der OPEN Anweisung) erzeugt wurden, sollten nur mit FORTRAN Programmen weiterverarbeitet werden.

4.2 Standard Ein/Ausgabe Files

4.2.1 Standard Read File

Der **Runstream** eines Runs besteht aus den Eingaben, die nicht mit @@ oder $$ beginnen, wie z.B.

- ECL Control Statements
- Direktiven (z.B. für @MAP)
- CTS Commands
- Daten (z.B. für FORTRAN READ(5,...)...)

Der Runstream wird aus dem **Standard Read File** eingelesen, dem im Demand Run per Default die Tastatur des Terminals zugeordnet ist. Teile des Runstreams können aber auch in **ADD Files** (DATA Files) oder **ADD Elementen** (symbolische Elemente von Libraries) bereitgestellt werden; ADD Files/Elemente können mit Hilfe eines Editors (CTS, @ED; vgl. 5.1, 6.1) erstellt werden. Sie entsprechen ansatzweise den Kommandoprozeduren anderer Kommandosprachen wie z.B. REXX (IBM) oder DCL (DEC).

CTS Beispiel: Das ADD ELement D-CTS-ADD in der Library ANW$*BEISPIEL enthält ein FORTRAN Quellprogramm samt CTS Commands zum Compilieren, Binden und Starten sowie Daten für das Programm. Bitte überzeugen Sie sich mit

```
>->old   anw$*beispiel.d-cts-add <XMIT>
>->list <XMIT>
```

Diese Commands werden ausgeführt mit dem **CTS Command ADD:**

```
>->add   anw$*beispiel.d-cts-add <XMIT>
```

EXEC Beispiel: Das ADD Element D-EXEC-ADD in der Library ANW$*BEISPIEL enthält ein FORTRAN Quellprogramm samt ECL Control Statements zum Compilieren, Binden und Starten sowie MAP Direktiven und Daten für das Programm. Bitte überzeugen Sie sich in CTS mit

```
>->old   anw$*beispiel.d-exec-add <XMIT>
>->list <XMIT>
   ...
>->xcts <XMIT>
>IN EXEC MODE
```

Im EXEC Mode werden diese ECL Control Statements mit dem **ECL Control Statement @ADD** ausgeführt:

```
>@add   anw$*beispiel.d-exec-add <XMIT>
```

Zu jedem Run gehört ein Zeiger auf seinen Standard Read File; zu Beginn eines Demand Runs zeigt dieser Zeiger auf das Terminal. ADD bzw. @ADD setzt diesen Zeiger um; der Runstream wird danach zeilenweise aus dem angegebenen File bzw. Element gelesen; nachdem alle Zeilen des Files bzw. Elements gelesen sind, wird der Zeiger zurückgesetzt (auf den alten Wert; Schachtelung möglich).

Das ECL Control Statement @ADD kann auch anstelle von Daten (z.B. für FORTRAN READ(5,...)...) eingegeben werden, vgl. Abb. 4-1 und Tabelle 3-1.

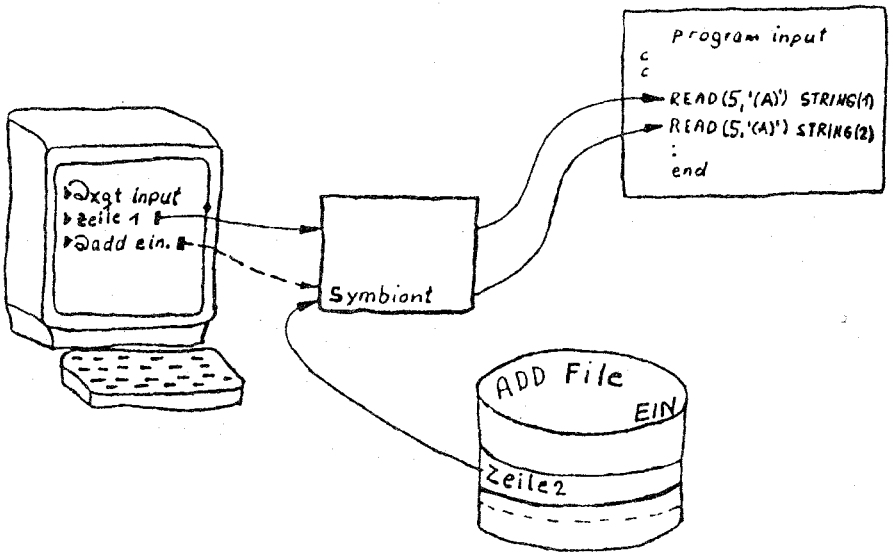


Abb. 4-1: Eingaben aus ADD File lesen

4.2.2 Standard Print File

Terminal Ausgaben wie z.B. die Ausgaben von

- ECL Control Statements, z.B.
  . Listings (@FTN, @MAP)
  . Informationen (@PRT, @MFD)
  . Fehler- und Endemeldungen
- CTS Commands, z.B.
  . Zeilen der Workarea (LIST, CHANGE)
  . Diagnostic Scan (RUN)
  . Informationen (HELP, LIST SAVED)
  . Fehler- und Endemeldungen
- Programmen, z.B.
  . FORTRAN WRITE(6,...)...

werden vom Control Statement/Command/Programm durch Ausgaben in den **Standard Print File** realisiert.

Dem Standard Print File ist im Demand Run per Default der Bildschirm des Terminals zugeordnet (Terminal Ausgaben können dann gleichzeitig protokolliert werden, vgl. 3.5.2). Terminal Ausgaben können aber auch in einen **Breakpoint File** (Print File) "umgeleitet" werden (am Bildschirm ist dann aber nichts mehr zu sehen!). Permanente Breakpoint Files können mit @SYM gedruckt werden, temporäre Breakpoint Files nur mit SITE (Vorschubinformation wird ignoriert), vgl. 3.5.3. Bei anderen Rechnern wird ein solcher Mechanismus i.a. gar nicht benötigt; die Parameter der Commands steuern, ob die Ausgaben auf das Terminal oder in einen File (der gedruckt werden kann) erfolgen.

CTS Beispiel: Das ausführbare (FORTRAN) Programm D-PRINT in der Library ANW$*BEISPIEL erzeugt Ausgaben mit WRITE(6,...)...; diese sollen gedruckt werden. Dazu wird zunächst ein permanenter File DRUCK-FILE kreiert

```
>->create,u  druck-file.,f70m, . <XMIT>
>->@@brkpt   print$,druck-file <XMIT>
```

Das **Transparent Control Statement @@BRKPT** lenkt die folgenden Terminal Ausgaben in den File DRUCK-FILE um. Das ausführbare Programm wird gestartet mit

```
>xqt   anw$*beispiel.d-print <XMIT>
```

Die Beendigung der Programmausführung wird lediglich durch ein neues Prompt > angezeigt. Danach können mit

```
>@@brkpt   print$ <XMIT>
```

die folgenden Standard Print File Ausgaben wieder auf das Terminal gelenkt werden. Der Breakpoint File DRUCK-FILE kann z.B. auf dem Zeilendrucker PR (im Rechnerraum) gedruckt werden mit dem Transparent Control Statement

```
>@@sym   druck-file,1,pr <XMIT>
```

EXEC Beispiel: Das Source Listing des FORTRAN Compilers soll gedruckt werden. Dazu wird zunächst ein permanenter File LISTING kreiert:

```
>@asg,u   listing <XMIT>
>@brkpt   print$,listing <XMIT>
```

Das ECL Control Statement @BRKPT lenkt die folgenden Terminal Ausgaben in den File LISTING um. Der FORTRAN Compiler wird aufgerufen mit

```
>@ftn,s   anw$*beispiel.d-prog,tpf$.prog <XMIT>
>@eof <XMIT>
```

Die Beendigung der Compilierung wird lediglich durch ein neues Prompt > angezeigt. Danach können mit

```
>@brkpt   print$ <XMIT>
```

die folgenden Standard Print File Ausgaben wieder auf das Terminal gelenkt werden. Der Breakpoint File LISTING kann z.B. auf dem Zeilendrucker PR (im Rechnerraum) gedruckt werden mit

```
>@free   listing <XMIT>
>@sym    listing,1,pr <XMIT>
```

Zu jedem Run gehört ein Zeiger auf seinen Standard Print File; zu Beginn eines Demand Runs zeigt dieser Zeiger auf das Terminal. @@BRKPT bzw. @BRKPT setzt diesen Zeiger um; die Standard Print File Ausgaben werden danach in den angegebenen File geschrieben. [@]@BRKPT PRINT$ (ohne File Reference) setzt diesen Zeiger zurück (auf das Terminal, Schachtelung nicht möglich).

Das ECL Control Statement @BRKPT kann mit Hilfe der Routine FACSF auch von einem FORTRAN Programm abgesetzt werden, vgl. Abb. 4-2.

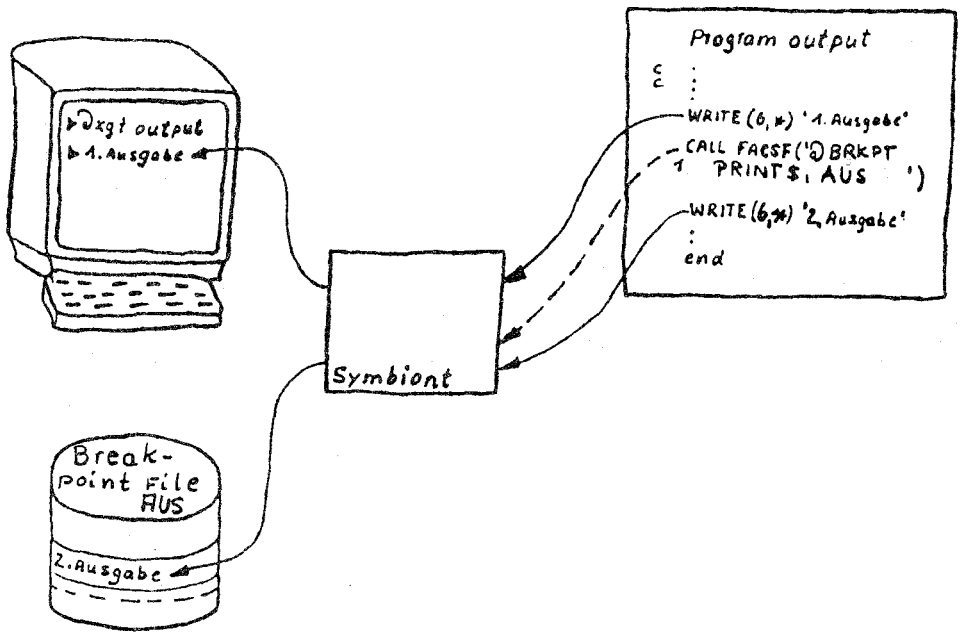


Abb. 4-2: Terminal Ausgaben in Breakpoint File umlenken

5. Programmentwicklung (FORTRAN) in CTS

Zur Programmentwicklung stehen mehrere höhere Programmiersprachen zur Verfügung (z.B. FORTRAN, PASCAL, COBOL, PL/I). Dieser Abschnitt stellt exemplarisch die Programmentwicklungsumgebung für die Programmiersprache **FORTRAN 77** vor; er ist lediglich als Einstieg konzipiert; weitergehende Informationen (z.B. langfristige Speicherung verschieblicher und ausführbarer Programme) finden Sie in der Notiz FORTRAN. Zu den anderen Programmiersprachen gibt es entsprechende Notizen.

Abb. 5-1 beschreibt das Edieren sowie das Compilieren, Binden und Starten eines Hauptprogramms (ggf. mit Unterprogrammen) mit Hilfe des RUN Commands; es faßt die Commands COMPILE, MAP und XQT zusammen. Das RUN Command ist für einfache Probleme ausreichend, für komplexere Probleme müssen die Commands einzeln verwendet werden. Die nachfolgenden Abschnitte enthalten weitere Erläuterungen zu Abb. 5-1.

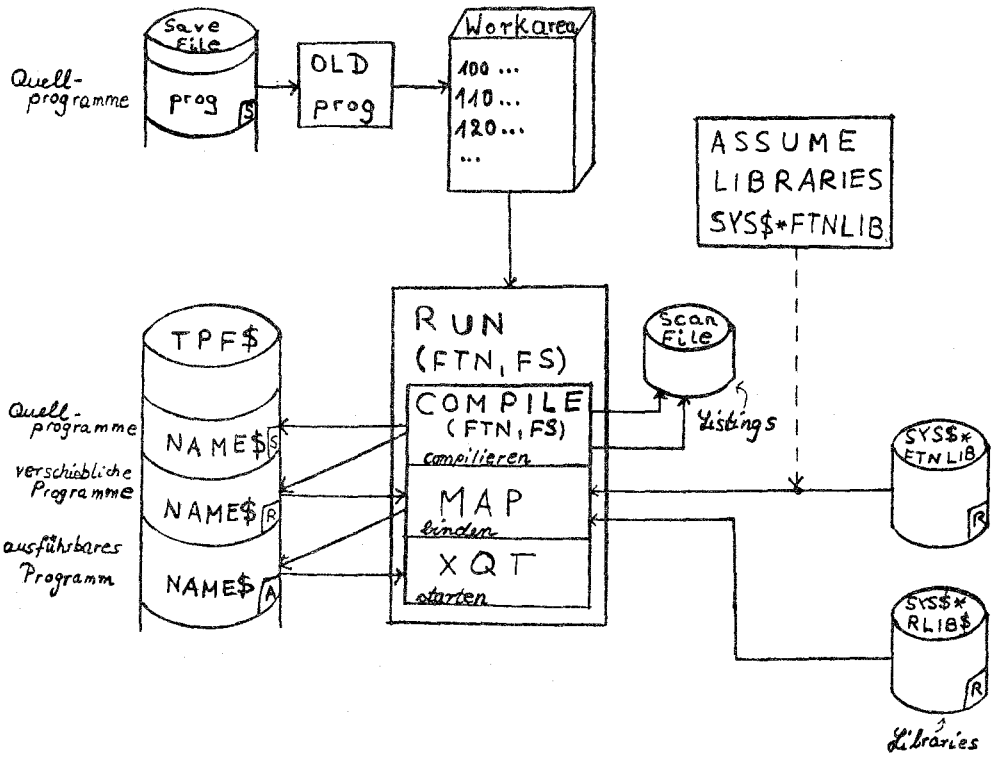


Abb. 5-1: Programmentwicklung (FORTRAN) in CTS

Die Arbeitsweise vieler CTS Commands wird durch **ASSUMEs** festgelegt. Einige dieser ASSUMEs haben Defaults (z.B. ASSUME PROGRAM), andere nicht (z.B. ASSUME LIBRARIES). Jedes ASSUME kann mit dem entsprechenden ASSUME Command überschrieben oder mit dem STATUS Command abgefragt werden. Beispiel:

>->*assume program xyz* <XMIT>

legt fest, daß als Save File von nun an XYZ (Fileart: Library; muß bereits existieren) verwendet wird.

>->*status program* <XMIT>

gibt die File Reference des Save Files am Terminal aus.

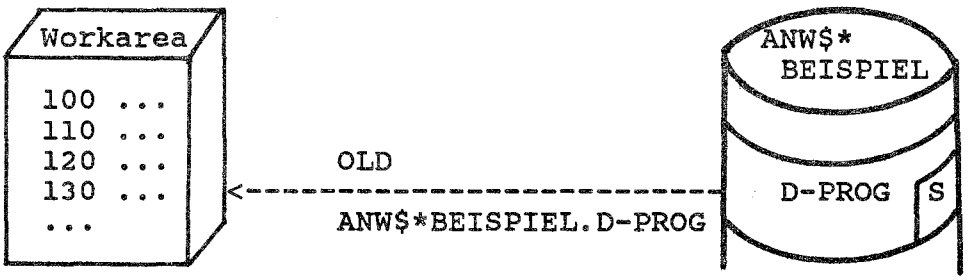
Ein ASSUME bleibt in der Regel bis zum nächsten entsprechenden ASSUME Command in Kraft; es empfiehlt sich daher, ASSUME Commands, die am Anfang eines Demand Runs immer ausgeführt werden müssen, wie z.B. ASSUME LIBRARIES, in das symbolische Element USER$ des Save Files zu schreiben (die Commands werden dann bei der Initialisierung von CTS implizit ausgeführt, vgl. 3.1.2).

5.1 Edieren (Quellprogramme)

Zu CTS gehört ein eingebetteter, **zeilenorientierter Editor**; seine Leistungen werden mit Hilfe von CTS Commands angefordert. Eine ausführliche Beschreibung finden Sie im Kapitel TEXTVERARBEITUNG MIT CTS. Die wichtigsten Funktionen zum Edieren von Quellprogrammen werden hier beschrieben. Darüber hinaus gibt es innerhalb von CTS noch einen **FORTRAN Prescanner**. Er formatiert jede eingegebene Zeile und prüft sie auf syntaktische Richtigkeit (vgl. Notiz FORTRAN).

Ediert werden können **Quellprogramme** (Texte) nur in der **Workarea** von CTS. Sie wird bei der Initialisierung von CTS (vgl. 3.1.2) bereitgestellt. Sie ist temporär, d.h. sie wird spätestens bei Beendigung des Runs (@FIN) gelöscht. In die Workarea können Zeilen eingetragen und/oder der Inhalt von symbolischen Elementen oder DATA Files kopiert werden (OLD, MERGE), z.B.

```
>->old  anw$*beispiel.d-prog <XMIT>
>->list <XMIT>                          (Auflisten der Workarea)
```



Jede Zeile hat eine positive (max. 6-stellige) **Zeilennummer** (≤ 262141). Jede Eingabe, die mit einer Zeilennummer beginnt, wie z.B.

```
>->155 C   hier wird ein Kommentar eingefuegt <XMIT>
```

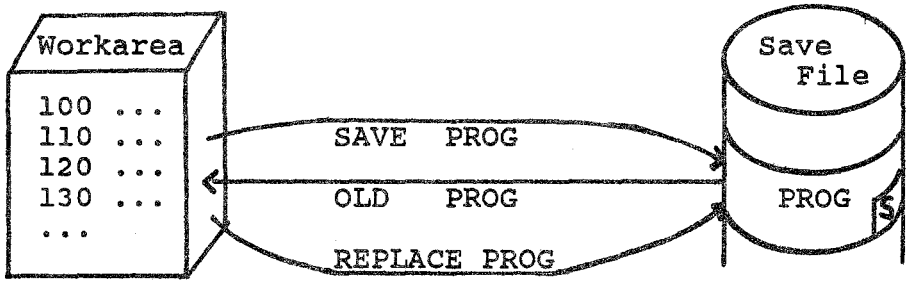
wird von CTS in die Workarea eingetragen und nicht als Command interpretiert. Existiert bereits eine Zeile mit dieser Zeilennummer, so wird sie überschrieben. Dies wird in der Regel zum **einfügen** und **ändern von Zeilen** (die bereits mit Zeilennummer auf dem Bildschirm stehen, z.B. nach LIST) verwendet. Das **Eintragen von Quellprogrammen** erfolgt in der Regel im **Number Mode**. Beispiel:

```
>->new  prog <XMIT>                     (Loeschen der Workarea)
>->number <XMIT>                        (Number Mode einschalten)
>100 > .
>110 > .
       Eingeben der
       Quellprogramme
       .
>510 > .
>520 >*manual <XMIT>                    (Number Mode ausschalten)
>->save  prog <XMIT>                    (Quellprogramme sichern)
```

Obiges **SAVE Command** kreiert im Save File (permanenter File, der bei der Initialisierung von CTS bereitgestellt wird, vgl. 3.1.2) ein symbolisches Element PROG und kopiert den Inhalt der Workarea hinein; die Workarea wird dadurch nicht geändert. Der Inhalt des Elements PROG kann später mit dem **OLD Command** in die Workarea zurück kopiert und ediert werden:

```
>->old     prog <XMIT>
   ... edieren ...
>->replace prog <XMIT>
```

Das **REPLACE Command** sichert den edierten Inhalt (Element wird ersetzt).



Hinweis: Beim "Kopieren in die Workarea" mit OLD oder MERGE werden die Zeilen nicht abgeschrieben, sondern es werden nur Zeiger auf die Zeilen (File-Adressen) angelegt. Das Verändern des Files außerhalb von CTS (z.B. durch @PACK) zerstört den Inhalt der Workarea (Inhalt muß mit NEW oder OLD gelöscht werden).

Groß/Kleinschreibung ist per Default (ASSUME ASCII ON) möglich. Der FORTRAN Compiler interpretiert kleine Buchstaben - außer in CHARACTER Konstanten - wie große (Erweiterung von FORTRAN 77). Vorsicht: der FORTRAN Compiler der IBM 4361 akzeptiert nur Großbuchstaben.

Tabelle 5-1 listet die wichtigsten Commands zum Edieren von Quellprogrammen auf; das angegebene Format ist nicht immer vollständig; es kann weitere Parameter geben (z.B. für CHANGE). Weitere Informationen liefert >->*help* <XMIT> (vgl. 3.3.1). Die wichtigsten Parameterwerte finden Sie in Tabelle 5-2.

CTS Command	Funktion
NEW d	Workarea löschen, Default für d in SAVE und REPLACE setzen
OLD [num] d [l]	Workarea löschen, [Zeilenbereich l von] d hineinkopieren, Default für d in SAVE und REPLACE setzen
MERGE [num] d [l]	[Zeilenbereich l aus] d in Workarea kopieren
SAVE [d] [l]	[Zeilenbereich l der] Workarea sichern in d (Kreation von d)
REPLACE [d] [l]	[Zeilenbereich l der] Workarea sichern in d (Ersetzen von d)
NUMBER [num]	Number Mode einschalten
*MANUAL	Number Mode ausschalten
LIST [l] [c]	Bereich (Spaltenbereich c in Zeilenbereich l) am Terminal auflisten
LOCATE 's' [l] [c]	Zeichenfolge suchen [in Bereich]
CHANGE '[s1]'[s2]' [l] [c]	Zeichenfolge austauschen [in Bereich]
DELETE l [c]	Bereich löschen
MOVE l [num]	Zeilenbereich umstellen
DITTO l [num]	Zeilenbereich kopieren
RESEQUENCE [num] [l]	Zeilenbereich neu numerieren

Tabelle 5-1: CTS Commands zum Edieren

Parameter Formate	Erläuterungen	Beispiele
d	File oder Element	
f.e	symb. Element e der Library f	LIB.TEXT
e	symb. Element e des Save Files	PROG
f.	DATA File f	DATEN.
l	Zeilenbereich	
ALL	alle Zeilen	ALL
n1,n2	Zeilen zwischen n1 und n2	100,240
n+k	k Zeilen nach Zeile n	230+20
n	Zeile n	523
c	Spaltenbereich	
(m1,m2)	Spalte (Position) m1 bis m2	(7,72)
(m)	Spalte m	(6)
num	Zeilennumerierung	
i,j	Anfang: i, Schrittweite: j	1500,100
i	Anfang: i, Schrittweite: 10	2370
,j	Anfang: 100, Schrittweite: j	,50
s,s1,s2	Zeichenfolgen	CALL

Tabelle 5-2: Parameter der CTS Commands

5.2 File-Bearbeitung

Direkte Files müssen, sequentielle Files können vom Programm mit Hilfe der OPEN Anweisung geöffnet werden. Dabei kann der **Unit Number** (z.B. 20) die File Reference des Files (z.B. DATEN) zugeordnet werden:

```
OPEN (UNIT=20, FILE='DATEN', ...)
```

Erfolgt die Zuordnung nicht im Programm, so muß sie vor dem Start des ausführbaren Programms mit dem **CTS Command USE** erfolgen, z.B.

>->*use 20,daten* <XMIT>

ordnet der Unit Number 20 (genauer: dem internen Filename 20) die File Reference DATEN zu.

Standard Ein/Ausgabe: Für die folgenden Unit Numbers gelten Default Zuordnungen:

UNIT=5 : Standard Read File
(Terminal oder ADD File/Element, vgl. 4.2.1)
UNIT=6 : Standard Print File
(Terminal oder Breakpoint File, vgl. 4.2.2)

Diese Zuordnungen können nur mit OPEN, nicht aber mit USE überschrieben werden.

5.3 Compilieren, Binden, Starten

Problem: Das Hauptprogramm und alle explizit aufgerufenen Unterprogramme (außer INTRINSICs) stehen in der Workarea (z.B. nach dem Edieren oder nach OLD, vgl. 5.1). Die Programme sollen compiliert, gebunden und gestartet werden.

Lösung: Das **CTS Command RUN** führt hintereinander die **CTS Commands COMPILE** (compilieren), **MAP** (binden) und **XQT** (starten) aus. Fürs Binden muß zuvor festgelegt werden, daß auch die **FORTRAN Library** SYS$*FTNLIB nach Unterprogrammen zu durchsuchen ist (per Default werden nur TPF$ und SYS$*RLIB$ durchsucht):

>->*assume libraries sys$*ftnlib* <XMIT>

Das CTS Command

>->*run (ftn,fs)* <XMIT>

kopiert zunächst die **Quellprogramme** aus der Workarea in das symbolische Element TPF$.NAME$ (die temporäre Library TPF$ wird beim Beginn eines Runs implizit kreiert).

Compilieren: Der Inhalt des symbolischen Elements TPF$.NAME$ wird vom **FORTRAN Compiler** (FTN) mit FS Options (F: dumpfähig, S: Source Listing) compiliert. Die erzeugten **verschieblichen Programme** werden in das verschiebliche Element TPF$.NAME$ geschrieben.

Binden: Der Inhalt des verschieblichen Elements TPF$.NAME$ wird (vom MAP Prozessor) mit verschieblichen Programmen aus der FORTRAN Library SYS$*FTNLIB und der System Relocatable Library SYS$*RLIB$ zusammengebunden; das erzeugte **ausführbare Programm** wird in das absolute Element TPF$.NAME$ geschrieben.

Starten: Das ausführbare Programm in TPF$.NAME$ wird gestartet; es kann erneut gestartet werden mit

>->*xqt* <XMIT>

(XQT f.e startet das ausführbare Programm im absoluten ELement f.e)

Listings und Messages von Compilieren und Binden werden von CTS in den **Scan File** geschrieben. Der Scan File ist ein temporärer File; er wird von CTS bei Bedarf implizit kreiert. Auf die Anfrage

>*DIAGNOSTIC SCAN?>*yes* <XMIT>

gibt CTS einen Auszug aus dem Scan File am Terminal aus. Das vollständige **Compiler** bzw. **Binder Listing** kann nach RUN mit

>->*scan ,ftn* <XMIT>
bzw.
>->*scan ,map* <XMIT>

in die Workarea kopiert und z.B. mit LIST am Terminal aufgelistet oder mit SITE gedruckt werden (Scan Mode). Der Scan Mode wird beendet mit

>->*edit* <XMIT>

Danach steht der ursprüngliche Inhalt der Workarea wieder zur Verfügung.

Der interaktive **FORTRAN Post Mortem Dump** (FTN PMD) wird implizit aufgerufen, wenn ein dumpfähiges ausführbares Programm durch einen Laufzeitfehler (z.B. Guard Mode Error) abgebrochen wird. Er meldet sich mit

```
>***** ENTER FTN PMD *****
>->
```

Sein Prompt ist leider mit dem CTS Prompt identisch; er fordert FTN PMD Commands an:

```
>->dump !<XMIT>
```

listet die Werte von allen Variablen und Feldern der dumpfähigen Programmeinheiten am Terminal auf. Beendet wird FTN PMD mit

```
>->exit <XMIT>
```

ASSUMEs für RUN: Die oben beschriebene Arbeitsweise von RUN kann mit zahlreichen ASSUMEs verändert werden; diese ASSUMEs beeinflussen teilweise auch andere CTS Commands, sie beeinflussen sich teilweise sogar gegenseitig. Falls - wie vorausgesetzt - das Hauptprogramm in der Workarea steht, kommen Sie i.a. mit folgenden ASSUMEs aus:

```
ASSUME   LIBRARIES     [f,...,]SYS$*FTNLIB
ASSUME   RELOCATABLE   f.e
ASSUME   XQT           f.e
```

wobei f und e durch die entsprechenden File und Element References zu ersetzen sind.

ASSUME LIBRARIES legt die Libraries fest, die beim Binden neben TPF$ und der System Relocatable Library SYS$*RLIB$ nach (verschieblichen, d.h. bereits compilierten) Unterprogrammen durchsucht werden sollen; SYS$*FTNLIB muß, weitere Libraries können angegeben werden, z.B.

```
ASSUME  LIBRARIES  ANW$*IMSL,SYS$*FTNLIB
```

falls Unterprogramme aus der IMSL Library ANW$*IMSL aufgerufen werden, vgl. LIBRARIES. **ASSUME RELOCATABLE** legt das verschiebliche Element fest, in das der Compiler die erzeugten verschieblichen Programme schreibt (Output für COMPILE, Input für MAP). **ASSUME XQT** legt das absolute Element fest, in das der Binder das erzeugte ausführbare Programm schreibt (Output für MAP, Default für XQT).

6. Programmentwicklung (FORTRAN) im EXEC Mode

Zur Programmentwicklung stehen mehrere höhere Programmiersprachen zur Verfügung (z.B. FORTRAN, PASCAL, COBOL, PL/I). Dieser Abschnitt stellt exemplarisch die Programmentwicklungsumgebung für die Programmiersprache **FORTRAN 77** vor; er ist lediglich als Einstieg konzipiert; weitergehende Informationen (z.B. langfristige Speicherung verschieblicher und ausführbarer Programme) finden Sie in der Notiz FORTRAN. Zu den anderen Programmiersprachen gibt es entsprechende Notizen.

Abb. 6-1 beschreibt das Edieren, Compilieren, Binden und Starten eines Hauptprogramms (ggf. mit Unterprogrammen). Die nachfolgenden Abschnitte enthalten weitere Erläuterungen zu Abb. 6-1.

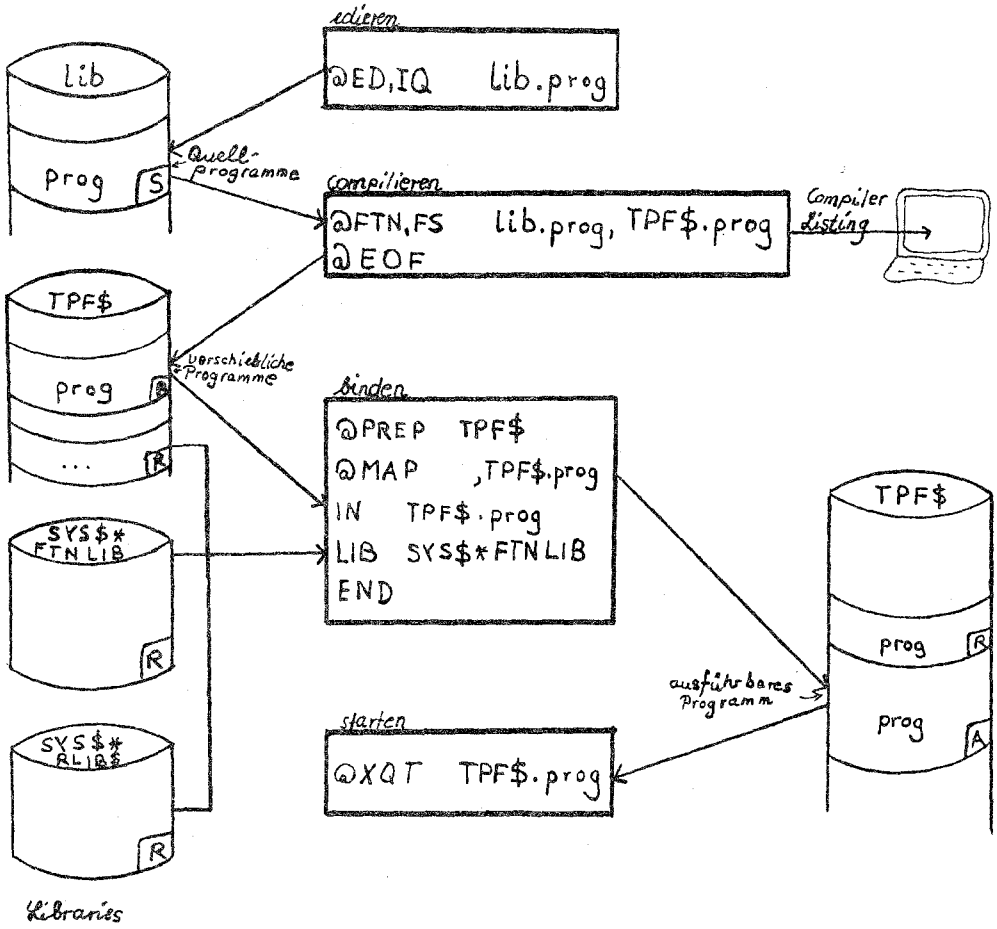


Abb. 6-1: Programmentwicklung (FORTRAN) im EXEC Mode

6.1 Edieren (Quellprogramme)

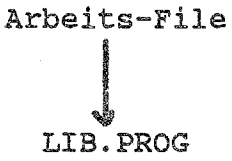
Quellprogramme können entweder in CTS (vgl. 5.1) oder mit dem ebenfalls **zeilenorientiert** arbeitenden **ED Prozessor** ediert werden. Eine ausführliche Beschreibung des ED Prozessors finden Sie im Kapitel TEXTVERARBEITUNG MIT ED. Die wichtigsten Funktionen zum Edieren von Quellprogrammen werden hier beschrieben. Für den Aufruf des ED Prozessors kann das folgende vereinfachte Format des **ECL Control Statements @ED** verwendet werden:

```
@ED,options   f.e
```

wobei f und e durch die entsprechende File und Element Reference des symbolischen Elements zu ersetzen sind; options s.u.

Beispiel: IQ Options zum Eintragen von Quellprogrammen

```
>@ed,iq   lib.prog <XMIT>
```

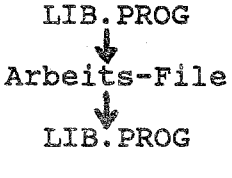



```
... Quellprogramme ...
...   eintragen    ...

>102:>exit <XMIT>
```

Beispiel: U Option zum Korrigieren von Quellprogrammen

```
>@ed,u   lib.prog <XMIT>
```




```
...  edieren  ...

>73:>exit <XMIT>
```

Ediert werden können nur Texte im **Arbeits-File** des Editors. Beim Aufruf mit I Option wird ein leerer Arbeits-File bereitgestellt, beim Aufruf mit U Option wird der Inhalt des angegebenen symbolischen Elements hineinkopiert; der Inhalt des Arbeits-Files wird beim EXIT ins angegebene symbolische Element (im Beispiel LIB.PROG) geschrieben.

Der ED Prozessor arbeitet zeilenorientiert; die **Zeilennummern** (1,2,3,...) geben Positionen von Zeilen innerhalb eines Textes an (z.B. hat die 52. Zeile die Zeilennummer 52). Löschen (DELETE) oder Einfügen (INPUT) von Zeilen ändert die Zeilennummern der nachfolgenden Zeilen. Ein Zeilenzeiger legt die sogenannte **Arbeitszeile** (current line, Default für viele ED Commands) fest. Die Zeilennummer der Arbeitszeile steht im **Prompt** des ED Prozessors, z.B.

```
>52I:>        im INPUT Mode
>52:>         im EDIT Mode
```

Im **INPUT Mode** (Default beim Aufruf mit I Option) können Zeilen in den Arbeits-File eingetragen werden; eine leere Eingabe (> <XMIT>) schaltet in den EDIT Mode um. Im **EDIT Mode** (Default beim Aufruf mit U Option) können **ED Commands** zum Edieren des Textes eingegeben werden; ED Command INPUT schaltet in den INPUT Mode um.

Groß/Kleinschreibung wird ermöglicht durch die Q Option bzw. durch Edieren (mit U Option) eines symbolischen Elements mit Groß/Kleinschreibung. Der FORTRAN Compiler interpretiert kleine Buchstaben - außer in CHARACTER Konstanten - wie große (Erweiterung von FORTRAN 77). Vorsicht: der FORTRAN Compiler der IBM 4361 akzeptiert nur große Buchstaben.

Tabelle 6-1 listet den Aufruf sowie die wichtigsten Commands des ED Prozessors auf. Das angegebene Format ist nicht immer vollständig; es kann weitere Paramater und weitere Formate geben. Eckige Klammern enthalten Spaltenbereiche; sie werden - abweichend von den Konventionen des Kapitels - nicht zur Kennzeichnung von optionalen Angaben verwendet. In Tabelle 6-1 bezeichnet

d einen File oder ein Element.

Für d ist eines der folgenden Formate zu verwenden:

f.e	symbolisches Element e der Library f
e	symbolisches ELement e der Library TPF$
f.	DATA File oder Breakpoint File f

ED Aufruf/Commands	Funktion
@ED,IQ d	Edieren beginnen; leerer Arbeits-File; INPUT Mode
@ED,U d	Edieren beginnen; d in Arbeits-File kopieren; EDIT Mode
ADD+ d n1 n2	Zeilen n1 bis n2 von d in Arbeits-File (hinter Arbeitszeile) kopieren
EXIT	Edieren beenden; Arbeits-File in d kopieren und löschen
INPUT	INPUT Mode einschalten
leere Eingabe	INPUT Mode ausschalten
n	neue Arbeitszeile: n
+n	neue Arbeitszeile: alte+n
-n	neue Arbeitszeile: alte-n
INLINE n	ändern in Zeile n
RETYPE+	ersetzen der Arbeitszeile
INSERT+	Zeile einfügen hinter Arbeitszeile
PRINT n1 n2 PRINT!	Zeilen n1 bis n2 bzw. alle am Terminal auflisten
SITE n1 n2 site SITE site	Zeilen n1 bis n2, bzw. alle drucken am Gerät site (vgl. 3.5.3)
LOCATE,n[m1,m2] 's' LOCATE,n 's' LOCATE 's'	Zeichenfolge s suchen in Bereich (Zeilen: n bzw. alle ab Arbeitszeile, Spalten: m1 bis m2 bzw. alle)
CHANGE 's1's2'[m1,m2] n G CHANGE 's1's2' n G CHANGE 's1's2' ALL	Zeichenfolge s1 austauschen durch s2 in Bereich (Zeilen: n bzw. alle nach Arbeitszeile, Spalten: m1 bis m2 bzw. alle)
DELETE n1 n2	Zeilen n1 bis n2 löschen
MOVE n1 n2	Zeilen n1 bis n2 umstellen hinter Arbeitszeile
DITTO n1 n2	Zeilen n1 bis n2 kopieren hinter Arbeitszeile

Tabelle 6-1: @ED Control Statement/ED Commands

6.2 Compilieren (Quellprogramme)

Problem: Die Quellprogramme (Hauptprogramm ggf. mit Unterprogrammen) im symbolischen Element LIB.PROG sollen compiliert werden.

Lösung: Das **ECL Control Statement @FTN** ruft den **FORTRAN Compiler** auf, z.B.

```
>@ftn,fs   lib.prog,tpf$.prog <XMIT>
>FTN 11RA1   10/06/86 - 10:21(0,)
>@eof <XMIT>
```

compiliert die **Quellprogramme** im symbolischen Element LIB.PROG und schreibt die erzeugten dumpfähigen (F Option) **verschieblichen Programme** in das verschiebliche Element PROG der temporären Library TPF$ (sie wird beim Beginn des Runs implizit kreiert).

Das **Compiler Listing** (Source Listing, S Option) wird am Terminal ausgegeben. Das **Drucken** des Listings ist nur möglich durch

- Protokollieren der Demand Session an dem angeschlossenen Matrixdrucker mit @@PRNT (vgl. 3.5.2)
- Umlenken der Terminal Ausgaben in einen Breakpoint File mit @BRKPT (vgl. 4.2.2)

Hinweis: Zum Testen einzelner Unterprogramme wird empfohlen, diese in ein eigenes symbolisches Element (z.B. lib.up) zu schreiben und separat zu compilieren: @FTN,FS lib.up,TPF$.up (Binden erfolgt dann wie unten beschrieben).

6.3 Binden (verschiebliche Programme)

Problem: Die verschieblichen Programme (Hauptprogramm, ggf. mit Unterprogrammen) im verschieblichen Element TPF$.PROG sollen gebunden werden.

Lösung: Zunächst muß die temporäre Library TPF$ **gepreppt** werden:

>*@prep tpf$* <XMIT>

Dabei wird in TPF$ fürs Binden eine Tabelle Unterprogramm -> Element für die verschieblichen Programme angelegt. Danach wird mit dem **ECL Control Statement @MAP** der Binder (syn. MAP Prozessor) aufgerufen:

>*@map ,tpf$.prog* <XMIT>

Die **MAP Direktiven** (Direktiven fürs Binden) werden vom Terminal eingelesen:

>*in tpf$.prog* <XMIT>
>*lib sys$*ftnlib* <XMIT>
>*end* <XMIT>

Die IN Direktive gibt das verschiebliche Element an, in dem das Hauptprogramm steht. Die LIB Direktive legt die Libraries fest, die neben TPF$ und der System Relocatable Library SYS$*RLIB$ nach verschieblichen Programmen (Unterprogrammen) durchsucht werden sollen; die **FORTRAN Library** SYS$*FTNLIB muß immer angegeben werden, weitere Libraries können angegeben werden, z.B.

LIB ANW$*IMSL,SYS$*FTNLIB

falls Unterprogramme aus der IMSL Library ANW$*IMSL aufgerufen werden, vgl. LIBRARIES.

Das erzeugte **ausführbare Programm** wird ins absolute Element TPF$.PROG geschrieben (Parameter in @MAP).

Ein **Binder Listing** wird nur ausgegeben - und zwar am Terminal -, falls der Aufruf mit S (@MAP,S: Short Listing) oder L Option (@MAP,L: Long Listing) erfolgt. Das Short Listing ist i.a. schon sehr umfangreich; seine Interpretation sollte Experten vorbehalten bleiben.

6.4 File-Bearbeitung

Direkte Files müssen, sequentielle Files können vom Programm mit Hilfe der OPEN Anweisung geöffnet werden. Dabei kann der **Unit Number** (z.B. 20) die File Reference des Files (z.B. DATEN) zugeordnet werden:

```
OPEN (UNIT=20, FILE='DATEN', ...)
```

Erfolgt die Zuordnung nicht im Programm, so muß sie vor dem Start des ausführbaren Programms mit dem **ECL Control Statement @USE** erfolgen, z.B.

>*@use 20,daten* <XMIT>

ordnet der Unit Number 20 (genauer: dem internen Filename 20) die File Reference DATEN zu.

Standard Ein/Ausgabe: Für die folgenden Unit Numbers gelten Default Zuordnungen:

UNIT=5 : Standard Read File
(Terminal oder ADD File/Element, vgl. 4.2.1)
UNIT=6 : Standard Print File
(Terminal oder Breakpoint File, vgl. 4.2.2)

Diese Zuordnungen können nur mit OPEN, nicht aber mit @USE überschrieben werden.

6.5 Starten (ausführbares Programm)

Problem: Das ausführbare Programm im absoluten Element TPF$.PROG soll gestartet werden.

Lösung: Das **ECL Control Statement @XQT** startet das ausführbare Programm im angegebenen absoluten Element:

>*@xqt tpf$.prog* <XMIT>

Falls die Programmausführung durch einen Laufzeitfehler abgebrochen wird, so wird der interaktive FORTRAN Post Mortem Dump aufgerufen; es können FTN PMD Commands (z.B. DUMP ¦ oder EXIT) eingegeben werden, vgl. 5.3.

7. Beispiele für Demand Runs

Die folgenden Beispiele setzen voraus, daß die Demand Runs an einem UTS20 geführt werden.

7.1 CTS

Wenn das UTS20 frei ist, wird ggf. nach <XMIT> die Aufforderung ausgegeben

```
>ENTER USERID/PASSWORD:
>faust/$eines <XMIT>
 ...
```

Mit der Eingabe von Userid (FAUST) und Paßwort ($EINES) wird ein Demand Run begonnen und CTS initialisiert. Zuerst soll ein Quellprogramm in die Workarea eingetragen (Number Mode), korrigiert und in das symbolische Element PROG des Save Files kopiert werden:

```
>->new  prog <XMIT>
>->number <XMIT>
>100 >       program sinus <XMIT>
>110 >10     read(5,*) r <XMIT>
>120 >       if(r .lt. 0.) stop <XMIT>
>130 >       write(6,*) 'SIN(',r,')=',sin(r) <XMIT>
>140 >       goto 10 <XMIT>
>150 >       end <XMIT>
>160 >*manual <XMIT>
>->105 10    write(6,*)'Bitte REAL-Zahl eingeben' <XMIT>
>->110       read(5,*) r <XMIT>
>->save prog <XMIT>
```

Dieses Programm wird compiliert, gebunden und gestartet mit

```
>->assume  libraries  sys$*ftnlib <XMIT>
><58> WARNING - FILE SYS$*FTNLIB IS WRITE INHIBITED
>->run   (ftn,fs) <XMIT>
>COMPILING...
>Bitte REAL-Zahl eingeben
>3.14 <XMIT>
>SIN(  3.1400000    )=   .15926672-002
>Bitte REAL-Zahl eingeben
>-1. <XMIT>
>*DIAGNOSTIC SCAN?>yes <XMIT>
>1 @FTN,FS   TPF$.NAME$,TPF$.NAME$
 ...
>*END DIAGNOSTIC SCAN
```

Der Demand Run wird beendet mit

```
>->@fin <XMIT>
>IN EXEC MODE
>  RUNID: FAUST   ACCT: MAGIE   PROJECT: MAGIE
 ...
>*TERMINAL INACTIVE*
```

7.2 EXEC Mode

Wenn das UTS20 frei ist, wird ggf. nach <XMIT> die Aufforderung ausgegeben

```
>ENTER USERID/PASSWORD:
>faust/$eines <XMIT>
```

...

Mit der Eingabe von Userid (FAUST) und Paßwort ($EINES) wird ein Demand Run begonnen und CTS initialisiert. Mit

```
>->xcts <XMIT>
>IN EXEC MODE
```

wird in den EXEC Mode umgeschaltet. Zunächst wird ein permanenter File kreiert:

```
>@asg,u   lib <XMIT>
>I:002333 ASG complete
```

dann der ED Prozessor zum Eintragen eines Quellprogramms aufgerufen:

```
>@ed,iq   lib.prog <XMIT>
>ED  16R1E.  16R1B 16R1C-1 FRI-11/07/86-16:28:51-(,0)
>INPUT
>1I:>       program sinus <XMIT>
>2I:>10     read(5,*) r <XMIT>
>3I:>       if(r .lt. 0.) stop <XMIT>
>4I:>       write(6,*) 'SIN(',r,') = ',sin(r) <XMIT>
>5I:>       goto 10 <XMIT>
>6I:>       end <XMIT>
>7I:>
>EDIT
>6:>1 <XMIT>
>      program sinus
>1:>insert+ <XMIT>
>+>10    write(6,*) 'Bitte REAL-Zahl eingeben' <XMIT>
>2:>inline 3 <XMIT>
>+++10    read(5,*) r
>+>r  ! <XMIT>
>      read(5,*) r
>3:>exit <XMIT>
>END  ED.   LINES: 7 ASCII
```

EXIT kreiert in LIB ein symbolisches Element PROG und schreibt das eben erstellte Quellprogramm hinein. Jetzt wird der FORTRAN Compiler aufgerufen:

```
>@ftn,fs   lib.prog,tpf$.prog <XMIT>
>FTN  11R101   11/07/86-16:55(0,0)
>@eof <XMIT>
> 1.       program sinus
> 2. 10    write(6,*) 'Bitte REAL-Zahl eingeben'
> 3.       read(5,*) r
> 4.       if(r .lt. 0.) stop
> 5.       write(6,*) 'SIN(',r,') = ',sin(r)
> 6.       goto 10
> 7.       end
>END  FTN  16 IBANK  54 DBANK
```

Das erzeugte verschiebliche Programm wird ins verschiebliche Element PROG der temporären (zu Beginn des Runs implizit kreierten) Library TPF$ geschrieben. TPF$ muß vor dem Binden (@MAP) immer gepreppt (@PREP) werden:

```
>@prep   tpf$ <XMIT>
>FURPUR 28R3A  S74T11  11/07/86 16:55:52
>END PREP.  MAGIE*TPF$(0)  1 REL  1 ENTRY PT(S) NO DUP(S)

>@map  ,tpf$.prog <XMIT>
>Collector 31R2B (841126 1925:45) 1926  Nov 07 Fri 1656:05
>in    tpf$.prog <XMIT>
>lib   sys$*ftnlib <XMIT>
>end <XMIT>
>START = 015225, PROG SIZE(I/D)=6309/7344
>END MAP. ERRORS:0  TIME: 20.750   STORAGE: ...
```

Das ausführbare Programm wird ins absolute Element TPF$.PROG geschrieben (Parameter in @MAP). Es wird gestartet mit

```
>@xqt   tpf$.prog <XMIT>
>Bitte REAL-Zahl eingeben
>3.14 <XMIT>
>SIN(   3.1400000   ) =   .15926672-002
>Bitte REAL-Zahl eingeben
>-1. <XMIT>
```

Der Run wird beendet mit

```
>@fin <XMIT>
>  RUNID: FAUST   ACCT: MAGIE     PROJECT: MAGIE
  ...
>*TERMINAL INACTIVE*
```