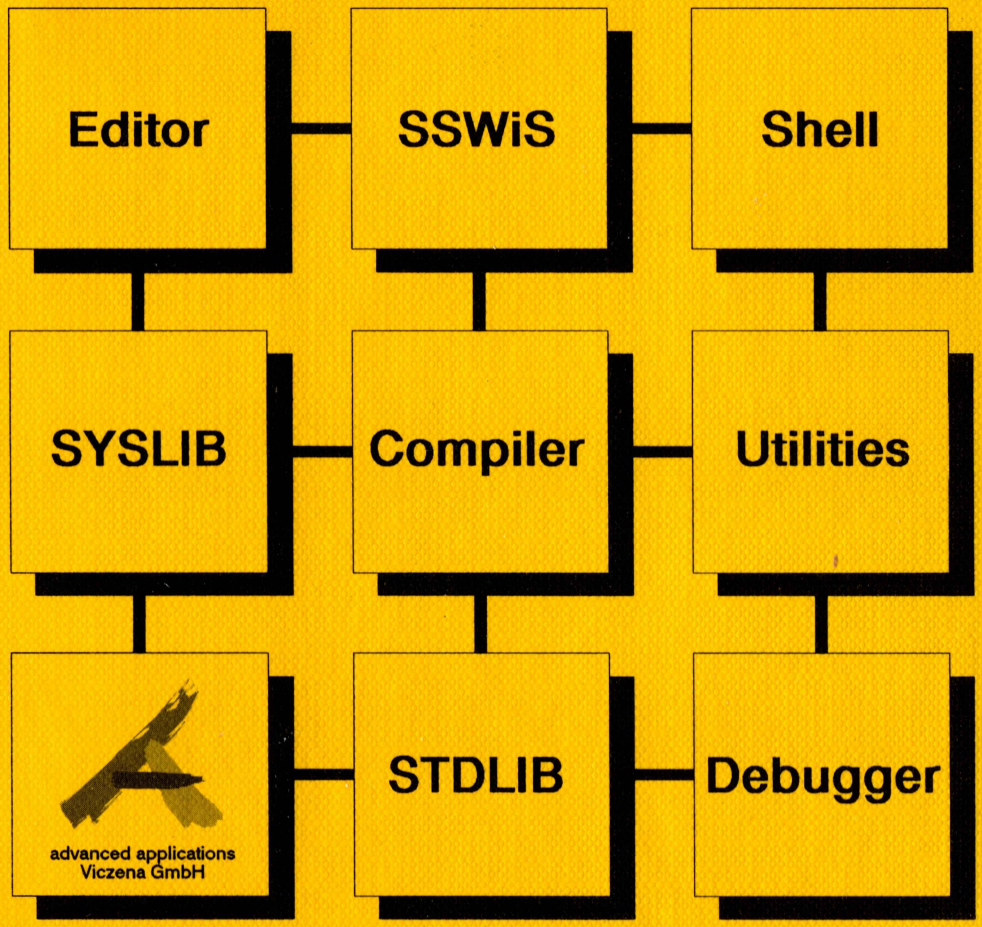
NEU! Jetzt mit Modulakurs

SPC Modula-2

Für ATARI ST und MEGA ST



SPC MODULA-2

für den ATARI ST
Benutzerhandbuch

Version 1.4

Advanced Applications Viczena GmbH

Haftungsausschluß

Es wurden alle erdenklichen Maßnahmen getroffen, um die Korrektheit dieser Dokumentation und der dazugehörigen Software zu gewährleisten. Da wir jedoch ständig Verbesserungen und Nacharbeiten an unseren Produkten vornehmen, können wir keine Gewähr für deren Vollständigkeit und Korrektheit übernehmen und schließen deshalb alle Gewährleistungsansprüche aufgrund von Fehlern der Software oder der Dokumentation aus.

Handelsmarken

GEM, GEMDOS sind Warenzeichen bzw. eingetragene Warenzeichen der Digital Research Inc.

MOTOROLA, MC68000 sind Warenzeichen bzw. eingetragene Warenzeichen der MOTOROLA Inc.

ATARI, 260ST, 520ST, 520ST+, TOS, MEGA ST sind eingetragene Warenzeichen der ATARI Corp.

Teile dieses Sprachsystems wurden am Institut für Informatik der Eidgenössischen Technischen Hochschule (ETH) Zürich von Niklaus Wirth und seinem Team entwickelt.

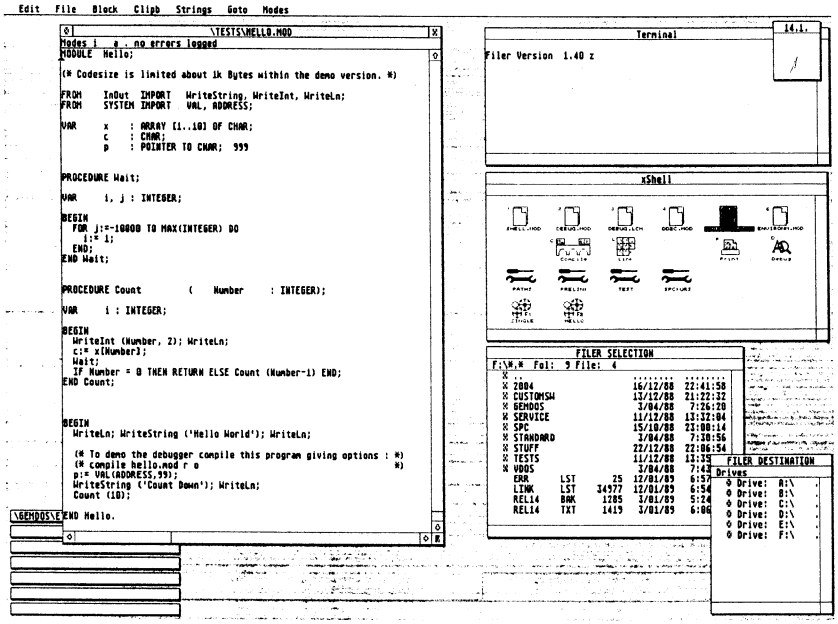
Inhaltsverzeichnis

4 Der Editor

5 Der Compiler

6 Der Debugger

7 Die Utilities


Edit File Block Clipb Strings Goto Modes
\TESTS\HELLO.MOD
Modes i a . no errors logged
MODULE Hello;
(* Codesize is limited about 1k Bytes within the demo version. *)
FROM InOut IMPORT WriteString, WriteInt, WriteLn;
FROM SYSTEM IMPORT VAL, ADDRESS;
VAR x : ARRAY [1..10] OF CHAR;
c : CHAR;
p : POINTER TO CHAR; 999
PROCEDURE Wait;
VAR i, j : INTEGER;
BEGIN
FOR j:=-10000 TO MAX(INTEGER) DO
i:= 1;
END;
END Wait;
PROCEDURE Count (Number : INTEGER);
VAR i : INTEGER;
BEGIN
WriteInt (Number, 2); WriteLn;
c:= x[Number];
Wait;
IF Number = 0 THEN RETURN ELSE Count (Number-1) END;
END Count;
BEGIN
WriteLn; WriteString ('Hello World'); WriteLn;
(* To demo the debugger compile this program giving options : *)
(* compile hello.mod r o *)
p:= VAL(ADDRESS,99);
WriteString ('Count Down'); WriteLn;
Count (10);
\GEMDOS\E
END Hello.
Terminal
Filer Version 1.40 z
xShell
FILER SELECTION
F:*.* Fol: 9 File: 4
2004 16/12/88 22:41:58
CUSTOMSW 13/12/88 21:22:32
GEMDOS 3/04/88 7:26:20
SERVICE 11/12/88 13:32:04
SPC 15/10/88 23:00:14
STANDARD 3/04/88 7:30:56
STUFF 22/12/88 22:06:54
TESTS 11/12/88 13:35
VDOS 3/04/88 7:43
ERR LST 25 12/01/89 6:57
LINK LST 34977 12/01/89 6:54
REL14 BAK 1285 1/01/89 5:24
REL14 TXT 1415 3/01/89 6:06
FILER DESTINATION
Drives
0 Drive: A:\
0 Drive: B:\
0 Drive: C:\
0 Drive: D:\
0 Drive: E:\
0 Drive: F:\

Vorwort

Das Produkt

SPC MODULA–2 ist ein komplettes Sprachsystem für die Entwicklung von MODULA–2 Programmen auf dem ATARI ST oder dem MEGA ST. Der Compiler des Sprachsystems wurde von N. Wirth selbst und seinem Team an der Eidgenössischen Technischen Hochschule (ETH) Zürich entwickelt.

Lieferumfang

Das Sprachsystem beeinhaltet u.a.:

- eine grafische Shell
- einen schnellen Compiler
- einen sprachsensitiven Editor
- einen source–level Debugger
- einen dynamischen linkenden Lader
- einen Linker
- eine File–Utility
- ein deutsches Handbuch
- die MODULA–2 Standardbibliotheken
- alle GEM–Bibliotheken

Entwicklungsziele

Die Entwicklung von SPC MODULA–2 war und ist von dem Ziel getragen, dem Entwickler ein effizientes und schnelles Werkzeug bereitzustellen, so daß dieser seine wertvolle Zeit weitestgehend auf die kreativen Phasen des Programmierens verwenden kann.

Entwicklungskomfort

Dies wurde vor allem dadurch erreicht, daß auf das meist zeitaufwendige Binden der Programme verzichtet werden kann. Einen weiteren Beitrag leistet der Compiler mit einer Übersetzungsgeschwindigkeit von bis zu 5000 Zeilen in der Minute. Der Editor zeigt die vom Compiler entdeckten Fehler direkt im Programmtext an. Falls während des Programmtests ein Laufzeitfehler

entdeckt wird, wird automatisch der Debugger nachgestartet. Dieser zeigt dann die fehlerhafte Stelle – ebenfalls im Quelltext– sowie die Inhalte von Variablen zur Zeit des Absturzes. Ein übriges tut die xShell zum Komfort beim Programmieren. Sie zeigt in grafischer Weise die momentane "Umgebung". Darunter wird die Menge von Dateien und Werkzeugen verstanden, die der Entwickler gerade am häufigsten benötigt. Die meisten Kommandos können mit einem oder zwei Maus–Klicks gegeben werden oder indem einfach nur die SPACE–Taste gedrückt wird.

Das Handbuch

Um das Sprachsystem abzurunden wurde ein Handbuch erstellt, bei dem Übersichtlichkeit eines der wichtigsten Kriterien war. Gleichzeitig wurde versucht, alle für den Entwickler wichtigen Informationen kompakt darzustellen, um die Menge des zu bewältigenden Materials gering zu halten.

Umfang des Handbuchs

Das Handbuch beschreibt die Komponenten des SPC MODULA–2 Sprachsystems auf dem ATARI ST unter GEM. Hierzu gehören neben den Sprachwerkzeugen inbesondere die mitgelieferten Bibliotheken. Sie werden im Detail durch ihre Definitionsmoduln beschrieben. Neben dem SPC MODULA–2 Handbuch sollten Sie über weitere Literatur verfügen, die die Bedienung Ihres Computers beschreibt. Die Beschreibung der Programmiersprache MODULA–2 ist ebenfalls nicht Teil des Handbuchs. Jeder MODULA–2 Programmierer benötigt als Nachschlagewerk das Buch von N. Wirth "Programming in MODULA–2", das auch in der deutschen Übersetzung vorliegt.

Organisation des Handbuchs

Das Handbuch ist in Kapitel und Anhänge aufgeteilt. Die Gliederung orientiert sich an den Komponenten des Sprachsystems. Die Beschreibung einer Komponente erfolgt so, daß zuerst beschrieben wird, **was** die Komponente leistet. Ein zweiter Teil befaßt sich damit, **wie** man die Leistung in Anspruch nimmt. Ein

dritter, optionaler Teil beschreibt schließlich, wie eine Komponente intern funktioniert, um z.B. Restriktionen oder Querbezüge aufzuzeigen.

notationelle Konventionen

Auf jeder Handbuchseite finden sich am äußeren Rand Stichwörter, die auch im Index aufgelistet sind. Namen von Tasten der Tastatur werden [Quelle] gesetzt. Betonungen werden **fett** gedruckt. Dateinamen werden GROSS geschrieben.

☞ In dieser Weise werden wichtige Hinweise hervorgehoben. Lesen Sie diese unbedingt, bevor sie Bedienungen vornehmen.

bevor Sie anfangen

Bevor Sie sich nun an die Arbeit machen und Ihre eigenen MODULA-2 Programme entwickeln, sollten Sie sich die Zeit nehmen, das Handbuch zu lesen. Auch wenn Sie bereits ein erfahrener MODULA-2 Programmierer sind, ist es wichtig, daß Sie sich einen Überblick über die Punkte verschaffen, die durch die Dokumentation abgedeckt werden. Außerdem werden an vielen Stellen Hinweise über die richtige und sinnvolle Benutzung des Sprachsystems gegeben.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Installation und Inbetriebnahme

Kapitel 1

Lieferumfang

Bevor Sie mit der Installation beginnen, überprüfen Sie bitte die Vollständigkeit Ihrer Lieferung und ob die gerätemäßigen Voraussetzungen für eine Installation gegeben sind. Die Lieferung besteht aus folgenden Teilen:

- 1 Handbuch im DIN A5 Ordner
- 3 doppelseitig beschriebene 3.5" Disketten
- 1 Bogen Aufkleber für Funktionstasten
- 1 Software–Registrierkarte
- 1 Software–Lizenzvertrag

Vorbereitungen

Für die Installation benötigen Sie als Mindestausstattung:

- 512 kByte freien Hauptspeichers
- 720 kByte Massenspeicher
- monochromen Monitor

Sie können SPC–MODULA–2 auch auf einem MEGA ST mit Blitter–TOS betreiben. Eine Festplatte ist für ein produktives Arbeiten sinnvoll. Bitte stellen Sie aber in jedem Fall sicher, daß 512 kByte im Arbeitsspeicher frei sind.

Lizenzvereinbarung

Die Diskette ist versiegelt. Bitte beachten Sie, daß Sie mit dem Öffnen der Diskette den Lizenzvertrag akzeptieren. Sie verpflichten sich darin ausdrücklich, dieses Exemplar von SPC–MODULA–2 zu einer Zeit nur auf einer CPU einzusetzen. Weiter stellen Sie sicher, daß Ihr Exemplar nicht außerhalb der Lizenzbestimmungen

eingesetzt wird, und benachrichtigen den Vertreiber, falls Sie dennoch davon Kenntnis erhalten sollten.

Registrierung

Füllen Sie die Registrierkarte aus und schicken Sie sie an den Vertreiber zurück. Sie werden damit in die Liste der registrierten Anwender aufgenommen und werden automatisch über neue Versionen informiert. Während der Garantiezeit von 6 Monaten erhalten Sie Updates gegen einen Unkostenbeitrag für Porto und Verpackung. Sollte Ihre Version schon zum Kaufdatum nicht mehr aktuell sein, dann erhalten Sie nach Eingang der Registrierkarte eine aktuelle Version.

Service Hotline

Der Hersteller unterhält eine Mailbox, über die der Service von SPC MODULA-2 abgewickelt wird. Die Mailbox kann mit einem 300 oder 1200 Baud Modem (Akustikkoppler) erreicht werden. Die Modem-Parameter sind:

☞ 0721 / 700963, 300/1200bd, 8 Datenbits, keine Parität

Serialisierung

Ihr Exemplar von SPC MODULA-2 ist serialisiert, d.h. daß die Software mit einer Seriennummer versehen ist. Die Seriennummer ist in einer sogenannten Environment-Variablen gespeichert. Um zu verhindern, daß die Seriennummer entfernt wird, wurde sie um einen Seriencode ergänzt. Seriennummer und SerienCode stehen in einem bestimmten Zusammenhang und werden von Zeit zu Zeit durch SPC MODULA-2 abgefragt. Sie sollten deshalb nicht versuchen, die Nummer zu verändern oder zu entfernen. Ihre eigenen Programme sind von Seriennummer und Seriencode selbstverständlich unabhängig. Seriennummer und Seriencode sind auf Ihrer Originaldiskette und auf dem Software-Lizenzvertrag vermerkt. Bei jeder Korrespondenz mit dem Vertreiber sollten Sie Seriennummer und Seriencode angeben.

Updates

☞ Updates werden immer ohne Seriennummer ausgeliefert. Sie müssen deshalb selbst dafür Sorge

tragen, daß nach dem Einspielen eines Updates das Profile wieder die Seriennummer enthält.

Installation

Die folgende Installationsprozedur wird von einem SPC MODULA-2 Programm gesteuert, das sich auf der Release-Diskette befindet. Das Programm will wissen, auf welchem Laufwerk Sie SPC MODULA-2 installieren wollen. Falls Sie SPC MODULA-2 ohne Harddisk betreiben wollen, dann müssen Sie jetzt zunächst drei Disketten doppelseitig formatieren. Falls Sie SPC MODULA-2 auf einer Harddisk installieren möchten, dann müssen Sie eine Partition auswählen, auf der noch ca. 1MByte Speicher frei ist.

INSTALL.OBM

Gehen Sie nun wie folgt vor:

- Legen Sie die Diskette mit der Beschriftung "Disk 1 of 3" in Laufwerk A:
- Öffnen Sie den Ordner \SPC\USER
- Starten Sie das Programm XSHELL.PRG
- Kilcken Sie auf das Icon mit der Beschriftung "Install"
- folgen Sie den Anweisungen des Programms bis zu der Meldung, daß die Installation beendet ist.

☞ Verstauen Sie die Original-Disketten an einem sicheren Ort.

README.1ST

Lesen Sie nun zunächst die Datei \SPC\README.1ST. Sie enthält letzte Informationen, die nicht im Handbuch enthalten sind.

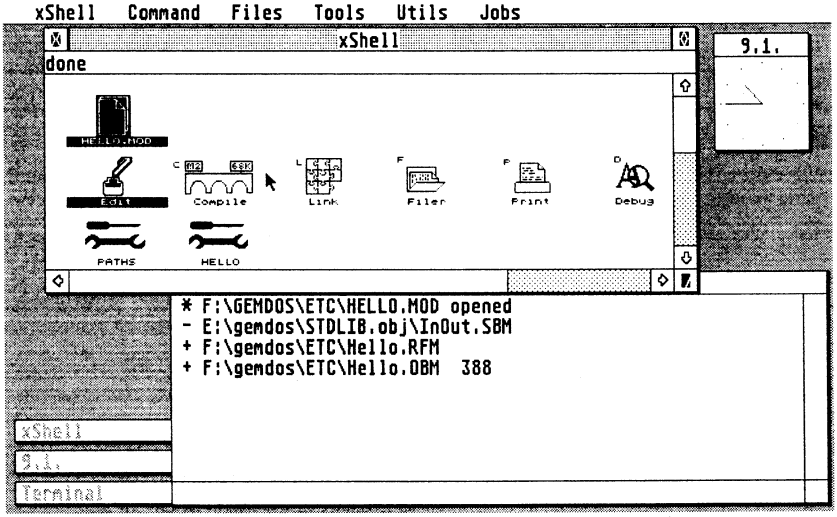
Prüfen der Installation

SPC MODULA-2 ist nun installiert. Die folgenden Schritte sollen sicherstellen, daß die Installation erfolgreich war. Dazu werden Sie die wesentlichen Komponenten des Systems kurz kennen lernen. Öffnen Sie nun das Laufwerk oder die Partition, auf der SPC MODULA-2 installiert wurde. Der Ordner \SPC enthält den ganzen Release. Um neue Versionen ohne Pro-

bleme einspielen zu können, sollten Sie keine eigenen Dateien in dem Ordner \SPC anlegen. Im Ordner \SPC befinden sich mehrere weitere Ordner, von denen im Moment der Ordner \SPC\USER von Interesse ist. Öffnen Sie diesen Ordner.

SPC MODULA–2 starten

Sie finden in dem Ordner das Programm XSHELL.PRG. Starten Sie dieses Programm. Der Ladevorgang dauert, wenn Sie mit Diskettenlaufwerken arbeiten, etwas über eine halbe Minute. Danach wird der Desktop initialisiert und zwei Fenster mit de Titeln "Terminal" und "xShell" geöffnet. Wenn Sie schon etwas vertrauter mit MODULA–2 sind, werden Sie wissen, daß Terminal (bzw. InOut) die Standardein–/–ausgabe auf den Bildschirm, bzw. von der Tastatur abwickelt. Die xShell ist ein Programm, von dem aus Sie alle Werkzeuge und Utilities des Systems komfortabel starten können. Die Bedienung der Shell ist sehr einfach und in Kapitel 3 dieses Handbuchs ausführlich erläutert. Für den



Moment verfahren Sie einfach so wie unten beschrieben. Wir werden versuchen, das Programm HELLO.MOD zu übersetzen und zu starten. Dabei lernen Sie neben der xShell den Compiler, den Editor und den Debugger kennen.

HELLO.MOD

Sie sehen im Fenster der xShell ein Icon mit der Beschriftung "HELLO.MOD". Klicken Sie das Icon mit der linken Maus-Taste. Es wird daraufhin invertiert dargestellt. In der zweiten Icon-Reihe sehen Sie als zweites von links das Compiler-Icon. Klicken Sie auch dieses mit der linken Maus-Taste.

übersetzen

Es wird jetzt der Compiler geladen. Der Fortgang Übersetzung wird im Terminal-Fenster protokolliert. Die Übersetzung wird nicht erfolgreich sein. Das ist normal, denn die Datei enthält, so wie sie geliefert wird, einen Fehler, den Sie als nächstes beheben sollen.

editieren

Inzwischen ist das Editor-Icon (äußerst linkes Icon der zweiten Reihe) invers dargestellt. Dadurch zeigt die xShell an, welches Kommando als nächstes vorgeschlagen wird. Gleichzeitig ist immer noch das Icon der Datei HELOO.MOD selektiert. Wenn Sie nun die Lerrtaste (SPACE) drücken, akzeptieren Sie das Default-Kommando und der Editor wird gestartet.

Der Editor öffnet sofort die Datei HELLO.MOD und stellt den Cursor auf die Stelle, an der der Fehler vom Compiler erkannt wurde. Die Zahl 999 hinter dem Semikolon ist offensichtlich fehlerhaft. Entfernen Sie diese indem sie 3 mal BACKSPACE drücken. Verlassen Sie nun den Editor indem Sie die Taste F10 drücken. Die Datei wird zurückgeschrieben.

Starten Sie erneut den Compiler, indem Sie die Taste SPACE drücken, und damit das Default-Kommando akzeptieren. So einfach ist das! Die Übersetzung wird diesmal erfolgreich sein, was daran zu erkennen ist,

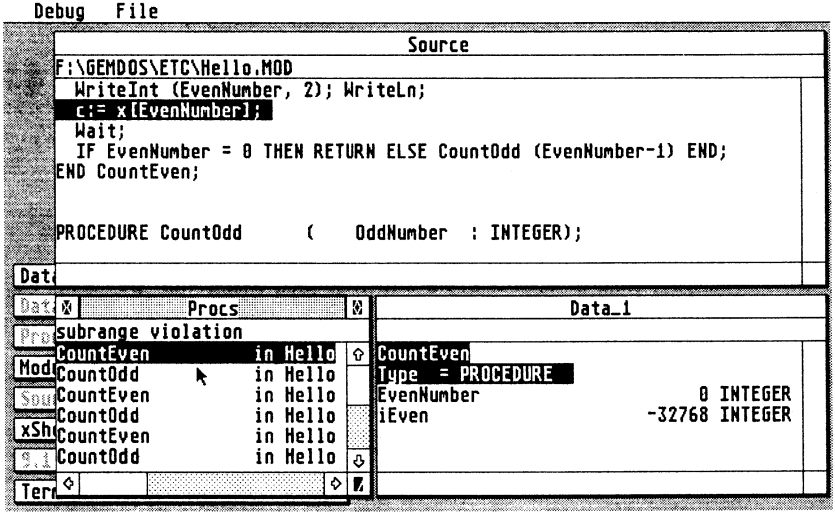
daß der Compiler nicht mehr die Meldung 'Errors Detected' ausgibt, sondern eine Zahl.

HELLO.MOD starten

Nun wollen wir das Programm testen. Das Programm wurde der Einfachheit halber schon als sogenannte Utility installiert. Es ist deshalb auch als Icon in der dritten Icon–Reihe vertreten. Zum Starten brauchen Sie nur dieses Icon anzuklicken. Leider enthält das Programm auch einen Laufzeitfehler, so daß es abstürzen wird. Die xShell wird dann automatisch den Debugger starten.

debuggen

Das Programm wird einen Count–Down von 10 bis 1 ausgeben, und dann einen Laufzeitfehler melden. In der Dialog–Box klicken Sie nun bitte Debug an. Es dauert einen kleinen Moment, bis der Debugger geladen ist. Dieser öffnet 3 Fenster auf dem Desktop. Im oberen Fenster sehen Sie den Quelltext von



HELLO.MOD und eine invers dargestellte Zeile. Sie enthält das fehlerverursachende Statement. Im linken Fenster sehen sie die Prozedur-Aufrufkette und die Fehlerursache, nämlich Index-/Range-Check in diesem Falle. Wenn Sie nochmals den Quelltext betrachten, dann sehen Sie, daß der Fehler bei einem Zugriff auf ein Array x aufgetreten ist. Der Index dabei war i. Die Variablen und ihre Werte sehen Sie im rechten Fenster. Der Inhalt von i ist 0. Der Grund für einen Fehler liegt darin, daß das Array x mit einem Indexbereich von 1 bis 10 erklärt wurde und 0 deshalb unzulässig ist.

Wenn Sie nun -wie allgmein üblich- das File-Menü anwählen und darin den Eintrag Quit selektieren, dann wird der Debugger verlassen.

Die xShell wird wieder die Dialog-Box zeigen, um zu erfahren, was sie nun weiter mit dem fehlerhaften Programm tun soll. Klicken Sie auf Abort um das Programm abzubrechen.

xShell verlassen

Verlassen Sie anschließend die xShell, indem Sie die Taste Q drücken. Die xShell bietet noch ein letztes Formular an, in dem Sie Quit wählen. Daraufhin wird die xShell endgültig verlassen.

☞ Sollten Sie nicht wie beschrieben bis hierher gekommen sein, dann ist ihre Installation nicht gelungen.

fehlerhafte Installation

Eventuell sind beim Einspielen der Dateien Fehler aufgetreten. Möglicherweise haben Sie nicht genügend freien Speicher. Booten Sie ihren Rechner noch einmal ohne Desktop-Accessories und versuchen Sie es noch einmal. Es kann natürlich auch sein, daß ihre Disketten defekt sind. In diesem Fall müssen Sie sich an den Händler oder direkt an die Firma Advanced Applications Viczena GmbH wenden.

SPC MODULA-2 benutzen

Sie haben die ganze Zeit in dem Ordner \SPC\USER gearbeitet. Das war eine Ausnahme.

☞ Normalerweise sollten Sie keine eigenen Dateien unter \SPC anlegen, sondern sich neue Ordner neben \SPC schaffen, in denen Sie Ihre Projekte organisieren.

eigene Projekte

weitere Ordner anlegen

Diesen Ordnern können Sie beliebige Namen geben (nicht unbedingt USER). Für den Anfang kopieren Sie bitte den Ordner \SPC\USER auf das gleiche Laufwerk, auf dem \SPC liegt, sodaß Sie dort mindestens die beiden Ordner \SPC und \USER vorfinden. Öffnen Sie den Ordner \USER und starten Sie noch einmal XSHELL.PRG. Es sollte sich wieder die xShell melden. Verlassen Sie die xShell wieder wie oben erläutert.

XSHELL.PRG als Anwendung anmelden

Wenn Sie schon erfahren im Umgang mit GEM sind, können Sie nun XSHELL.PRG als Anwendung für Dateien mit der Endung .CNF anmelden.

☞ Falls Sie noch nicht genügend mit GEM vertraut sind, dann überspringen Sie diesen Abschnitt. Sie können diesen Teil jederzeit auch später durchführen.

DESKTOP.INF hacken

Sichern Sie die Arbeit. Starten Sie XSHELL.PRG, drücken Sie die Taste F3 zum Editieren und wählen Sie in der Dateiauswahl-Box die Datei DESKTOP.INF auf ihrem Boot-Laufwerk aus. Bewegen Sie den Cursor nach unten bis zu der Zeile, in der XSHELL.PRG steht. Bewegen Sie den Cursor nach rechts bis auf das Xvon XSHELL und fügen Sie \SPC\USER\ ein. Bitte beachten Sie, daß der Editor das Zeichen \ ohne Shift auf die [\] -Taste auflegt. Tragen Sie vor \SPC\USER\XSHELL.PRG den Namen des Laufwerks ein, auf dem der Ordner \SPC abliegt, also z.B.: A:, B: oder eine der Harddisk-Partitions.

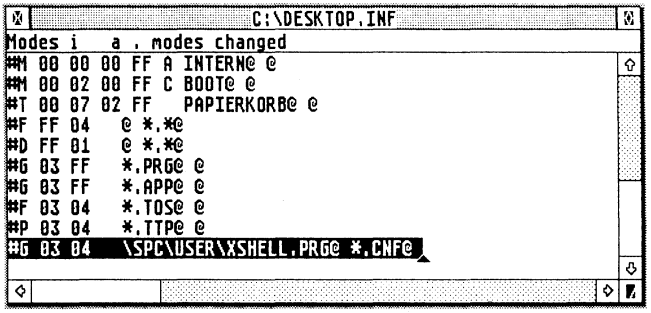
Wenn Sie absolut sicher sind, daß Sie alles richtig gemacht haben, dann verlassen Sie den Editor mit F10.

Don't Panic!

☞ Falls Sie Zweifel haben, verlassen Sie den Editor mit Shift F10 ohne die Datei zurückzuschreiben. In diesem Fall kommen Sie bitte zu einem späteren Zeitpunkt auf diesen Abschnitt zurück. Überspringen Sie den Rest des Abschnittes.

Anmeldung war erfolgreich

Falls Sie DESKTOP.INF erfolgreich gehackt haben, booten Sie Ihr System und öffnen Sie wieder den Ordner \USER. Löschen Sie die Datei XSHELL.PRG und öffnen Sie (durch Doppelclick) PROFILE.CNF. In der obersten Zeile ihres Bildschirmes sollte nun der Name \SPC\USER\XSHELL.PRG erscheinen. Nach einiger Zeit muß sich die xShell melden. Sie haben nun XSHELL.PRG so montiert, daß sie die Datei nur noch einmal auf ihrer Diskette halten müssen. In allen weiteren Ordnern, in denen Sie eigenständige Projekte abwickeln wollen, brauchen Sie nur noch die Datei PROFILE.CNF. Die Bedeutung dieser Datei lernen Sie in einem späteren Kapitel.



weitere Schritte

ein Arbeits-Directory einrichten

Richten Sie nun einen Ordner außerhalb des Ordners \SPC ein, auf dem Sie ihre ersten Programme entwickeln. Nennen Sie den Ordner vorläufig einfach \WORK. Sie können später weitere Ordner nach dem gleichen Muster anlegen, um ihre Arbeiten zu strukturieren. Kopieren Sie aus dem Ordner \SPC\USER die Datei PROFILE.CNF. Falls Sie XSHELL.PRG wie oben beschrieben als Anwendung montiert haben, sind Sie nun fertig. Andernfalls müssen Sie noch XSHELL.PRG auf den Ordner \WORK kopieren.

SPC MODULA-2 starten

Falls XSHELL.PRG angemeldet ist, können Sie SPC-MODULA-2 nun durch Doppel-Klicken von PROFILE.CNF starten. Wenn Sie XSHELL.PRG nicht angemeldet haben, dann starten Sie SPC MODULA-2 vorerst durch Klicken von .PRG. Sie können dann später die Anmeldung durchführen, um nicht in jedem Ihrer Ordner immer SHELL.PRG halten zu müssen.

Einführung in SPC MODULA-2

Kapitel 2

Übersicht

Das folgende Kapitel erläutert die besonderen Merkmale von MODULA-2 als Programmiersprache sowie die von SPC MODULA-2 als Sprachsystem.

Unterschiede gegenüber PASCAL

Die Programmiersprache MODULA-2 ist der Nachfolger von PASCAL. Beide Sprachen wurden von N.Wirth an der ETH Zürich entwickelt. Gegenüber PASCAL zeichnet sich MODULA-2 durch Verbesserungen aus, die 4 Kategorien zugeordnet werden können:

- Modulkonzept
- maschinen- bzw. systemnahe Elemente
- Prozedurtyp
- syntaktische Überarbeitungen

Unterschiede von MODULA-2 Implementierungen

Die Sprachdefinition von MODULA-2 wird derzeit durch Wirth's Buch "Programming in MODULA-2" gegeben. Allerdings bleiben einige nicht unwesentliche Details ungeklärt, so daß sich verschiedene Implementierungen der Sprache durchaus unterscheiden können. Da größten Unterschiede ergeben sich jedoch durch den Umfang und die Schnittstellen der zum Sprachsystem gehörenden Standard-Bibliotheken.

Normung von MODULA-2

Die international Standardisierungs-Organisation ISO befaßt sich zur Zeit mit der Normung von MODULA-2. Es wird damit gerechnet, daß die Norm Ende 1989 vorliegt. SPC MODULA-2 wird dann natürlich der genormten Sprachdefinition entsprechen.

Modul-Konzept

Das Modulkonzept ist die wichtigste Verbesserung von MODULA-2 gegenüber seinem Vorgänger PASCAL. Erst durch ein Modulkonzept wird eine Sprache für große Softwareprojekte brauchbar. Das Modulkonzept ermöglicht es nämlich, Schnittstellen zwischen verschiedenen Systemteilen festzuschreiben, und ihre Einhaltung durch den Compiler bzw. den Lader prüfen zu lassen. Dadurch können mehrere Entwickler an einem großen System arbeiten. Jeder Entwickler erstellt und testet einen, durch klare Schnittstellen beschriebenen Modul (oder mehrere). Die Herstellung eines Moduls bezeichnet man als seine Implementierung - im Gegensatz zu seiner Schnittstelle, seiner Definition.

Schnittstelle und Implementierung

Wie ein Modul implementiert ist, d.h. wie er die von der Schnittstelle geforderte Leistung erbringt, interessiert i.a. außerhalb des Moduls nicht, da der Modul durch seine Schnittstelle ausreichend beschrieben ist. Es liegt also nahe, die Beschreibung der Schnittstelle von der Implementierung zu trennen. Die Schnittstelle bezeichnet man in MODULA-2 als einen DEFINITION MODULE. Die Implementierung heißt IMPLEMENTATION MODULE. Beide Teile werden getrennt übersetzt. Der DEFINITION MODULE kann insbesondere schon übersetzt werden, ohne daß die Implementierung steht. Mehrere Entwickler können also mit ihrer Arbeit beginnen, sobald die Schnittstellen der zu realisierenden Moduln definiert sind. In großen Projekten ist dies ein wichtiger Faktor.

Reduktion der Programmkomplexität

Aber auch, wenn man alleine an einem Softwaresystem arbeitet, ist die Modularisierung wichtig. Die Schnittstelle eines Moduls ist nämlich wesentlich weniger komplex als seine Implementierung. Andererseits beinhaltet sie alles wesentliche eines Moduls, nämlich seine Leistungsbeschreibung. Ein größeres System wird deshalb leichter durchschaubar und besser wartbar. Da alle Teile einzeln übersetzt werden können,

sind alle Ladezeiten kürzer, der Editierzyklus also kürzer.

wiederverwendbare Softwarebausteine

Last not least ist natürlich anzumerken, daß bei geeigneter Modularisierung Softwarebausteine (oder auch Software-Chips) entstehen, die in anderen Projekten wieder verwendet werden können. Der Festlegung von Modulschnittstellen kommt deshalb besondere Bedeutung zu, und Sie sollten sich niemals scheuen, eine für ungünstig befundene Schnittstelle zu überarbeiten, auch wenn dies mit Arbeit verbunden ist, die sich für den Moment nicht auszuzahlen scheint.

Standardbibliothek

Beispiele für wiederverwendbare Softwarebausteine finden sich in der MODULA-2 Standardbibliothek. Diese Moduln werden von den Werkzeugen des Sprachsystems verwendet, stehen aber auch den Anwendungsprogrammen zur Verfügung. Die Standardbibliothek hat ihren Namen daher, daß sie Dienste bereitstellt, die standardmäßig in allen MODULA-2 Implementierungen angeboten werden sollen. Programme, die nur solche Dienste verwenden, sind dann offensichtlich portabel, d.h. sie können leicht von einer MODULA-2 Implementierung auf eine andere übertragen werden. Für fast jeden professionellen Softwareentwickler ist dies eine wichtige Eigenschaft, denn ein Programm, das auf vielen Systemen angeboten werden kann, ist bestimmt mehr wert, als eines, das nur für eine einzige Maschine zu haben ist.

systemabhängige Moduln

Programme, die nicht portabel sind, nennt man systemabhängig, denn sie verwenden Dienste, die nur auf dem betreffenden System zur Verfügung stehen. Solche Dienste werden von den Moduln der Systembibliothek zur Verfügung gestellt. Auf dem ATARI ST z.B. enthält sie die Moduln zur direkten Ansprache des GEMDOS, des AES und des VDI. Programme, die diese Moduln verwenden, also auf ihre Schnittstellen Bezug nehmen, sind nicht ohne weiteres auf andere Rechner übertragbar. Da ein Programm i.a. in mehrere Moduln zerfällt, muß man derartige Betrachtungen

Systemabhängigkeiten isolieren

eigentlich auf die einzelnen Moduln anwenden. Man muß deshalb auch nicht gleich das Kind mit dem Bade ausschütten, wenn man einen systemabhängigen Modul verwenden will. Vielmehr wird man versuchen, die Systemabhängigkeiten in einen oder wenige Moduln zu konzentrieren, sodaß im Falle eines Falles nur diese wenigen Moduln anzupassen sind.

leider kein Bibliotheksstandard

Wenn Sie nun die Dienste der Standardbibliothek betrachten, dann werden Sie feststellen, daß diese bei verschiedenen MODULA-2 Implementierungen geringfügige Unterschiede aufweisen. Das liegt daran, daß MODULA-2 momentan noch nicht standardisiert ist. Erst wenn der MODULA-2 Standard zumindest als endgültiger Normvorschlag vorliegt, kann man davon ausgehen, daß alle MODULA-2 Implementierungen die gleichen Standard-Bibliotheken bereitstellen.

systemnahe Elemente

Systemabhängige Programme sind nun nicht zwangsläufig zweitklassige Programme. Vielmehr muß es auch systemabhängige Programme geben, man denke nur an Betriebssysteme, grafische Subsysteme, etc. Sie sollten nur immer darauf achten, nicht ein ganzes großes Programm durchweg systemabhängig zu machen. Wenn Sie nun aber systemabhängige Moduln schreiben, dann werden Sie Elemente zur maschinennahen Programmierung benötigen. Dazu gehört z.B. der Zugriff auf Registerinhalte, der direkte Zugriff auf bestimmte Speicherstellen und überhaupt das Vorhandensein von Datentypen wie Adressen, Worten, Bytes. Diese Elemente haben ebenso wie das Modulkonzept in PASCAL vollkommen gefehlt, denn PASCAL war ursprünglich für die Ausbildung von Studenten der Informatik entwickelt worden. Für Ausbildungsaufgaben hatte man solche maschinennahen Elemente nicht für nötig befunden. MODULA-2 trägt auch hierin modernen Anforderungen Rechnung und rundet die

Leistungsfähigkeit der Sprache nach unten ab (Anm.: unten ist bei einem Softwaresystem immer da, wo die Maschine ist, oben ist da wo die Anwendung ist).

Prozedurtyp

Die Einführung eines Prozedurtyps erlaubt es, Prozedurvariablen zu vereinbaren und Prozeduren an sie zuzuweisen. Die Prozedurvariablen können über Parameterschnittstellen transportiert und auch sonst wir alle anderen Variablen behandelt werden. Mit Prozedurvariablen sind Konstruktionen realisierbar, die in PASCAL oder anderen Sprachen sehr viel umständlicher zu programmieren wären.

syntaktische Straffungen

Abgesehen von dem Modulkonzept, dem Prozedurtyp und den maschinennahen Elementen unterscheidet sich MODULA–2 von PASCAL noch durch einige syntaktische Änderungen, die hauptsächlich auf Straffungen der PASCAL–Syntax hinauslaufen. Die Kompaktheit von PASCAL ist dabei erhalten geblieben. Kompaktheit einer Programmiersprache heißt, daß sie ihre ganze Leistungsfähigkeit über einige wenige Konzepte bereitstellt. Das hat den Vorteil, daß Compiler für die Sprache leichter zu implementieren sind, und sich die Sprache deshalb schnell verbreiten kann. Ein weiterer Aspekt der Kompaktheit ist, daß die Sprache leichter zu erlernen ist. Wie schon PASCAL hat N.Wirth auch die Sprache MODULA–2 in einem kleinen Buch beschrieben ("Programming in MODULA–2, Springer). Die formale Sprachbeschreibung umfaßt darin nur 60 Seiten, weitere 140 Seiten sind einer weniger formalen Einführung und Beispielen gewidmet.

Kompaktheit

Zielgruppe von MODULA-2

Zusammenfassend läßt sich über die Sprache MODULA-2 sagen:

- jeder PASCAL-Programmierer kann an einem langen Wochenende MODULA-2 erlernen.
- jeder Programmierneuling findet in MODULA-2 eine Sprache, die wegen ihrer Kompaktheit und strukturellen Klarheit gut als Lernsprache geeignet ist. SPC MODULA-2 enthält zudem einen vollständigen Programmierkurs
- gerade für große Programmsysteme ist MODULA-2 gut geeignet, da das Modulkonzept zu besser strukturierten Programmen führt.
- MODULA-2 wird keine Dialekte wie PASCAL nötig haben, da alle Erweiterungen als Modulschnittstellen formuliert werden können.
- Nach der Normung von MODULA-2 wird die Erstellung portabler Programme optimal unterstützt.
- Selbst für systemnahe Programme ist MODULA-2 geeignet, da es über maschinennahe Elemente verfügt.

wie Sie anfangen

Wenn Sie noch keine Programmiererfahrung besitzen, dann werden Sie zunächst kleine Programme schreiben, und keine maschinennahen Sprachelemente verwenden. Sie kommen dann meist mit den Diensten der Standard-Bibliothek aus. In einem späteren Kapitel werden Sie Gelegenheit haben, ein kleines Programm unter Anleitung zu erstellen. Zunächst sollten Sie sich mit den Werkzeugen des Sprachsystems vertraut machen. Dazu sind einige Bemerkungen zum Aufbau des SPC MODULA-2 Systems erforderlich.

Features von SPC MODULA-2

Die folgenden Abschnitte beschreiben den Aufbau und die Besonderheiten des SPC MODULA-2 Sprachsystems gegenüber anderen Implementierungen von MODULA-2.

Aufgaben eines Linkers

Normalerweise müssen Programmteile, die einzeln übersetzt wurden, vor der Ausführung zu einem Gesamtprogramm gebunden werden. Dazu wird ein Binder (engl. Linker) benützt, der als Ausgabe eine vom Betriebssystem ladbare Datei enthält. Der Bindevorgang fügt also Ihre Programmteile mit verschiedenen Bibliotheks-Moduln zusammen. Je nach Größe des Programms kann das Binden durchaus 5-10 Minuten in Anspruch nehmen.

dynamisches Binden

Bei SPC MODULA-2 können Sie das Binden vergessen. SPC MODULA-2 führt ein sogenanntes dynamisches Binden während des Ladens von Moduln aus. Dazu hat es einen speziellen Lader. Dieser stellt die Verbindungen zwischen den Moduln dynamisch (d.h. beim Laden) her, und zwar viel schneller, als das ein normaler Binder kann. Der Lader ist ein normaler Modul, der von jedem anderen Modul benutzt werden kann. Sie können sich also auch leicht selbst eine Shell schreiben, oder in ihren Programmen den Lader benutzen, um nicht immer das ganze Programm im Hauptspeicher halten zu müssen.

der dynamisch bindende Lader

der Linker

Natürlich enthält SPC MODULA-2 auch einen konventionellen Binder, der aus Ihren Programmen eigenständige, unter GEM ausführbare Programme macht. Diesen benutzen Sie aber erst am Ende Ihrer Entwicklung, und zwar ohne daß dadurch Änderungen in Ihrem Programm nötig werden.

den Modulvorrat strukturieren

Da ein Programm normalerweise aus vielen Moduln besteht, muß der Lader bzw. der Linker i.a. mehrere Dateien laden. Die Dateien, werden in verschiedenen Ordnern gesucht. Die Aufteilung aller vorhandenen Moduln in Ordner schafft eine Struktur innerhalb der Moduln. Im Lieferumfang sind die Moduln auf fünf

Ordner verteilt:

- USER in diesem Ordner liegen zwei kleine Testprogramme, die Sie schon zur Installation benützt haben.
- SYSLIB enthält alle die Moduln, deren Schnittstellen systemabhängig sind.
- SPCLIB enthält Moduln, die sie nur bei SPC MODULA–2 finden werden, deren Schnittstellen jedoch nicht systemabhängig sind.
- STDLIB enthält die Moduln der MODULA–2 Standardbibliotheken.
- UTILITY enthält die Hilfsprogramme des Sprachsystems wie Editor, Compiler, Debugger.

die STDLIB

Für Ihre ersten Programme sind die Moduln der Standardbibliothek STDLIB von Bedeutung. Sie enthalten alles, was man braucht, um "konventionelle" Programme zu schreiben. Die Leistungen der Moduln der STDLIB sollen kurz umrissen werden. Eine ausführliche Beschreibung erfolgt an anderer Stelle.

InOut

InOut unterstützt die formattierte Ein– und Ausgabe auf dem Standardein– bzw. –ausgabegerät des Sprachsystems. Dieses ist normalerweise das Terminal. Jedoch können Eingabe und Ausgabe auf andere Geräte (Dateien) umgelegt werden. InOut ersetzt die aus PASCAL bekannten Standarddateien INPUT und OUTPUT.

Terminal

Die elementare Ansprache des interaktiven Terminals wird vom Modul Terminal geleistet. InOut verwendet normalerweise diesen Modul, um Zeichen auf das Terminal auszugeben, oder von diesem (bzw. seiner Tastatur) einzulesen. Im Gegensatz zu InOut bietet Terminal nur rohe Funktionalität und keine Formatierungsfunktionen.

ByteStreams

Ein ByteStream ist ein Strom von Bytes, der vom Programm zu einem Gerät (oder Datei) fließt, oder umgekehrt. Die innere Struktur eines Byte Stream ist nur dem Anwendungsprogramm bekannt. Der Modul der STDLIB stellt –neben Öffnen und Schließen– nur Funktionen zum Lesen und Schreiben von Bytes und Worten bereit. Ein ByteStream kann von und zum Terminal, zum Drucker und von und zu Dateien gehen.

TextStreams

Das Pendant zu ByteStream ist TextStream. Ein TextStream hat eine innere Struktur. Er enthält druckbare Zeichen, die als Strings, Zahlen oder nur Zeichen formatiert sein können. Darüberhinaus enthält ein TextStream Zeilenende–Zeichen, die von TextStreams auf das jeweilige Gerät abgebildet werden. InOut ist über zwei TextStreams (INPUT und OUTPUT) realisiert.

FileSystem

Einen direkteren Durchgriff auf das Dateiensystem erlaubt der Modul FileSystem. Eine Datei von FileSystem ist im Prinzip ein ByteStream. FileSystem enthält aus den Funktionen eines ByteStream solche, um Dateien umzubennen oder zu löschen.

HFS

(Hierarchical File System)

Die Schnittstelle von FileSystem ist von der Directory–Struktur des Betriebs unabhängig, indem der Dateiname ohne weitere Interpretation an das Betriebssystem durchgereicht wird. Viele Programme benötigen jedoch Operationen auf Directories (Ordnern). Sie werden vom Modul Directories bereitgestellt.

Printer

Wie für das Terminal gibt es auch für den Drucker eine elementare Schnittstelle. Die in SPC MODULA–2 angebotene Schnittstelle geht allerdings über das Elementare weit hinaus. Sie können die Druckeranpassung von !stWord benutzen, um Ihren Drucker optimal anzusteuern.

Clock

Die Darstellung von Uhrzeit und Datum ist auf verschiedenen Rechnern unterschiedlich. Clock verdeckt diese Unterschiede und konvertiert die systemabhängige Darstellung in eine systemunabhängige. Der

Modul enthält Funktionen zum Abfragen und Setzen der Systemzeit.

MathLib und LMathLib

MODULA–2 enthält keine eingebauten (intrinsic) Funktionen für die höheren mathematischen Funktionen. Diese werdem vom Modul MathLib bereitgestellt. Die Floating–Point–Arithmetik in SPC MODULA–2 folgt dem IEEE Floatingpoint–Standard. Alle Operatoren werden in Single und als Double Precision bereitgestellt.

Strings

Die Verarbeitung von Zeichenketten ist in MODULA–2 ebenfalls auf einen Modul der Standardbibliothek verlagert worden. Strings in MODULA–2 sind beliebige ARRAY OF CHAR. Das Ende eines Strings wird immer durch ein NUL–Zeichen (0C) angegeben.

NumberConversions

Das formatierten von Zahlen zu Strings und das Dekodieren von Strings zu Zahlen werden durch den Modul NumberConversions bereitgestellt.

RealConversions

Die Formattierung von Gleitkomma–Zahlen wird üblicherweise getrennt von NumberConversions bereitgestellt um nicht jedes Programm mit den Moduln der Floarting–Point–Arithmetik zu belasten..

Storage

Ähnlich wie in PASCAL ist es auch in MODULA–2 möglich, dynamische Datenstrukturen auf einem sogenannten Heap anzulegen. Die Funktionen zum Erzeugen und Löschen solcher Datenstrukturen ist in MODULA–2 auf den Standardmodul Storage ausgelagert worden.

Coroutines

MODULA–2 enthält ein Konzept, welches in eingeschränktem Maße nebenläufige Prozesse zu programmieren erlaubt. Diese Prozesse werden Koroutinen genannt. Gegenüber konkurrierenden Prozessen von Multitask–Betriebssystemen können Koroutinen nicht vom System unterbrochen werden, um eine andere Koroutine fortzuführen (z.B. nach Ablauf einer Zeitscheibe). Koroutinen geben immer von sich aus die Kontrolle an eine andere Koroutine weiter. Das Koroutinenkonzept war zunächst integraler Bestandteil der

Sprache. Es wurde jedoch an der ETH Zürich erkannt, daß Koroutinen auch über eine Modulschnittstelle realisierbar sind, was den Vorteil hat, daß der Compiler weiter entlastet wird. SPC MODULA-2 folgt diesem Trend und stellt die Funktionalität von Koroutinen über den Modul Coroutines bereit.

die SPCLIB

Zwischen den durch den (de facto) MODULA-2 Standard definierten Moduln und den systemabhängigen Moduln gibt es eine weitere Schicht von Moduln, die allerdings in anderen MODULA-2 Implementierungen üblicherweise nicht bereitgestellt werden. Sie sind deshalb SPC-spezifisch und folgerichtig in der SPCLIB untergebracht.

Process

Wer mit Koroutinen nebenläufige Prozesse realisieren will, wird eine Funktionalität benötigen, die über den elementaren Umfang von Coroutines hinausgeht. Dazu gehören z.B. Warteschlangen und Semaphore.

Envionment

SPC MODULA-2 stellt sogenannte Environment-Variablen zur Verfügung. Über sie werden z.B. Optionen gesetzt oder Grundeinstellungen festgelegt. Alle Environment-Variablen werden bei Verlassen des Systems auf eine Datei PROFILE.CNF gerettet und beim nächsten Start wieder geladen.

TextFiles

Die Speicherung und Verwaltung von Textdateien wird oft gebraucht und verursacht gewöhnlich viel Programmier- und Testaufwand. SPC MODULA-2 stellt standardmäßig den Modul TextFiles bereit, der solche Aufgaben übernimmt.

XStr

Die Leistungen der Standardmoduln zur Stringverarbeitung sind mitunter nicht ausreichend. SPC MODULA.2 stellt eine erweiterte String bibliothek (eXtendend Strings) bereit, deren Funktionen flexibler und mächtiger sind als die der Standardmoduln.

SSWiS

Moderne, interaktive Systeme benützen fensterorientierte Bedieneroberflächen, die normalerweise grafisch orientiert sind, und mit der Maus bedient werden. GEM ist nur ein Beispiel. X-Windows ist ein System, das in Zukunft auf UNIX-Systemen anzutreffen sein wird. Programme, die solche Fenstersysteme verwenden, sind i.a. von dem jeweiligen System abhängig und schlecht portierbar. Für aufwendige Utilities ist dies ein schwerer Nachteil. SSWiS steht für Small Systems Windowing Standard und soll eine systemunabhängige Schnittstelle zu einem Fenstersystem bereitstellen, die in Zukunft auch auf anderen Systemen verfügbar sein wird. Den Konzepten und der Programmierung von SSWiS ist ein eigenes Kapitel gewidmet.

CmdLine

Alle Utilities innerhalb des SPC MODULA-2 Sprachsystems benutzen eine einheitlich aufgebaute Kommandozeile zur Parameterübergabe. Die Übergabe und das Interpretieren der Kommandozeile wird vom Modul CmdLine einheitlich unterstützt.

System

Die Implementierung von SPC MODULA-2, insbesondere die Organisation von Moduln im Hauptspeicher, aber auch viele andere Details, sind im Modul System implementiert. Bitte verwechseln Sie nicht System mit dem Pseudo-Modul SYSTEM. Der Zugriff auf diese Strukturen wird teilweise für andere Modul bereitgestellt. Z.B. der Lader oder der Debugger benützen diese Funktionen.

die SYSLIB

Die genannten obengenannten Moduln basieren natürlich auf systemabhängigen Moduln. Z.B. wird Terminal auf ein GEM-Window abgebildet. Alle Ein-Ausgabe-Funktionen werden auf GEMDOS-Funktionen zurückgeführt, etc. Diese systemabhängigen Moduln sind in dem Ordner SYSLIB zusammengefaßt. Sie sollten von diesem Moduln sparsamen Gebrauch machen, da Ihre Programme damit systemabhängig werden. Die Moduln werden hier nicht alle im Einzelnen aufgeführt. Es handelt sich um

alle Bindings verfügbar

- Die Schnittstellen zum GEM-AES
- Die Schnittstellen zum GEM-VDI
- Die Schnittstelle zu den LineA-Funktionen
- Die Schnittstelle zum GEMDOS
- Die Schnittstelle zum BIOS
- Die SChnittstelle zum XBIOS

eigene Bindings erstellen

Es soll darauf hingewiesen werden, daß SPC MODULA-2 eine elegante Möglichkeit enthält, Betriebssystemfunktionen über Traps anzusteuern (sogenannte CODE-Prozeduren). Es ist deshalb überhaupt kein Problem, eigene Bindings -so heißen die Schnittstellen zu den Systemfunktionen- zu erstellen.

Diese Seite wurde aus satztechnischen Gründen frei gelassen.

Die xShell

Kapitel 3

Übersicht

Die Bedienung des SPC MODULA–2 Sprachsystems erfolgt ab Version 1.4 über die xShell. Die xShell ist eine grafische, auf SSWiS basierende Shell. Sie wurde entwickelt, um die Softwareentwicklung mit SPC MODULA–2 optimal zu unterstützen.

Design–Idee

Während der Entwicklung von Software benutzt der Entwickler i.a. mehrere Werkzeuge und Hilfsprogramme, um Dateien verschiedenen Typs in bestimmter Weise zu bearbeiten. Es gibt folglich zwei Kategorien von Objekten, die während der Entwicklung von Interesse sind: Dateien und Werkzeuge.

Dateien, Werkzeuge, Utilities und Jobs

Bei genauerer Betrachtung lassen sich die Werkzeuge noch untergliedern in die primären Sprachwerkzeuge, Hilfsprogramme und Jobs. Alle zusammen werden im folgenden Werkzeuge genannt und es wird nicht weiter unterschieden, um welchen Typ es sich im einzelnen handelt. Dies ist auf der Ebene der Benutzung von Werkzeugen über weite Strecken auch uninteressant.

Pseudo–Multitasking

Die Shell ist ein SPC MODULA–2 Programm, welches direkt unter GEM ablauffähig ist. Alle Werkzeuge sind SPC MODULA–2 Programme, die vom dynamischen Lader hinzugeladen werden. Die Shell verwendet die grafische Oberfläche von SSWiS und bietet damit die Möglichkeit, mehrere Werkzeuge quasiparallel zu betreiben, indem einfach zwischen den zu den Werkzeugen gehörenden Fenstern hin und her gewechselt wird. Für den Benutzer entsteht dadurch der Eindruck eines Multitasking–Systems, was natürlich auf GEMDOS nicht realisierbar ist. Für die Softwareentwicklung bedeutet es jedoch einen großen Fortschritt, wenn mehrere Werkzeuge nebeneinander benutzbar sind, ohne daß man

jeweils immmer erst eine Utility verlassen muß, bevor man die nächste starten kann.

eigene Utilities einbinden

Es ist ohne weiteres möglich, weitere Utilities zu entwickeln, die an diesem Pseudo–Multitasking teilnehmen. Die einzige Bedingung ist, daß die Utilities in SPC MODULA–2 geschrieben sind und SSWiS benützen.

Eigenschaften von SSWiS

SSWiS ist nicht auf die Fensteranzahl von GEM limitiert, sondern verwaltet bis zu 32 Fenster. Da aber SSWiS auf GEM aufsetzt, können maximal 6 Fenster wirklich geöffnet sein, die restlichen werden als Icons auf dem Desktop dargestellt und können jederzeit geöffnet werden. Weitere Einzelheiten über die Bedienung von SSWiS Applikationen werden im Kapitel über SSWiS behandelt.

Einführung

Objekte

Die Objekte der xShell werden in einem Fenster dargestellt. Potentiell hat der Entwickler eine (fast) beliebige Menge von Objekten im Zugriff. Insbesondere die Anzahl der Dateien ist mitunter sehr groß. In der praktischen Arbeit kommt man jedoch mit einer wesentlich geringerern Zahl aus; dafür möchte man auf diese wenigen Dateien und Werkzeuge ohne große Umstände zugreifen können. Es gibt also eine kleine, überschaubare Menge von Objekten, die in der augenblicklichen Entwicklungsphase von besonderer Bedeutung sind.

unmittelbare Umgebung

Diese sammeln sich im Laufe der Arbeit im Fenster der xShell an und verbleiben dort, bis sie durch wichtigere Objekte wieder verdrängt werden. Die Objekte, die sich im xShell Fenster befinden werden im weiteren Verlauf die (Objekte der) unmittelbaren Umgebung genannt.

mittelbare Umgebung

Alle anderen Objekte, die natürlich auch zugreifbar sind, gehören zur mittelbaren Umgebung.

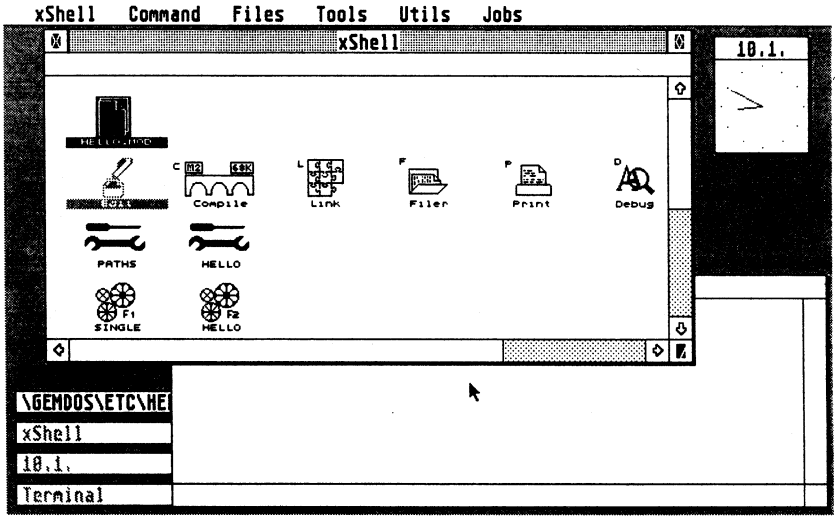
Objekte der mittelbaren Umgebung wechseln in die unmittelbare Umgebung über, sobald sie erstmals ver–

wendet werden. Sie verbleiben in der unmittelbaren Umgebung, bis sie entweder zugunsten anderer Objekte verdrängt werden müssen, oder bis sie vom Benutzer explizit entfernt werden. Selbst wenn die xShell verlassen wird, um zum GEM Desktop zurückzukehren, werden die (Namen der) Objekte der unmittelbaren Umgebung gespeichert. Bei einem neuerlichen Start wird die unmittelbare Umgebung wieder hergestellt.

die unmittelbare Umgebung aufbauen

Wie werden nun die Objekte der unmittelbaren Umgebung benutzt? Die Werkzeuge, Utilities und Jobs werden gestartet, indem man sie mit der Maus anklickt.

Objekte benutzen



Ein gestartetes Werkzeug wird maskiert, da es nicht noch einmal gestartet werden kann.

Werkzeuge benötigen i.a. Argumente, um sinnvolle Arbeit zu verrichten. Normalerweise sind Dateien Gegenstand der Arbeit eines Werkzeuges. Ob ein Werkzeug Dateien als Argumente akzeptiert bzw. benötigt, kann eingestellt werden. Es soll vorerst davon ausgegangen werden, daß Argumente verlangt werden.

Argumente von Werkzeugen

Dateien selektieren

Wenn eine Datei der unmittelbaren Umgebung selektiert ist, dann wird der Name der Datei als Argument an das aufgerufene Werkzeug übergeben. Selektierte Dateien werden invers dargestellt. Eine Datei kann selektiert werden, indem sie mit der Maus angeklickt wird. Es

mehrere Argumente

können auch mehrere Dateien selektiert werden, dann werden sie alle in der Reihenfolge ihrer Selektion an das Werkzeug übergeben. Mehrere Dateien werden selektiert, indem während der Selektion die SHIFT Taste gedrückt wird. Auf diese Weise wird eine sogenannte erweiterte Selektion aufgebaut. Wird eine bereits selektierte Datei mit gedrückter SHIFT Taste angeklickt, dann wird sie aus der Selektion wieder entfernt. Natürlich hängt es von dem aufgerufenen Werkzeug ab, ob es Argumente akzeptiert. Auch der Typ und die Zahl der zulässigen Argumente hängt von dem jeweiligen Werkzeug ab.

wenn keine Datei selektiert ist

Wenn keine Datei der unmittelbaren Umgebung selektiert ist, dann bietet die xShell (vorausgesetzt es werden Argumente verlangt) eine Datei–Auswahl–Box an. Der Benutzer kann dann eine beliebige Datei aus der mittelbaren Umgebung auswählen. Die ausgewählte Datei wird alsdann in die unmittelbare Umgebung geholt und bis auf weiteres im Fenster der xShell dargestellt. Gleichzeitig wird die Datei selektiert und bleibt vorerst selektiert. Die letzte Dateiauswahl wird gespeichert und bei einer erneuten Dateiauswahl wieder als Vorschlagswert verwendet.

standardisierte Kommandozeile

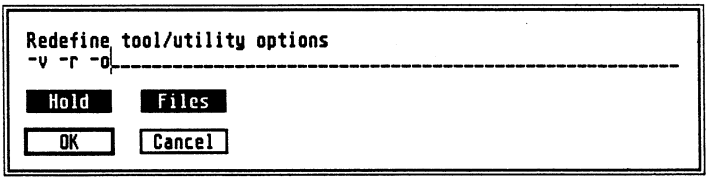
Die Werkzeuge werden mit einer standardisierten Kommandozeile aufgerufen. Sie enthält immer den Namen des Werkzeuges. Darüberhinaus können mehrerere Dateinamen als Argumente vertreten sein. Zuletzt können noch Optionen angegeben werden. Das Format einer Kommandozeile sieht damit so aus:

```
<Kommandozeile>::= <Werkzeugname> {<Argument>} {<Option>}
<Argument>     ::= ' ' <Dateiname>
<Option>       ::= ' ' '-' <Optionsbuchstabe> [<Zeichenkette>]
```

Natürlich bestimmt das Werkzeug selbst, ob und wieviele Argumente und Optionen es akzeptiert, und welche Bedeutung diesen jeweils zukommt. Das Format der jeweiligen Kommandozeile ist deshalb bei den Werkzeugen und Utilities selbst dokumentiert.

Optionen

Die Dateinamen, welche als Argumente übergeben werden, werden von der xShell aus der evtl. erweiterten Selektion bestimmt. Die Optionen werden vom Bediener eingestellt und bleiben i.a. während der Arbeit fest. Dazu wird das Icon, welches ein Werkzeug repräsentiert, bei gedrückter ALTERNATE-Taste durch Doppelklick geöffnet. Es erscheint ein Formular, in dem zunächst eine Zeichenkette angegeben werden kann. Diese Zeichenkette bildet später des Ende der Kommandozeile, d.h. sie enthält normalerweise die letzten Argumente und die Optionen, oder sie ist leer.



Dateiargumente konfigurieren

Weiterhin kann in dem Formular eingestellt werden, ob das Werkzeug Dateinamen als Argumente akzeptiert. Falls die Option gewählt wird, übergibt die xShell auf jeden Fall mindestens einen Dateinamen als Argument.

Werkzeug im Hauptspeicher halten

Eine weitere Option des Formulars erlaubt einzustellen, ob das Werkzeug im Speicher gehalten werden soll. Für häufig benötigte Werkzeuge bietet sich das an, um den Startvorgang zu beschleunigen.

Resultate anzeigen

Die komplette, von xShell gebildete Komamndozeile wird beim Start eines Werkzeuges in der Meldezeile des Fensters eingeblendet. Nach Beendigung des Werkzeuges wird in der Meldezeile ein Ergebnis angezeigt. Falls das Programm nicht gestartet werden konnte, wird eine Fehlermeldung des Laders ausgegeben.

der Modul CmdLine

Der Aufruf von Programmen durch eine Kommandozeile ist unter SPC MODULA-2 Standard. Entsprechend gibt es einen Modul, der die Behandlung der Standard-Kommandozeile unterstützt (CmdLine). Obwohl die Kommandozeile in der xShell praktisch durch Selektieren von Objekten interaktiv aufgebaut wird, handelt es sich um eine Kommandozeile, wie man sie schon von den klassischen, zeilenorientierten Shells her kennt. Für ein Werkzeug ist es vollkommen unerheblich, wie seine Kommandozeile entstanden ist. Sie hätte genauso gut als Text eingelesen werden können. Von dieser Möglichkeit wird Gebrauch gemacht, wenn mit der Batch Funkion, die in die xShell integriert ist, automatisierte Arbeitssequenzen erzeugt werden.

Bedienung

Nach dieser Einführung in die Ideen und Konzepte der xShell wird in den folgenden Abschnitten die Bedienung im Detail erklärt.

Starten

Die xShell ist ein SPC MODULA-2 Programm, das mit dem Linker zu einem unter GEM ablauffähigen Programm gemacht wurde (Namesendung .PRG). Das Programm wird wie üblich vom Desktop aus durch Anklicken oder Öffnen gestartet. Falls keine Fehler auftreten, erscheint nach dem Start ein neuer Desktop mit einer Uhr in der rechten oberen Ecke. Gleichzeitig wird ein Fenster mit dem Titel Terminal eröffnet. Über dieses Fenster wird später die Standard-Ein und Ausgabe von Programmen abgewickelt (Moduln Terminal und InOut). Als nächstes öffnet die xShell ihr eigenes Fenster mit dem Titel xShell. Für jedes geöffnete Fenster wurde am linken unteren Bildschirmrand beginnend ein mit dem Fenstertitel beschrifteter Balken angelegt. Die Funktion dieser Balken wird in der Beschreibung von SSWiS erklärt.

Terminal Fenster

xShell Fenster

Wenn soweit alles fehlerfrei abgelaufen ist, dann zeigt die xShell in ihrem Fenster die unmittelbare Umgebung an, so wie sie zuletzt verlassen wurde. Falls Sie die xShell zum ersten Male starten ist eine Voreinstellung gewählt.

wenn Fehler auftreten

Wenn sich die xShell nicht wie beschrieben starten läßt, kann dies, insbesondere beim erstmaligen Starten, mehrere Ursachen haben:

zu wenig freier Speicher

Es ist nicht genügend Hauptspeicher vorhanden Die xShell wird mit einem 30 Kilobytes großen Stack geliefert, der der eigentlichen Programmgröße noch zugeschlagen werden muß. Außerdem wird sofort ein sogenannter Heap angelegt, der auch nicht kleiner als 20 kBytes sein darf. Falls Sie also vermuten, daß zu wenig Speicher zur Verfügung steht, entfernen sie testhalber geladene RAM–Disks und speicherintensive Accessories. Bedenken Sie auch, was die Programme benötigen, die im AUTO Ordner gestartet werden. Mit 200 kBytes freien Speichers sollten Sie jedoch keine Probleme haben, die xShell zu laden.

das Profile wird nicht gefunden

Die Einstellung der unmittelbaren Umgebung sowie viele andere Parameter werden unter SPC MODULA–2 als sogenannte Environment–Variablen gehalten. Sie werden bei Beendigung der Shell auf ein sogenanntes Profile gerettet und bei einem neuerlichen Start wieder geladen. Dadurch entfällt das lästige Neueinrichten, wenn man zwischenzeitig aus irgendwelchen Gründen zum GEM Desktop zurückkehren mußte. Das Profile mit dem Namen PROFILE.CNF wird im aktuellen Ordner gesucht. Falls es da nicht gefunden wurde, wird es im Wurzelverzeichnis des aktuellen Laufwerks gesucht. Falls das Profile nicht gefunden wurde, können verschiedene Einstellungen nicht vorgenommen werden, und es wird eine Warnmeldung ausgegeben.

das RSC File wird nicht gefunden

Die xShell benützt ein sogenanntes Resource File mit Namen XSHELL.RSC, das die Grafiken enthält. Diese Datei wird in dem Ordner \SPC\MISC auf dem Laufwerk

erwartet, von dem die xShell geladen wurde. Falls die Datei dort nicht gefunden wurde, wird ebenfalls eine Warnmeldung ausgegeben und das Programm terminiert sofort wieder.

Werkzeuge starten

Werkzeuge werden in der xShell durch Anklicken des entsprechenden Icons gestartet. Falls das Werkzeug Datei–Argumente verlangt werden die selektierten Dateinamen übergeben, oder es wird durch eine Datei–Auswahl–Box ein neuer Dateiname erfragt. Dieser wird dann in die unmittelbare Umgebung übernommen. Die Bedienung der Datei–Auswahl–Box ist in der zu Ihrem Computer gehörenden Dokumentation beschrieben. Das

wenn keine Datei selektiert ist

Icon eines gestarteten Werkzeuges wird maskiert, um anzudeuten, daß das Werkzeug nicht noch einmal gestartet werden kann. Zusammen mit den Argumenten werden beim Start eines Werkzeuges seine konfigurierten Optionen übergeben. Die übergebene Kommandozeile wird in der Meldezeile des xShell Fensters angezeigt.

Optionen

Bedienung über Menüs

Werkzeuge, Utilities und Jobs der unmittelbaren Umgebung können auch über die Pull–Down–Menüs (Tools, Utilities, Jobs) gestartet werden. Dazu wird einfach das entsprechende Menü heruntergeklappt und das gewünschte Programm ausgewählt.

Bedienung über Tastatur

Eine letzte Methode gibt es für die Standardwerkzeuge von SPC MODULA–2. Neben dem jeweiligen Icon ist der Anfangsbuchstabe des Programmnamens angegeben. Wenn der Buchstabe über die Tastatur eingegeben wird, wird das dazugehörende Programm gestartet. Dies ist sicher eine sehr effiziente Methode, die besonders beim

Wechsel zwischen Editor, Compiler und Linker zum tragen kommt.

Default–Kommando

Damit nicht genug. Die Vereinfachung für den normalen Entwicklungszyklus geht noch weiter. Die xShell schlägt nämlich immer ein Werkzeug zum Aufruf vor. Das Icon des Default–Werkzeuges ist invers dargestellt. Das Default–Werkzeug kann durch Eingeben des Leerzeichens gestartet werden.

Fehler beim Starten von Werkzeugen

Beim Starten eines Werkzeuges können mehrere Probleme auftreten, die durch eine entsprechende Meldung im xShell Fenster angezeigt werden:

ein Modul kann nicht gefunden werden

Das Programm oder einer der dazu gehörenden Moduln wurde nicht gefunden. Der Lader, den die xShell zum Nachladen von Programmen verwendet, sucht zu ladende Moduln in verschiedenen Ordnern. Der Name der Datei ergibt sich jeweils aus den ersten 8 Buchstaben des Modulnamens. Die Namensendung ist .OBM. Die Namen der Ordner, die der Lader durchsucht, sind in Environment–Variablen gespeichert. Sie werden Ladepfade genannt. Weitere Einzelheiten sind im Abschnitt über den Lader beschrieben. In der Grundeinstellung sucht der Lader in den Ordnern

sind die Ladepfade richtig eingestellt

- .\ aktueller Ordner \SPC\UTILITY\ Ordner mit SPC MODULA–2 Werkzeugen
- \SPC\SYSLIB\ Ordner mit systemabhängigen Moduln
- \SPC\SPCLIB\ Ordner mit systemunabhängigen, SPC–spezifischen Moduln
- \SPC\STDLIB\ Ordner mit Standard–Moduln

inkonsistente Modulschlüssel

Modulschlüssel sind inkonsistent. MODULA–2 Moduln beeinhalten einen sogenannten Modulschlüssel, durch den sichergestellt wird, daß nur solche Moduln miteinander geladen werden, deren Schnittstellen vom Compiler auf Konsistenz überprüft wurden. Stellt der Lader eine Inkonsistenz fest, dann bricht er den Ladevorgang ab und entfernt die bis dahin geladenen Moduln wieder aus dem Speicher. Mit den gelieferten Moduln

sollten solche Fehler nicht auftreten. Bedenken Sie jedoch, daß sie Ihre eigenen Moduln neu übersetzen müssen, wenn sie eine neue Version von SPC MODULA–2 eingespielt haben.

Lesefehler und defekte Dateien

Während des Ladens eines Programmes können natürlich Lesefehler durch defekte Disketten oder Dateien auftreten. In einem solchen Fall sollten Sie der Ursache zunächst auf den Grund gehen, bevor Sie weiterarbeiten. Meist deutet sich dadurch ein bevorstehender Datenverlust an. Lassen Sie es nicht soweit kommen, sondern stellen Sie genau fest, was das Problem ist.

zu wenig freier Speicher

Es ist nicht mehr genügend Speicher vorhanden um das Werkzeug zu laden. Dafür kann es viele Gründe geben. Zunächst kann es sein, daß sich schon zu viele Werkzeuge im Speicher befinden, sei es deshalb, weil sie alle aktiv sind, oder aber weil die Hold–Option gewählt wurde. Wenn Sie die Shell kurz verlassen und wieder neu starten, können Sie feststellen, ob das das Problem war. Möglicherweise steht aber auch aus anderen Gründen nicht mehr genügend Speicher für SPC–MODULA–2 zur Verfügung. Gründe könnten z.B. eine RAM–Disk, Accessories oder Programme im Auto–Ordner sein. Die meisten SPC MODULA–2 Werkzeuge belegen neben ihrem eigenen Code weiteren Speicherplatz (z.B. der Editor für die geladenen Dateien). Auch dies könnte Ursache für zu knappen Speicher sein.

zu viele Programme gestartet

Derzeit können bis zu 15 Programme nebeneinander aktiv sein. Der Lader lehnt es ab, das 16. Programm zu laden.

Fehler im gestarteten Programm

Über die genannten Fehler hinaus können natürlich weitere Fehler innerhalb des gestarteten Programms selbst auftreten, z.B. falsche Argumente oder Optionen, etc. Die möglichen Fehler sind bei den Werkzeugen selbst dokumentiert.

Dateien selektieren

Wenn sich in der unmittebaren Umgebung die momentan benötigten Dateien angesammelt haben, brauchen i.a. nur noch die Icons der Dateien selektiert zu werden, um die Werkzeuge mit Argumenten zu versorgen. Genau wie beim Start von Werkzeugen gibt es auch bei der Selektion von Dateien mehrere Möglichkeiten. Zunächst

können Dateien selektiert werden, indem das zugehörige Icon angeklickt wird. Das momentan selektierte Icon wird dabei deselektiert, so daß im Normalfall immer nur ein Icon selektiert ist. Es besteht jedoch auch die Möglichkeit, mehrere Dateien zu selektieren, wenn beim Anklicken gleichzeitig die SHIFT-Taste gedrückt wird. Ist das mit gedrückter SHIFT-Taste selektierte Icon jedoch gerade selektiert, dann wird es wieder deselektiert.

erweiterte Selektion

Alle Datei-Icons sind von 1 bis 8 durchnumeriert. Durch Eingabe einer Zahl von 1 bis 8 wird die entsprechende Datei selektiert.

mit der Tastatur selektieren

Auch über ein Pull-Down-Menü (Files) können die Dateien selektiert werden. Dazu wird das Menü heruntergeklappt und die gewünschte Datei ausgewählt.

über Menüs selektieren

Dateien werden deselektiert, indem einfach in den leeren Bereich des xShell Fensters geklickt wird. Dateien werden aus der unmittelbaren Umgebung entfernt, indem das Icon bei gedrückter SHIFT-Taste doppelt angeklickt wird.

deselektieren und abmelden

Falls sich schon acht Dateien in der unmittelbaren Umgebung befinden, dann muß eine Datei wieder entfernt werden. Die xShell bevorzugt in diesem Fall die am längsten nicht mehr selektierte Datei. Es kann deshalb sinnvoll sein, ab und zu aufzuräumen und gezielt Dateien aus der unmittelbaren Umgebung zu entfernen.

Utilities

Die wichtigsten Sprachwerkzeuge sind in der zweiten Iconreihe im xShell–Fenster immer präsent, da davon ausgegangen wird, daß sie ständig zugreif–und benutzbar sein müssen. In der dritten Iconreihe werden Hilfsprogramme dargestellt, die im weitesten Sinne auch zu den Werkzeugen gehören, jedoch sind nie alle vorhandenen Hilfsprogramme in der unmittelbaren Umgebung, sondern nur die, die vom Bediener dort installiert wurden. Einmal installiert können sie mit den gleichen Methoden aufgerufen und parameteriert werden, wie die primären Sprachwerkzeuge.

an– und abmelden

Um eine Utility aus der mittelbaren in die unmittelbare Umgebung zu holen muß sie installiert werden. Dazu wird im Commands–Menü der Eintrag Inst Utility gewählt, oder einfach die Taste U gedrückt. Die xShell erfragt dann über eine Datei–Auswahl–Box den Namen der Utility. Die Bedienung der Datei–Auswahl–Box ist in der Dokumentation ihres Rechners beschrieben.

Nachdem ein Dateiname gewählt wurde, wird die Utility installiert und als Icon mit einem großen U im xShell–Fenster dargestellt.

Falls schon acht Utilities installiert sind, wird die am längsten nicht mehr benutzte Utility wieder entfernt. Alle Einstellungen sind damit ebenfalls nicht mehr vorhanden. Es empfiehlt sich deshalb auch hier, ab und zu aufzuräumen.

starten und konfigurieren

Utilities werden, wenn sie einmal installiert sind, wie Werkzeuge behandelt und bedient. Es ist für den Bediener dann auch nicht mehr wichtig, den Unterschied zwischen Werkzeug und Utility zu kennen. Auch Utilities werden mit einer Standard–Kommandozeile parametriert

und die Kommandozeile ergibt sich nach den gleichen Mechanismen wie die von Werkzeugen.

Optionen einstellen

Man beachte, daß nach der Installation einer Utility noch keine Optionen eingestellt sind. Optionen von Utilities können genauso wie die von Werkzeugen eingestellt werden (s.o.), indem das Icon doppelt geklickt wird und das daraufhin erscheinende Formular ausgefüllt wird.

Fehler beim Starten von Utilities

Beim Start von Utilities können natürlich die gleichen Fehler auftreten, die schon oben für Werkzeuge beschrieben wurden. Der weitere Ablauf nach dem Start einer Utility hängt natürlich von der Utility selbst ab. Es ist gute Praxis, bei eigenen Utilities nach den allgemeinen Konventionen die Standard–Kommandozeile auszuwerten.

Jobs

Neben den primären Sprachwerkzeugen und diversen Utilities benutzt der Softwareentwickler i.a. sogenannte Jobs. Jobs automatisieren kleinere Abläufe. Meist wird durch sie der Aufruf verschiedener Utilities gesteuert. Die Abläufe werden in einer Sprache, der Job Control Language (JCL) beschrieben. Unter UNIX ist die JCL der Programmiersprache C ähnlich, zumindest, was die Kontrollfluß–Konstrukte angeht.

Job Control Language

MODULA–2 als JCL

Innerhalb des SPC MODULA–2 Sprachsystems wird einfach MODULA–2 als JCL verwendet. Dadurch ergeben sich einige bemerkenswerte Vorteile gegenüber anderen Lösungen:

- der Software–Entwickler braucht nur eine Sprache zu lernen.
- es stehen alle Konstrukte von MODULA–2 zur Verfügung.
- es stehen alle Bibliotheksfunktionen zur Formulierung von Jobs zur Verfügung.

Operatoren der JCL–Ebene

Auf der JCL–Ebene benötigt der Entwickler natürlich im wesentlichen andere Operatoren, als sie von einer Programmiersprache bereitgestellt werden. Auf der JCL–Ebene werden z.B. Dateien kopiert und nicht Bytes gelesen und geschrieben; oder es werden Programme gestartet, weniger Prozeduren aufgerufen.

der Modul JCL

SPC MODULA–2 stellt diese Operatoren durch den Modul JCL zur Verfügung. Das Starten von Programmen ist durch den Lader ideal gelöst und der Compiler sorgt mit seiner Übersetzungsgeschwindigkeit dafür, daß ein Job genauso schnell entwickelt werden kann, als wenn er interpretiert würde. Einige weitere Vorteile dieses Verfahrens sind so wichtig, daß sie extra genannt werden sollen:

- jeder Job wird durch den Compiler auf seine syntaktische Korrektheit geprüft, bevor er ausgeführt wird.
- ein Job unterscheidet sich von außen nicht von anderen Utilities und kann deshalb auch als Utility

installiert werden.

- die JCL–Operatoren stehen auch "normalen" Programmen zur Verfügung.

Die Grenzen zwischen Job und Utility bzw. Programm verschwimmen. Dies ist kein Nachteil, sondern beabsichtigt. Lediglich die unterschiedliche Behandlung durch die xShell macht den Unterschied zwischen Job und Utility aus.

einen Job starten

Wenn ein Job durch Anklicken gestartet wird, prüft die xShell zunächst anhand des Dateidatums, ob der Job neu compiliert werden muß. Erst wenn die Übersetzung erfolgreich war, wird der Job selbst ausgeführt. Ein Job kann auch über das betreffende Menü gestartet werden. Außerdem sind sind die installierten Jobs auf die Funktionstasten F1 bis F8 aufgelegt.

einen Job installieren

Ein Job wird installiert, indem entweder der Eintrag Install Job im Commands–Menü selektiert wird, oder indem die Taste J gedrückt wird.

einen Job parametrieren

Jobs können wie alle anderen Werkzeuge parametriert werden, indem das Icon bei gedrückter ALTERNATE–Taste doppelt geklickt wird. Danach kann das übliche Formular ausgefüllt werden.

einen Job editieren

Ein Job wird editiert, indem das Icon durch Doppelklick geöffnet wird. Es wird (wie beim Doppelklicken von Dateien) implizit der Editor gestartet.

Fehlerbedingungen

Man beachte, daß beim Starten von Jobs die gleichen Fehlerbedingungen auftreten können, wie beim Starten aller anderen Programme auch.

weitere Möglichkeiten

Ein Job, der zufriedenstellend funktioniert und nicht mehr geändert werden muß, kann natürlich als Utility installiert werden. Umgekehrt bietet es sich an, Programme, die in Arbeit sind, einfach als Job anzumelden.

Textuelle Kommandos

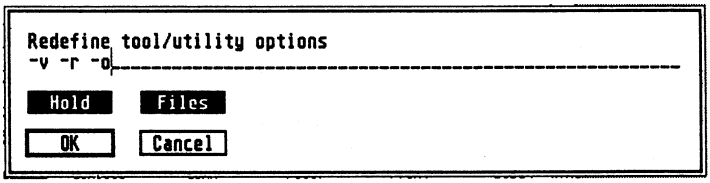
Die xShell bietet eine grafische Oberfläche und Methodik für die Kommandoeingabe. Dabei werden jedoch die vom Benutzer gegebenen grafischen Kommandos auf textuelle Kommandozeilen zurückgeführt. Alle Utilities interpretieren eine standardisierte Kommandozeile wie sie vom Modul CmdLine unterstützt wird.

Kommandozeile eingeben

Die xShell bietet nun neben der grafischen Methodik auch die Gelegenheit, die Kommandozeile über die Tastatur einzugeben. Dazu wird einfach die Taste ESC gedrückt. Es erscheint ein Formular, in das die Kommandozeile eingeben werden kann. Dabei wird die letzte Kommandozeile vorgeschlagen.

zurückliegende Kommandos

Über die Prev– und Next–Buttons können weiter zurückliegende Kommandos aufgeblättert werden.

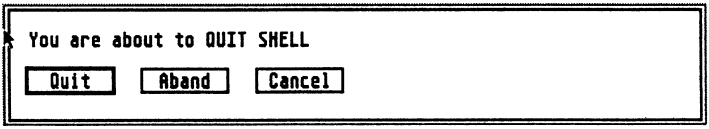


xShell verlassen

Die Shell wird verlassen, indem im Commands–Menü der Eintrag Quit gewählt wird, oder indem über die Tastatur einfach ein Q eingegeben wird. Zur Sicherheit wird über ein Formular noch einmal nachgefragt, ob die xShell wirklich verlassen werden soll.

Optionen

Durch Anklicken von Quit wird die xShell dann endgültig verlassen. Alle Environment–Variablen werden dabei auf das Profile zurückgeschrieben. Dies kann verhindert werden, wenn statt Quit das Abandon–Feld geklickt wird. Immer dann, wenn sie das Profile mit dem Editor verändert haben, sollten sie die xShell mit Abandon verlassen.



Environment–Variablen

Die xShell speichert die Inhalte der unmittelbaren Umgebung in Environment–Variablen ab. Dazu wird eine Environment–Variable XSH sowie bis zu 32 weitere Variablen XSH1 bis XSH32 für jedes belegte Icon benutzt.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Der Editor

Kapitel 4

Übersicht

SPC MODULA-2 enthält einen fensterorientierten Editor, der neben allgemeinen Editierfunktionen weitere Funktionen enthält, die Ihnen das Programmieren in MODULA-2 etwas angenehmer gestalten sollen. Hierzu zählen insbesondere Maßnahmen, die das Editieren selbst sowie die Fehlersuche beschleunigen. Es wurde besonderer Wert darauf gelegt, daß häufige kleine Korrekturen besonders gut von der Hand gehen.

mehrere Dateien editieren

Der Editor erlaubt, mehrere Dateien gleichzeitig in verschiedenen Fenstern zu editieren. Das ist bei der MODULA-2 Programmierung besonders nützlich, da verschiedene Moduln in verschiedenen Dateien abliegen. Bei der Programmierung hat man nun öfter Notwendigkeit, Schnittstellen zu anderen Moduln zu sichten, oder Änderungen in verschiedenen Moduln einzubringen.

direkte Fehleranzeige

Vom Compiler erzeugte Fehlermeldungen werden vom Editor direkt im Quelltext mit einer Klartext-Fehlermeldung angezeigt. Kommandos erlauben von einer Fehlerstelle zur nächsten zu springen.

Der Editor ist als SSWiS-Applikation realisiert. Seine Bedienung ist deshalb zu wesentlichen Teilen auch im Kapitel über SSWiS beschrieben.

den Editor starten

Der Editor wird von der xShell aus gestartet. Dazu wird (s. Kapitel 3) einfach das Editor-Icon geklickt oder über die Tastatur ein E eingegeben. Der Editor ist in der xShell so konfiguriert, daß er Dateiargumente akzeptiert. Falls also eine Datei selektiert war, dann wird sofort diese Datei zum Editieren geöffnet. Andernfalls bietet die xShell eine Dateiauswahl-Box an, über die eine neue Datei ausgewählt werden kann. Diese Datei wird anschließend editiert und gleichzeitig in die unmittelbare Umgebung der xShell aufgenommen.

existierende Datei editieren

Falls die gewählte Datei nicht existiert, wird ein leeres Fenster geöffnet. Wenn Sie keine neue Datei editieren wollen, dann können Sie das Fenster mit SHIFT F10 wieder schließen.

neue Datei anlegen

Die Fenster des Editors können wie gewohnt bedient werden. Der Editor kann mehrere Dateien in verschiedenen Fenstern öffnen. Wie Sie weitere Dateien öffnen können, erfahren Sie weiter unten.

Cursorposition nach dem Start

Falls die gerade geöffnete Datei dieselbe ist, die zuletzt editiert wurde, dann positioniert der Editor den Cursor gleich an die zuletzt editierte Stelle und blättert die entsprechende Seite auf. Das erspart Ihnen in den meisten Fällen, die Stelle, an der Sie gerade arbeiten immer wieder von neuem suchen zu müssen. Die Zeile bzw. Spalte, in der der Cursor steht, nennen wir im folgenden die Cursorzeile bzw. die Cursorspalte. Der Cursor bezeichnet die Stelle, an der Text in die Datei eingegeben werden kann. Der Cursor ist als kleines Dreieck repräsentiert.

Einfache Editierungen

den Cursor bewegen

Der Cursor kann mit den Pfeiltasten in alle vier Richtungen bewegt werden. Wenn er an einem Fensterrand angekommen ist, dann scrollt der Editor das Fenster wenn möglich entsprechend weiter, so daß der Cursor immer sichtbar bleibt. Die Pfeiltasten sind doppelt belegt. Werden die horizontalen Pfeiltasten zusammen mit SHIFT gedrückt, dann springt der Cursor an den Anfang bzw. an das Ende der Zeile. Werden die vertikalen Pfeiltasten zusammen mit SHIFT gedrückt, dann springt der Cursor 10 Zeilen in die gewählte Richtung.

Fenster scrollen

Wenn das Fenster mit den Scrollbalken weitergescrollt wird, dann bleibt der Cursor an seiner Position stehen. Er ist dann i.a. nicht mehr sichtbar. Will man an einer neuen Stelle weiterarbeiten, dann muß man zuerst den Cursor dorthin positionieren, indem man die Stelle mit der Maus anklickt. Will man dagegen wieder an der Stelle weiterarbeiten, an der der Cursor noch steht, dann kann man durch Drücken der ESC-Taste die entsprechende Seite wieder aufblättern. Diese Technik ist besonders nützlich, wenn man nur schnell eine andere Stelle in der gleichen Datei sichten will.

die Cursorzeile aufblättern

Insert- und Replace-Modus

Alle Eingaben der Tastatur erfolgen an der Cursorposition. Der Editor befindet sich normalerweise im Insert-Modus, d.h. neue Zeichen werden eingegeben, indem sie in den Text eingefügt werden. Im Replace-Modus dagegen wird das Zeichen rechts des Cursors mit dem neu eingegebenen überschrieben. Der Wechsel zwischen Insert- und Replace-Modus erfolgt durch Drücken der INSERT-Taste. Eine Ausgabe in der Meldezeile zeigt stets den eingeschalteten Zustand.

Zeichen löschen

Die Tasten DELETE und BACKSPACE löschen das Zeichen rechts bzw. links des Cursors. Besondere Verhältnisse treten ein, wenn der Cursor am Anfang bzw. Ende einer Zeile steht. DELETE am Ende einer Zeile löscht den Zeilenumbruch, d.h. die der Cursorzeile folgende Zeile wird an die Cursorzeile angehängt. BACKSPACE am Anfang einer Zeile hängt die Cursorzeile an die vorangehende Zeile an.

Zeilenanfang und –ende löschen

Die Taste CLRHOME löscht alle Zeichen vom Cursor (einschließlich) bis zum Ende einer Zeile. SHIFT CLRHOME löscht alle Zeichen links des Cursors bis zum Anfang der Zeile.

Zeilen löschen

Die Taste SHIFT DELETE löscht die Cursorzeile und positioniert den Cursor in die der gelöschten Zeile folgenden Zeile.

Löschen rückgängig machen

Alle löschenden Funktionen können durch Drücken von UNDO wieder rückgängig gemacht werden. Wenn Sie z.B. irrtümlich CLRHOME gedrückt haben, dann brauchen Sie nur UNDO zu drücken und die gelöschten Zeichen werden sofort wieder eingefügt. Auch mehrere Fehlbedienungen können durch mehrmaliges Drücken von UNDO wieder rückgängig gemacht werden. Dies funktioniert über 20 Ebenen.

Zeilenumbruch einfügen

Die Taste RETURN fügt im Insert–Modus einen Zeilenumbruch ein, d.h. der Cursor wird an den Anfang einer neuen Zeile bewegt. Im Replace–Modus, dagegen, wird der Cursor einfach nur auf den Anfang der nächsten Zeile gesetzt.

Leerzeilen einfügen

Die INSERT–Taste ist doppelt belegt. SHIFT INSERT erzeugt eine Leerzeile unter der Cursorzeile und positioniert den Cursor an den Anfang der neu eingefügten Zeile.

Spaltennummern anzeigen

Die Taste SHIFT ESC veranlaßt, daß in der Meldezeile die Nummer der Cursorspalte angezeigt wird. Die Spaltenzählung beginnt wie die Zeilenzählung bei 1.

Zeilennummern anzeigen

Die Taste SHIFT HELP veranlaßt, daß die Zeilennumerierung ein– bzw. ausgeschaltet wird. Die Bedeutung der Taste HELP wird später erläutert.

Menüs

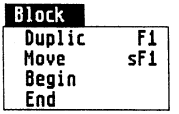
Weitere Editorfunktionen sind über Menüs zugänglich.Der Editor verwendet die von GEM gewohnten Pull-Down Menüs.

Die Menüs werden, wie alle Funktionen des Fenstersystems, über SSWiS (s. Kapitel 8) angesprochen. Die Menüleiste des Editors ist deshalb nur zu sehen, wenn das aktive Fenster auch vom Editor kontrolliert wird. Andernfalls ist die Menüleiste einer anderen Anwendung sichtbar.

Die Bedienung der Pull-Down-Menüs erfolgt wie üblich, indem zuerst ein Menütitel selektiert wird. Anschließend wird im geöffneten Menü ein Eintrag ausgewählt.

Die folgenden Abschnitte orientieren sich an den Menütiteln und erläutern die über Menüs wählbaren Funktionen.

Block Operationen



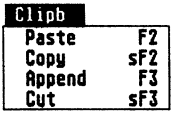
Das Block-Menü ermöglicht die Festlegung von Textblöcken und elementaren Operationen auf ihnen. Mit der Funktion MarkBeg wird ein neuer Block markiert. Er beginnt an der Cursorposition und endet (vorerst) an der Cursorposition. Nachdem der Cursor weiter in Richtung Dateiende verschoben wurde, kann der Block mit MarkEnd bis zur Cursorposition ausgedehnt werden. Der Block wird durch eine Hinterlegung der betroffenen Zeichen sichtbar gemacht. Eine Kurzform der Blockselektion ist über die Maus realisiert. Drücken (und Halten) der linken Maustaste selektiert einen neuen Block an der Stelle des Mauszeigers. Nun kann die Maus bewegt werden. Das Blockende wird durch Loslassen der Maustaste festgelegt. Weiterhin kann das Ende des Blockes noch korrigiert werden, wenn die SHIFT-Taste mit der linken Maustaste gedrückt wird. Verläßt man bei gedrückter Maus-Taste das Editor-Fenster, dann scrollt der Editor das Fenster in die entsprechende Richtung weiter.

Die Funktion Duplicate kopiert den ausgwählten Block in einen internen Zwischenpuffer und fügt ihn gleichzeitig an der Cursorposition wieder ein. Die Cursorposition wird dabei nicht verändert. Die Selektion ist danach wieder aufgehoben.

Die Funktion Move kopiert den Block ebenfalls in den Zwischenpuffer, löscht ihn aus der Datei und fügt ihn an der Cursorposition wieder ein.

Eine beim Programmieren in MODULA–2 nützliche Funktion besteht darin, ganze Blöcke spaltenweise nach links oder nach rechts zu bewegen. Dazu wird der entsprechende Textblock wie oben beschrieben selektiert und dann bei gedrückter CONTROL–Taste mit den Cursor–Tasten nach links oder rechts geschoben.

Clipboard



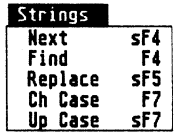
Das Clipboard–Menü stellt weitere Operationen auf dem Zwischenpuffer bereit. Das Übernehmen eines ausgewählten Blockes in den Zwischenpuffer erfolgt mit Copy.

Cut kopiert den Block in den Zwischenpuffer und löscht ihn dann aus der Datei.

Paste fügt den Inhalt des Puffers an der Cursorposition wieder ein. Die Folge Copy/Paste entspricht also dem Kommando Copy des Block Menüs. Die Folge Cut–Paste entspricht dem Move Kommando.

Append erlaubt es, einen Block an das Ende des Puffers anzufügen. Alle anderen oben erwähnten Kommandos löschen den alten Inhalt des Zwischenpuffers.

Das String–Menü macht Funktionen zum Suchen und Ersetzen von Zeichenketten zugänglich. Mit Find kann eine Zeichenkette angegeben werden, nach der gesucht werden soll. Gleichzeitig kann die Suchrichtung spezifiziert werden. Darauf wird die angegebene Zeichenkette in der Datei gesucht. Der Fortschritt beim Suchen sowie der Ausgang der Operation werden in der Meldezeile angezeigt. Falls die Zeichenkette gefunden wurde, wird der Cursor an deren Anfang positioniert.



Next erlaubt, die Suche ab der Cursorposition mit den gleichen Parametern fortzusetzen.

Über Replace kann eine Zeichenketten–Ersetzung durchgeführt werden. Hierzu wird die Zeichenkette angegeben, durch die die zu suchende Zeichenkette ersetzt werden soll. Weiter kann angegeben werden, ob vor einer Ersetzung zunächst eine explizite Bestätigung verlangt werden soll und ob alle Stellen ersetzt werden sollen. Falls während der Ersetzung eine Bestätigung verlangt wird, besteht die Möglichkeit, die Zeichenkette zu ersetzen, eine Ersetzung abzulehnen oder den ganzen Vorgang abzubrechen.

ChCase macht das Zeichen unter dem Cursor zu einem Großbuchstaben, wenn es ein Kleinbuchstabe war und umgekehrt. UpCase macht das Zeichen unter dem Cursor zu einem Großbuchstaben.

im Text springen

Das Goto–Menü erlaubt das Positionieren des Cursors durch Angabe von Zeilennummern oder auf die Marke (Label).

Zeilennummer

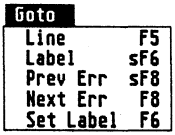
Mit Line kann eine Zeilennummer (und eine Spaltennummer) angegeben werden, auf die der Cursor positioniert werden soll.

Marke

Label veranlaßt, daß der Cursor zu einer vorher gesetzten Marke springt. Das ist nützlich, wenn man kurzzeitig eine andere Stelle eines Dokumentes bearbeiten will, und anschließend an die markierte Stelle zurückkehren will.

Fehlerposition

Next Error positioniert den Cursor auf den nächsten Fehler nach der momentanen Cursor–Position. Fehler werden beim Öffnen eines Fensters aus der Fehlerdatei des Compilers (ERR.LST) übernommen und in die geöffnete Datei eingetragen.



Von besonderem Vorteil ist, daß auch nach dem Einfügen und Löschen von Zeilen und Zeichen die Fehler an der richtigen Position angezeigt werden.

Prev Error positioniert den Cursor auf die der Cursor–Position vorangegangene, Fehlerstelle. Falls Sie den Fehler schon behoben haben, ist diese Funktion natürlich sinnlos.

Modi einstellen

Das Mode-Menü unterstützt diverse Modi des Editors, die jeweils ein- oder ausgeschaltet werden können. Die jeweils eingestellten Modi sind am linken Rand der Meldezeile kenntlich gemacht.

Zeilennumerierung

Numbers schaltet die Zeilennumerierung ein oder aus, was durch entsprechende Meldungen angezeigt wird. Es entspricht damit der Taste SHIFT HELP.

Insert und Replace

Insert schaltet wie die gleichnamige Taste zwischen dem Insert- und dem Replace-Modus hin und her.

automatisches Einrücken

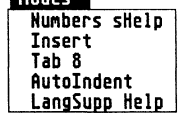
AutoIndent schaltet die automatische Einrückung an oder aus. Im eingeschalteten Zustand wird eine neue Zeile (nach RETURN) automatisch genauso weit eingerückt wie die vorangehende Zeile. Im Replace-Modus wird der Cursor nach RETURN automatisch auf das erste von Blank verschiedene Zeichen der Zeile positioniert.

Tabulierung

Tab 8 schaltet auf eine 8er Tabulation um. Beim Programmieren ist es dagegen meist sinnvoll mit der Taste TAB auf die Spalte zu positionieren, in der in der vorangehenden Zeile Text beginnt (Insert-Modus), bzw. auf die (Replace-Modus), in der Cursorzeile Text beginnt. Dadurch wird spaltengenaues Formatieren beim Editieren unterstützt.

Sprachunterstützung

LangSupp schaltet die Sprachunterstützung ein bzw. aus. Bei eingeschalteter Unterstützung interpretiert der Editor klein geschriebene Wörter als MODULA-2 Schlüsselwörter. Sobald Eindeutigkeit besteht (meist nach dem zweiten oder dritten Zeichen) ersetzt der Editor die kleinen Buchstaben durch das ganze (groß geschriebene) Schlüsselwort. So genügt für die Eingabe des Schlüsselwortes PROCEDURE die Eingabe der Buchstaben pr. Bei der Editierung von Kommentaren kann diese Funktion mitunter hinderlich sein, weshalb sie abschaltbar ist. Zur Vereinfachung ist die gleiche Funktion auch auf der Taste HELP aufgelegt.



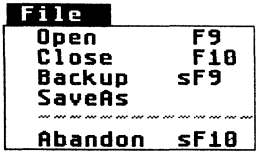
Dateien

Das File–Menü enthält mit die wichtigsten Funktionen des Editors, nämlich die zum Öffnen und Schließen von Dateien.

Öffnen und Schließen

Open öffnet weitere Dateien in weiteren Fenstern. Dazu bietet der Editor eine Dateiauswahl–Box an.

Close schreibt eine Datei auf Massenspeicher zurück und schließt das Fenster.



Backup schreibt die Datei zurück, ohne das Fenster zu schließen. Während längerer Editierungen sollte in jedem Fall zwischendurch Backup aufgerufen werden, um bei irgendwelchen Problemen keinen allzu großen Verlust an Daten zu erleiden.

ohne zu sichern verlassen

Abandon schließt die Datei ohne auf Massenspeicher zurückzuschreiben! Falls die Datei geändert wurde, wird zuerst noch einmal über ein Formular nachgefragt.

Die Originaldatei bekommt beim Zurückschreiben die Endung .BAK und steht somit weiterhin zur Verfügung.

Funktions–tasten

Einige der über Menüs bereitgestellten Funktionen sind auch über die Funktionstasten F1 bis F10 und SHIFT F1 bis SHIFT F10 zugänglich. Dies dient wiederum der Beschleunigung des Editiervorgangs. Zur Unterstützung des Bedieners liegen dem Handbuch Aufkleber bei, die über die entsprechenden Funktionstasten geklebt werden können. Außerdem wird in den Menüs durch eine Beschriftung am Ende des Menü–Eintrages jeweils angezeigt, ob eine Funktion auch auf einer Funktionstaste aufliegt.

Zehnerblock

Auf die Tasten des numerischen Tastenblocks können Textmakros aufgelegt werden. Die Textmakros können dann an der Cursorposition eingefügt werden. Zur Definition eines Textmakros wird die entsprechende

Taste zusammen mit SHIFT gedrückt. In das daraufhin erscheinende Formular kann ein beliebiger Text eingetragen werden.

Sonstiges

Fehlermeldungen des Compilers werden in die Datei ERR.LST geschrieben. Diese Datei wird sofort beim Starten des Editors geöffnet. Beim Öffnen weiterer Fenster schaut der Editor sofort in der Fehlerdatei nach, ob für den neuerlich geöffneten Modul vom Compiler Fehler eingetragen sind. In der Fehlerdatei stehen alle mit der letzten Compilierung übersetzten Moduln, und jeweils darunter die Fehler (falls Fehler entdeckt wurden). Die Fehlermeldungen bestehen jeweils aus drei Zahlen in der Form

<Zeile> <Spalte> <Fehlernummer>.

diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Der Compiler

Kapitel 5

Übersicht

Der Compiler hat die Aufgabe, MODULA–2 Programme von ihrer Quelltextform in ihre Objektform zu überführen.

Wie jeder MODULA–2 Compiler verarbeitet der SPC MODULA–2 Compiler zwei Arten von Eingabedateien, nämlich Modulschnittstellen (DEFINITION MODULEs) und Implementierungen (IMPLEMENTATION MODULEs). Je nachdem, von welchem Typ die Eingabedateien sind, werden unterschiedliche Ausgabedateien erzeugt.

Kommando–Dateien

Weiterhin akzeptiert der SPC MODULA–2 Compiler sogenannte Command–Files. Diese enthalten Namen von Moduln, die übersetzt werden müssen. Command–Files sind wichtig, da mitunter ein ganzes, aus vielen Moduln bestehendes System übersetzt werden muß. Die Dateien mit der Dateiauswahl–Box anzugeben wäre dann nicht angemessen.

dynamisches Binden

Dank des speziellen Formates der übersetzten Moduln müssen SPC MODULA–2 Moduln nicht mehr explizit gebunden werden, sondern können sofort gestartet werden. Der dabei dennoch notwendige Bindevorgang wird von einem Lader dynamisch, d.h. während des Ladevorgangs ausgeführt.

Single–Pass–Compiler

Der SPC MODULA–2 ist ein sogenannter Single–Pass–Compiler (SPC). Als solcher liest er die Eingabedatei nur einmal und erzeugt keine Zwischendateien, sondern baut im Speicher die notwendigen Strukturen auf. Dadurch wird der Übersetzungsvorgang enorm beschleunigt. Der Mehrbedarf an Speicher ist auf den moderen Mikrorechnern normalerweise kein Problem mehr, sodaß der Vorteil der hohen Geschwindigkeit

überwiegt. Auf einem ATARI ST können bei günstigen Verhältnissen Geschwindigkeiten bis zu 5000 Zeilen pro Minute erreicht werden.

Fehlerdatei

Die Namen aller mit der letzten Übersetzung übersetzten Moduln (evtl. nur einer) und dabei evtl. aufgetretene Fehler sind in einer Fehlerdatei vermerkt. Die Datei kann vom Editor wieder interpretiert werden, um die Fehler eines Moduls direkt im Quelltext anzuzeigen.

Sprachstandard

Die vom Compiler implementierte Sprache entspricht dem neuesten Stand der MODULA-2 Entwicklung an der ETH-Zürich. Die Details der Sprachimplementierung werden im Rahmen dieses Kapitels beschrieben. Wenn MODULA-2 vom ISO standardisiert ist, wird SPC MODULA-2 auf den dann gültigen Sprachstandard umgestellt werden.

Starten

Der Compiler wird über eine Standard–Kommandozeile aufgerufen und parametriert. Die Kommandozeile hat die Form:

```
compile <Dateiname> [-r] [-o] [-v]
```

Die zulässigen Dateinamen werden unten erklärt. Der Compiler akzeptiert nur einen Dateinamen.

–r Option

Die Option –r schaltet die Bereichsprüfung ein. Dadurch werden alle Zuweisungen an Unterbereichs– und enumerierte Variablen auf Zulässigkeit geprüft. Neben Zuweisungen werden die aktuellen Parameter in Prozeduraufrufen, sowie die Indizes bei Array–Zugriffen geprüft. Die Zulässigkeit von CASE–Ausdrücken wird, falls keine ELSE–Klausel exisitiert, ebenfalls geprüft.

–o Option

Die Option –o aktiviert die Prüfungen auf arithmetische Über– bzw. Unterläufe. Diese sind bei allen arithmetischen Ausdrücken wirksam.

–v Option

Sowohl die –r als auch die –o Option bewirken, daß zusätzlicher Code erzeugt wird. Die Option –v (verbose) veranlaßt den Compiler, den Übersetzungsverlauf am Terminal zu protokollieren.

über die xShell starten

Der Compiler wird normalerweise über die xShell aufgerufen. Der Dateiname wird dann ebenfalls von der xShell beigesteuert (s. Kapitel 3). Die Optionen können durch Parametrieren des Compiler–Icons der xShell eingestellt werden.

Ein- und Ausgabe-dateien

Die vom Compiler benutzten Dateien werden durch die Namenserweiterung (Extension) typisiert. Der Namensstamm ergibt sich aus den ersten Buchstaben des Modulnamens. Die Anzahl der signifikanten Zeichen ist vom Betriebssystem abhängig. Unter GEM werden 8 Zeichen ausgewertet. Für die Namenserweiterung gelten die folgenden Konventionen:

.DEF

.DEF – bezeichnet einen Definitions-Quellmodul. Der Modul beginnt mit der Konstruktion DEFINITION MODULE. Für jeden Definitions-Modul wird auch ein Implementierungs-modul benötigt.

.MOD

.MOD – bezeichnet einen Implementierungs-Quellmodul. Handelt es sich um einen sogenannten Programm-Modul, dann beginnt der Modul mit der Konstruktion MODULE, und der Compiler erwartet keinen Definitions-Modul. Andernfalls handelt es sich um einen IMPLEMENTATION MODULE und der Compiler sucht nach dem entsprechenden Definitions-Modul.

.SBM

.SBM – ein DEFINITION MODULE wird vom Compiler in seine Objektform übersetzt. Beim Importieren des Moduls in andere Moduln wird später nur die Objektform des DEFINITION MODULEs gelesen. Der Namensstamm ist der gleiche wie der des Quellmoduls.

.OBM

.OBM – kennzeichnet die Objektform eines Implementierungs-Moduls. Der Namensstamm ist der gleiche wie der der zugehörigen .MOD Datei. Die Objektmoduln werden später vom Lader oder vom Linker verarbeitet.

.RFM

.RFM – bezeichnet eine weitere aus der Übersetzung hervorgehende Datei, die Informationen für den Debugger enthält. Der Namensstamm ist gleich dem des Implementierungs-Moduls.

.CMD

.CMD – ist eine Eingabedatei, die eine Liste von Kommandozeilen für den Compiler enthält. Der Compiler erkennt eine solche Datei an ihrer Endung und inter-

pretiert alle darin enthaltenen Compiler–Kommandozeilen.

ERR.LST – ist die Fehler– und Protokolldatei. Sie enthält die Namen aller mit der letzten Compilation übersetzten Moduln und evtl. aufgetretene Fehler. Die Fehler werden durch die Zeilennummer, die Spaltennummer und die Fehlernummer beschrieben. Obwohl man die Fehlerdatei mitunter inspizieren will, wird man normalerweise die Funktionen des Editors benützten, um sich den Fehler direkt im Programmtext anzeigen zu lassen.

Fehlerdatei ERR.LST

Suchpfade

Da der Compiler während der Übersetzung mehrere Moduln importiert, also verschiedene Dateien eröffnet, erhebt sich die Frage, wo diese Dateien abliegen müssen, damit sie der Compiler findet. Meist ist es nämlich sinnvoll, Moduln in unterschiedlichen Ordnern unterzubringen, um eine Struktur in der Menge aller Moduln zu erhalten. Z.B. sind die Moduln der STDLIB in einem anderen Ordner zusammengefaßt, als die der SYSLIB. Die Namen der Ordner werden Suchpfade genannt, da sie vom Compiler durchsucht werden müssen, um einen Modul zu finden. Suchpfade werden immer in einer bestimmten Reihenfolge durchsucht. Die Suche bricht ab, wenn der Modul bzw. die Datei gefunden wurde.

Modulvorrat strukturieren

Der Compiler erfährt die Suchpfade aus Environment–Variablen mit den Namen Path1 bis Path<N>, wobei <N> im Prinzip beliebig ist. Im Interesse einer schnellen Übersetzung sollte <N> jedoch nicht zu groß sein, bzw. die meisten Dateien sollten auf Pfaden mit einer niedrigen Nummer liegen. Der Compiler konstruiert aus dem Pfadnamen, dem Namensstamm des zu übersetzenden oder importierenden Moduls und des benötigten Typs einen Dateinamen und versucht die Datei zu öffnen. Gelingt dies, dann schreitet die Übersetzung fort. Anderfalls wird der nächste Pfad (mit der nächst

Path1 bis PathN

höheren Nummer) herangezogen. Das Verfahren bricht ab, wenn es keine Environment-Variable mit dem Namen Path<N+1> gibt.

ObjPath1 bis ObjPathN

Mitunter möchte man die Quellmoduln in anderen Ordnern halten, als die übersetzten Moduln. Immerhin sollten Sie ihre Quellmoduln regelmäßig sichern, während Sie die Objektformen der Moduln jederzeit wieder neu erzeugen können. Der Compiler unterstützt eine solche Organisation, indem er bei allen Objektformen (.SBM, .OBM, .RFM) zunächst nach einer Environment-Variablen ObjPath<N> sucht. Falls eine solche Variable existiert, werden Objektformen von Moduln die auf Path<N> gefunden wurden auf ObjPath<N> abgelegt und von dort importiert. Falls keine entsprechende Variable existiert, benützt der Compiler Path<N> auch für die Objektformen der Moduln.

Die angegebenen Regeln zum Suchen von Moduln werden vom Modul Environment implementiert und stehen auch anderen Prorammen zur Verfügung. Insbesondere alle anderen Werkzeuge des Sprachsystems machen davon Gebrauch.

Pfade einstellen

Suchpfade können mit der Utility Paths inspiziert, gesetzt und gelöscht werden. Eine andere Methode ist, das Profile direkt mit dem Editor zu bearbeiten.

☞ Beachten Sie bitte in diesem Fall, daß sie die xShell über Abandon verlassen, da sonst das editierte Profile wieder überschrieben wird.

Modul-schlüssel

Ein MODULA-2 Programm besteht normalerweise aus vielen Moduln. Hierzu gehören auch die aus den mitgelieferten Bibliotheken benützten Moduln. Die Schnittstellen zwischen Moduln sind in den DEFINITION MODULEs beschrieben und werden vom Compiler in eine Objektform (.SBM) übersetzt. Alle Moduln, die auf diese Schnittstellen Bezug nehmen, werden vom Compiler auf Konsistenz mit den benützten Schnittstellen geprüft. Dabei werden z.B. falsche Parameterbestückungen von Prozeduraufrufen, etc. entdeckt. Durch die automatische Prüfung der Schnittstellen durch den Compiler werden viele Fehlermöglichkeiten ausgeschlossen, die in anderen Programmiersprachen immer wieder zu hohen Testaufwänden führen.

automatische Schnittstellenprüfung

Da auch DEFINITION MODULEs sich auf andere Modulschnittstellen beziehen können, muß eine Reihenfolge der Übersetzung von Moduln eingehalten werden. Für alle Moduln eines Systems gilt die folgende einfache Regel:

Übersetzungs-reihenfolge

☞ Jeder DEFINITION MODULE muß vor seinem ersten Import übersetzt werden.

Dieser Zusammenhang soll an einem Beispiel veranschaulicht werden. Ein Modul A nehme auf die Schnittstellen zweier Moduln B1 und B2 Bezug. Die Schnittstelle von B1 (DEFINITION MODULE) muß auf jeden Fall vor der Implementierung von B1 (IMPLEMENTATION MODULE) übersetzt werden, da sich natürlich jede Implementierung auf ihre eigene Schnittstelle bezieht. Das gleiche gilt für die Schnittstelle und die Implementierung von B2. Der Modul A (Implementierung) kann erst übersetzt werden, wenn seine eigene Schnittstelle und die von B1 und B2 übersetzt sind. Die Moduln A, B1 und B2 passen nun mit Sicherheit zusammen, da sie sich auf die gleiche Objektform der Schnittstellen B1 und B2 bezogen haben. Die Implementierungen aller Moduln

Beispiel

können beliebig oft übersetzt werden, ohne daß sich daran etwas ändert.

Inkonsistenzen

Wenn nun aber einer der Schnittstellenmoduln neu übersetzt wird, besteht die Möglichkeit, daß Moduln, die sich darauf beziehen nicht mehr von der gleichen Schnittstelle ausgehen, es sei denn, sie werden nach den Schnittstellen noch einmal übersetzt. Unterbleibt dies, dann muß der Linker bzw. der Lader feststellen, daß die Moduln nicht zusammenpassen.

Modulschlüssel

Dazu gibt der Compiler jeder Objektform eines Moduls (.SBM und .OBM) einen sogenannten Modulschlüssel mit, an dem Compiler, Linker und Lader das Zusammenpassen von Moduln überprüfen können. Falls dabei eine Inkonsistenz entdeckt wird, wird gemeldet, daß die Moduleschlüssel (Module Keys) nicht zusammenpassen.

.CMD Datei verwenden

Bei einem größeren System ist es mitunter recht mühsam, die richtige Übersetzungsreihenfolge einzuhalten. Es ist deshalb ratsam, von Anfang an eine Kommandodatei (.CMD) zu erstellen, in der die Moduln in der richtigen Reihenfolge aufgeführt sind. Falls später im System Schnittstellen geändert werden müssen, ist es dann meist am einfachsten, das ganze System neu zu übersetzen.

die SPC Implementierung

Die vom Compiler implementierte Sprache ist durch N.Wirth's "Programming in MODULA-2" beschrieben. Die folgenden Abschnitte beschreiben Details von SPC MODULA-2 auf dem ATARI ST.

Datentypen

Die vom Compiler unterstützten Datentypen umfassen die Standarddatentypen, sowie als Erweiterung 32 Bit lange Varianten von 16-Bit-Typen, sowie einen 64 Bit langen REAL Typ. Die unterstützten Typen sind im Einzelnen:

- INTEGER – der Typ belegt 2 Bytes und umfaßt einen Wertebereich von -32768..32767.

INTEGER

- LONGINT – der Typ (Erweiterung) belegt 4 Bytes und umfaßt einen Wertebereich von -2147483648 .. 2147483647.

LONGINT

- CARDINAL – der Typ belegt 2 Bytes und umfaßt einen Wertebereich von 0..65535.

CARDINAL

- LONGCARD – der Typ (Erweiterung) belegt 4 Bytes und umfaßt einen Wertebereich von 0..4294967295. Der Typ LONGCARD kann auf dem ATARI ST in eine Adresse konvertiert werden. Programme, die davon Gebrauch machen, sind natürlich maschinenabhängig.

LONGCARD

- REAL – der Typ belegt 4 Bytes und ist als IEEE Single-Precision Real implementiert. Die Mantisse belegt dabei 23 Bits, der Exponent 8 Bits. Der Wertebereich reicht von ca. -3.3E38 bis +3.3E38.

REAL

- LONGREAL – der Typ (Erweiterung) belegt 8 Bytes und ist als IEEE Double-Precision Real implementiert. Die Mantisse hat 52 Bits, der Exponent 11 Bits. Der Wertebereich reicht von -1.79E308 bis +1.79E308.

LONGREAL

- BITSET – der Typ ist als SET OF [0..15] definiert. Er belegt 2 Bytes.

BITSET

- LONGBITSET – der Typ (Erweiterung) ist als SET OF [0..15] definiert. Er belegt 2 Bytes.

LONGBITSET

- CHAR – der Typ belegt ein Byte.

CHAR

BOOLEAN

- BOOLEAN – der Typ umfaßt die Werte TRUE und FALSE und belegt 1 Byte.

Enumerationen

- Enumerationstypen – belegen 1 Byte, d.h. die Anzahl der Elemente ist auf 256 beschränkt.

SET OF

- SET OF – Typen belegen 2 oder 4 Bytes, d.h. die Anzahl der Elemente ist auf 32 beschränkt.

PROCEDURE

- PROCEDURE – Typen belegen 4 Bytes, da sie auf Pointers zurückgeführt werden.

POINTER TO

- POINTER – Typen belegen 4 Bytes.

zusammengesetzte Typen

Der Compiler legt alle Datenelemente, die mehr als ein Byte belegen, auf geraden Adressen ab. Alle 1 Byte Typen können auch auf ungeraden Adressen zu liegen kommen.

Kompatibilität

MODULA–2 verlangt bei allen Ausdrücken, daß die beteiligten Operanden vom gleichen (kompatiblen) Typ sind. Dadurch verbietet es sich, INTEGER Operanden mit CARDINAL Operanden zu vergleichen, usw. Nötigenfalls müssen die Operanden durch eine typkonvertierende Funktion auf den richtigen Typ gebracht werden. Die Konvertierungsfunktionen werden später dokumentiert.

Zuweisungs–Kompatibilität

Bei der Zuweisung von Werten an Variablen und bei der Übergabe von Parametern (nicht VAR–Parameter) wird bei MODULA–2 eine großzügigere Regelung angewandt, indem bestimmte Typen zuweisungskompatibel sind. INTEGER und CARDINAL sind zuweisungskompatibel, d.h. ein INTEGER–Wert darf einer CARDINAL–Variable ohne weitere Konvertierung zugewiesen werden, und umgekehrt. Weiterhin sind bei SPC MODULA–2 die Typen LONGCARD, LONGINT, CARDINAL und INTEGER untereinander sowie die Typen REAL und LONGREAL untereinander zuweisungskompatibel. INTEGER und REAL dagegen sind nicht zuweisungskompatibel und müssen explizit mit entsprechenden Konvertierungs–Funktionen umgewandelt werden.

☞ Man beachte, daß bei Typverengungen u.U. Überläufe auftreten können.

Typtransfers – VAL

Eine andere Möglichkeit, den Typ einer Variablen oder Konstanten zu ändern ist durch den sogenannten Typtransfer gegeben. Ein Typtransfer schaltet einfach vorübergehend die Typprüfung des Compilers ab. Es wird kein Code zur Konversion des Typs erzeugt. Das Ergebnis eines Typtransfers ist von der Bit–Repräsentation des Eingangs– und des Ausgangstyps abhängig. Ein Typtransfer ist deshalb hochgradig systemabhängig. Die Typtransfer–Funktion VAL wird deshalb bei SPC MODULA–2 von dem Pseudo–Modul SYSTEM exportiert, wodurch sich Moduln, die Typtransfers verwenden, explizit als systemabhängig erklären müssen.

Pseudomodul SYSTEM

Weitere systemabhängige Typen und Funktionen werden von dem Pseudo-Modul SYSTEM exportiert. SYSTEM wird deshalb als Pseudo-Modul bezeichnet, da es keinen DEFINITION MODULE dafür gibt. Vielmehr sind die von SYSTEM exportierten Elemente dem Compiler selbst bekannt. Zum Zwecke der Dokumentation ist die Schnittstelle von SYSTEM jedoch in Anhang B als DEFINITION MODULE aufgeschrieben.

ADDRESS und ADR

Zu den Elementen von SYSTEM gehört insbesondere der Datentyp ADDRESS und der ADR-Operator, der die Adresse einer Datenstruktur liefert. Der Datentyp ADDRESS ist kompatibel mit jedem POINTER-Typ, sowie mit dem Typ LONGCARD, wodurch Adress-Arithmetik möglich wird. Die Adresse einer Datenstruktur erhält man durch die Konstruktion ADR(<Variable>).

BYTE und ARRAY OF BYTE

Der Datentyp BYTE repräsentiert die kleinste auf der Maschine adressierbare Speichereinheit. BYTE wird nicht interpretiert. ARRAY OF BYTE ist mit jedem Typ kompatibel.

WORD

Der Datentyp WORD repräsentiert ein Maschinenwort. Beim ATARI ST belegt WORD 2 Bytes und liegt immer auf geraden Adressen. WORD wird nicht interpretiert.

SETREG und REG

Der Zugriff auf Register des MC68000 ist über die Funktionen SETREG und REG möglich. Die Register werden mit Indizes von 0 bis 15 bezeichnet. D0 hat den Index 0, A0 8 und A7 hat den Index 15. Das Ergebnis von REG ist vom Typ LONGINT. Das Argument von SETREG ist vom Typ LONGINT oder von einem Adress-Typ.

LONG und SHORT

Die Funktion LONG akzeptiert INTEGER- oder CARDINAL-Argumente und liefert ein LONGINT-Ergebnis zurück. Die Funktion SHORT dagegen wandelt LONGINT- oder LONGCARD-Argumente in INTEGER-Ergebnisse um. Die Funktione zählen damit zu den Typkonvertierungs-Funktionen.

SHIFT

Die Funktion SHIFT akzeptiert einen skalaren Typ als Argument und schiebt das Bitmuster um N Bits nach links oder rechts, je nachdem ob N größer oder kleiner 0 ist.

VAL

Die Funktion VAL realisiert den Typtransfer. Da sie einen Modul systemabhängig macht, wird sie bei SPC MODULA-2 von SYSTEM exportiert. Dadurch ist der Programmierer gezwungen, explizit aus SYSTEM zu importieren und seinen Modul so deutlich als systemabhängig zu markieren. Für den Typtransfer mit VAL wird kein zusätzlicher Code erzeugt (unsafe Transfer).

INLINE

Die Prozedur INLINE erlaubt es, in einem Modul Maschinecode-Sequenzen abzusetzen. INLINE akzeptiert beliebig viele (bis zu einer Obergrenze) CARDINAL-Argumente. Die Argumente werden ohne weitere Interpretation in den Code eingefügt. Man beachte, daß durch die Funktionen REG, SETREG, ADR und das unten erläuterte Konzept der Code-Prozeduren normalerweise keine Notwendigkeit für INLINE-Code entsteht.

CODE-Prozeduren

SPC MODULA-2 stellt sogenannte CODE-Prozeduren zur Verfügung. Dadurch ist es elegant möglich, Betriebssystemaufrufe über Traps abzusetzen, ohne auf INLINE-Statements angewiesen zu sein. Eine CODE-Prozedur hat einen Prozedurkopf (Name und Parameterliste) wie jede andere Prozedur auch. Der Prozedurrumpf wird jedoch duch die Konstrktion CODE <N> ersetzt. Der Compiler wird beim Aufruf der Prozedur zunächst die Parameter auf den Stack bringen, und dann die Prozedur aufrufen, indem er <N> als Code absetzt. Die Deklaration

das Betriebssystem aufrufen

```
ConOut (Ch : CHAR; Func02 : INTEGER); CODE
4E41H;
```

beschreibt die Schnittstelle zur GemDos-Funktion Nummer 2.

☞ Man beachte, daß die Systemaufrufe auf dem ATARI ST erwarten, daß der rufende Modul den Stack abräumt. Dazu ist in SPC-MODULA-2 noch ein kurzes INLINE-Codestück erforderlich.

Beispiel

Die Anbindung an Systemaufrufe mit Codeprozeduren ist am Beispiel des Moduls GemDos in Anhang H erläutert.

FORWARD

In SPC MODULA-2 müssen alle Prozeduren vor ihrem erstmaligen Aufruf deklariert werden. Eine Prozedur, die im DEFINITION MODULE deklariert ist, ist offensichtlich immer vor ihrem ersten Aufruf deklariert. Bei lokalen Prozeduren ist die Reihenfolge meist durch die Hinschreibung der Prozeduren einhaltbar. In Ausnahmefällen muß eine Prozedur FORWARD deklariert werden. Dazu wird der ganze Prozedurkopf z.B. am Anfang des Moduls wiederholt, und der Prozedurrumpf durch das Statement FORWARD ersetzt. Die FORWARD Deklaration muß die volle Parameterliste enthalten und auf der selben Schachtelungstiefe erfolgen, wie die Prozedurdeklaration selbst.

Standard-Prozeduren

Standard-Prozeduren sind vordefinierte Prozeduren oder Funktionen, die dem Compiler selbst bekannt sind, und die nicht explizit importiert werden müssen. Die Standard-Prozeduren in SPC MODULA-2 sind in alphabetischer Reihenfolge:

ABS

ABS(x) x ist vom Typ INTEGER, LONGINT, REAL oder LONGREAL (oder dazu zuweisungskompatibel). Das Ergebnis ist der Absolutbetrag des Arguments und ist vom gleichen Typ wie das Argument.

CAP

CAP(ch) ch ist vom Typ CHAR. Falls ch ein Kleinbuchstabe ist, dann liefert CAP den entsprechenden Großbuchstaben, sonst ch. Das Ergebnis von CAP ist mithin vom Typ CHAR.

CHR

CHR (x) x ist vom Typ INTEGER (oder dazu zuweisungskompatibel). Das Ergebnis ist vom Typ CHAR und enthält das Zeichen mit der Ordnungsnummer (Ordinalität) x.

DEC

DEC (x,n) x ist ein skalarer Typ. DEC erniedrigt die Ordinalität von x um n. n ist folglich vom Typ INTEGER. Die Form DEC (x) erniedrigt x um 1.

EXCL

EXCL (s,i) s ist ein SET-Typ, i ist vom Basistyp von s. Es wird die Operation s:= s - i ausgeführt.

FLOAT

FLOAT (x) x ist vom Typ INTEGER oder LONGINT. x wird in einen REAL-Wert konvertiert und als Ergebnis zurückgeliefert.

FLOATD

FLOATD (x) konvertiert x in die LONGREAL Darstellung.

HALT

HALT beendet die Programmausführung.

HIGH

HIGH (a) a ist ein Array oder ein Array-Typ. Das Ergebnis ist vom yp INTEGER und enthält den höchsten Index des Arrays a. Man beachte, daß bei sogenannten Conformant Arrays der niedrigste Index zu 0 angenommen wird. Die Länge ergibt sich dann zu HIGH(a)+1.

INC	INC (x,n) x ist ein skalarer Typ. INC erhöht die Ordinalität von x um n. n ist folglich vom Typ INTEGER. Die Form INC (x) erhöht x um 1.
INCL	INCL (s,i) s ist ein SET–Typ, i ist vom Basistyp von s. Es wird die Operation s:= s + i ausgeführt.
MAX	MAX (t) t ist ein skalarer Typ (einschließlich REAL und LONGREAL). Das Ergebnis ist ebenfalls vom Typ t und enthält den höchsten durch den Typ t darstellbaren Wert.
MIN	MIN (t) t ist ein skalarer Typ (einschließlich REAL und LONGREAL). Das Ergebnis ist ebenfalls vom Typ t und enthält den niedrigsten durch den Typ t darstellbaren Wert.
ODD	ODD(x) x ist vom Typ INTEGER, LONGINT, CARDINAL oder LONGCARD. Das Ergebnis von ODD ist vom Typ BOOLEAN und zeigt an, ob x ungerade ist.
ORD	ORD (x) x ist vom Typ INTEGER, CARDINAL, CHAR oder ein Enumerationstyp. Das Ergebnis ist vom Typ INTEGER und gibt die Ordnungsnummer des Arguments innerhalb des Argumenttyps an.
SIZE	SIZE (x) x ist entweder eine Variable oder ein Typ. Das Ergebnis ist vom Typ INTEGER und enthält die Anzahl von Bytes, die die Repräsentierung von x belegt. ☞ Man beachte, daß SIZE nicht von SYSTEM importiert wird und TSIZE ersetzt.
TRUNC	TRUNC (x) x ist vom Typ REAL. Das Ergebnis vom Typ INTEGER enthält den ganzzahligen Anteil von x.
TRUNCD	TRUNCD (x) x ist vom Typ LONGRAL. Das Ergebnis vom Typ LONGINT enthält den ganzzahligen Anteil von x.

Die Codegröße eines Moduls ist bei SPC MODULA–2 auf 32k Bytes beschränkt. Die gesamte Codegröße eines Programms ist nicht beschränkt.

Die Größe des globalen Datenbereichs eines Moduls ist auf 32k Bytes beschränkt. Die gesamte Größe aller Datenbereiche ist nicht beschränkt. Außerdem ist die Göße der dynamisch allokierten Datenbereiche nicht begrenzt.

Der Ergebnistyp von Funktionsprozeduren kann nur 1, 2, 4 oder 8 Bytes lang sein. Darin sind alle elementaren Datentypen enthalten.

Die Obergrenze eines Unterbereichstyps muß kleiner als 2^{15} sein. Der Wertebereich eines Unterbereichstyps darf ebenfalls nicht mehr als 2^{15} Elemente umfassen.

Opake Typen sind auf POINTER–Typen beschränkt.

Enumerationstypen können maximal 256 Elemente haben.

SET Typen können maximal 32 Elemente haben.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Der Debugger

Kapitel 6

Übersicht

Das SPC-MODULA-2 Sprachsystem verfügt über einen symbolischen Debugger. Der Debugger erlaubt, die Datenbestände der geladenen Moduln und der zum Zeitpunkt des Debugger-Aufrufes aktiven Prozeduren zu untersuchen. Alle Informationen werden auf Source-Ebene, also mit den MODULA-2 Variablen- und Prozedurnamen angezeigt. Dadurch ist es sehr schnell möglich zur Fehlerursache vorzustoßen. Die Anzahl der Testläufe kann durch den Debugger verringert werden. Der Debugger ist vollständig in SSWiS integriert. Bei seiner Aktivierung öffnet er 5 Fenster, die in gewohnter Weise mit der Maus bedient werden können. Die verschiedenen Fenster zeigen verschiedene Datenbestände, Programmtext, die Prozeduraufrufkette, etc.

Laufzeitfehler

SPC-MODULA-2 erkennt bestimmte Laufzeitfehler, die den weiteren Ablauf des Programms gefährden. Hierzu zählen

- Zuweisungen von Werten, die außerhalb des zulässigen Wertebereichs liegen,
- arithmetische Über- oder Unterläufe,
- Division durch 0, Verletzung von ARRAY-Indexbereichen,
- Zugriffe über korrumpierte POINTER, etc.

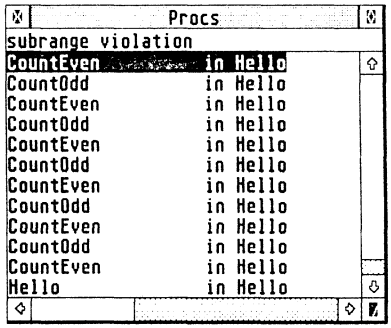
Die Erkennung von arithmetischen Über- und Unterläufen, sowie die Überwachung von ARRAY-Indizes erfordert zusätzlichen Code und kann über Optionen des Compilers ein oder ausgeschaltet werden. Das Laufzeitsystem gibt nach einem erkannten Fehler die Möglichkeit, das Programm abzubrechen, es fortzuführen oder den Debugger zu aktivieren.

Eine weitere Möglichkeit ist, den Debugger an einer bestimmten Stelle aus einem Programm heraus explizit zu aktivieren, indem z.B. einer der oben beschriebenen Fehler absichtlich erzeugt wird.

Nach dem Start des Debuggers sind 3 neue Fenster geöffnet worden. Sie zeigen die Prozeduraufrufkette zum Zeitpunkt des Fehlers, die Fehlerursache, den Quelltext des fehlerhaften Moduls (falls vorhanden) und die lokalen Variablen der fehlerhaften Prozedur.

Procedures-Fenster

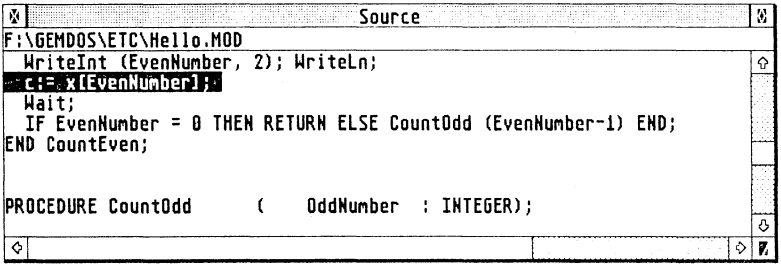
Das Procs Fenster zeigt die zum Zeitpunkt der Aktivierung des Debuggers aktiven Prozeduren und die Moduln, zu denen die Prozeduren gehören. Die unterste Zeile nennt die zuerst aufgerufene Prozedur. Die zweitunterste die von der ersten aufgerufenen, usw. In der ersten Zeile ist die Fehlerursache angegeben (z.B. Index/Range Error).



Source-Fenster

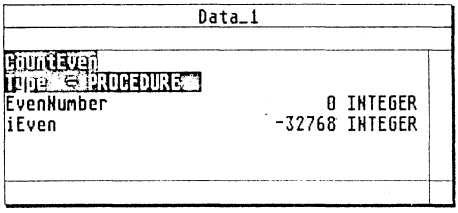
Das Source Fenster zeigt den Quelltext eines ausgewählten Moduls. Zu Beginn wird der Quelltext des fehlerhaften Moduls gezeigt. Dabei ist die Zeile mit dem fehlerverursachenden Statement fett gedruckt. Der Debugger sucht die Quelltexte von Moduln nach dem gleichen Verfahren wie der Compiler, also unter Verwendung der Environment-Variablen PATH<N>. Falls

der Quelltext nicht gefunden wurde, gibt der Debugger die Meldung –no sourcefile– aus.



Data–Fenster

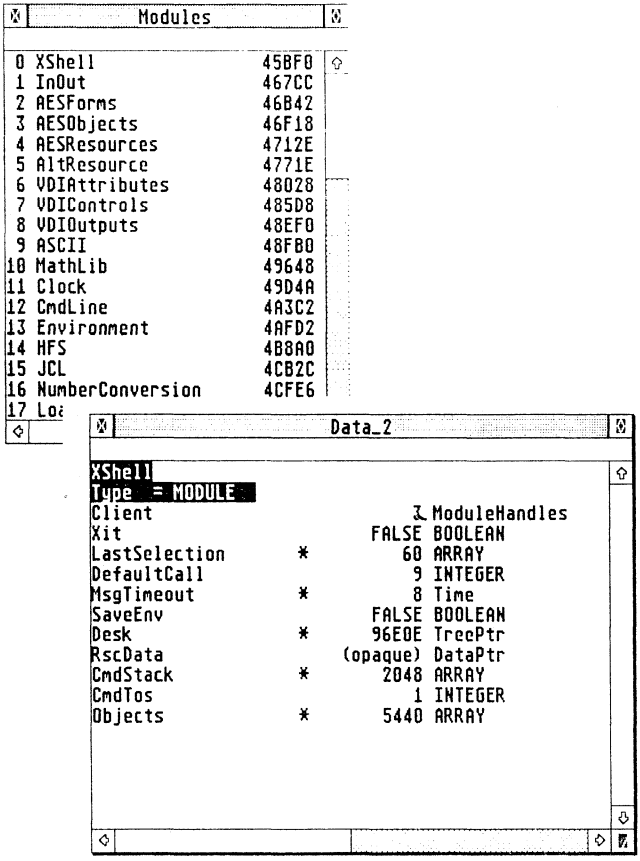
Das Data–1 Fenster enthält die Variablen der im Procs Fenster ausgewählten Prozedur. Zu Beginn ist dies die fehlerhafte Prozedur. Die Variablen werden mit ihrem Namen, ihrem Typ und ihrem Wert ausgegeben. Falls es sich dabei um einen strukturierten Datentyp (ARRAY oder RECORD) oder um einen POINTER handelt, dann ist durch einen Stern (*) angedeutet, daß weitere Details ausgewählt werden können. Dieses erfolgt durch Anklicken der entsprechenden Zeile des Data–1 Fensters. Die derzeit angezeigte Variable wird in der Überschrift des Fensters bezeichnet. Es können auch die lokalen Variablen aller anderen aktivierten Prozeduren gesichtet werden. Dazu muß nur die Prozedur im Procs Fenster angeklickt werden.



Modules-Fenster

Im Modules Fenster werden alle geladenen Moduln angezeigt. Durch Anklicken eines Modulnamens werden dessen globale Variablen im Data-2 Fenster angezeigt. Wird beim Auswählen gleichzeitig die Alternate-Taste gedrückt, dann wird der Quelltext des Moduls im Source Fenster angezeigt.

Das Data-2 Fenster zeigt die globalen Variablen eines ausgewählten Moduls. Bedienung und Anzeigeformat sind beim Data-1 Fenster beschrieben.



Debugger verlassen

Der Debugger wird über das File–Menü durch Selektieren des Eintrages Quit verlassen. Danach erscheint wieder die Meldung des Laufzeitsystems mit der Frage, ob das Programm fortgesetzt oder abgebrochen werden soll.

☞ Man beachte, daß ein Programmmabbruch u.U. dazu führt, daß globale Resourcen nicht wieder freigegeben werden, und es deshalb in der Folge zu Fehlern kommen kann.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Die Utilities

Kapitel 7

Das SPC MODULA–2 Sprachsystem enthält eine Reihe von Utilities, die die Leistungen des Systems abrunden. Hierzu gehören insbesondere

- Filer, er unterstützt die Arbeit mit Dateien und Ordnern.
- Prelink, damit können einzelne Moduln in eine einzige Datei zusammengefaßt werden. Das Laden von Moduln wird dadurch beschleunigt.
- Link, die Utility gestattet es, aus einzelnen .OBM Dateien ein unter GEM ladbares Programm zu erstellen.
- Print, zum Ausdrucken von Textdateien.
- Paths, zum Setzen der Compiler Suchpfade.
- SetEnv, zum Setzen und Abfragen von Environment–Variablen,
- DecObm, zum Dekodieren von .OBM Dateien in Assembler Source–Form.
- Make, zur Automatisierung des Compilationsprozesses.

Die Liste der Utilities wird ständig erweitert. Soweit die Utilities zum Sprachsystem im engeren Sinne gehören (Standard–Utilities), werden registrierte Benutzer im Rahmen des Update–Verfahrens damit versorgt.

Darüberhinaus erhalten unabhängige Anbieter über die SPC MODULA–2 Vertriebswege die Möglichkeit, zusätzliche Utilities anzubieten.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Der Filer

Erfordernisse der Softwareentwicklung

Bei der Softwareentwicklung entstehen viele Dateien. Einige davon werden nur vorübergehend benötigt, andere wiederum sind so wichtig, daß sie innerhalb des Systems mehrmals vorhanden sein sollten, um Datenverlusten vorzubeugen. Ein sinnvolles Arbeiten mit Dateien ist aber nur bei einer klaren Strukturierung der gesamten Dateimenge möglich. So ist es ein Muß jeder Softwareentwicklungsumgebung, Werkzeuge sowohl zum Bearbeiten von Dateien als auch zum Modifizieren des Dateisystems zur Verfügung zu stellen. Wichtig ist dabei, daß diese Arbeiten schnell, effektiv und ohne großen Aufwand seitens des Programmierers vonstatten gehen können.

Die File-Utility wurde unter diesen Gesichtspunkten geschrieben. Dabei wurde besonders darauf geachtet, daß alltägliche Arbeiten elegant und ohne großen Aufwand erledigt werden können.

Konzeption

Der Filer wurde so konzipiert, daß einerseits dem Benutzer eine homogene und leicht zu steuernde Kommandomenge zur Verfügung steht und andererseits alle in der Entwicklungsumgebung notwendigen Dateioperatoren auch komplexerer Natur vorhanden sind. Um dies zu realisieren wurde das Grundprinzip

Grundprinzip

"Erst Auswählen, dann Bearbeiten"

eingeführt, dessen Vorteile Sie im folgenden und beim Arbeiten mit dem Filer sehen werden. Im Gegensatz zur kommandozeilenorientierten Eingabe (zuerst Kommando dann Auswahl) ist das Auswählen unabhängig vom Kommando selbst. Dadurch kann eine nahezu beliebige Menge von verschiedenen zu bearbeitenden Elementen (Dateien, Ordner, Laufwerke) für eine Operation zusammengestellt werden. Ein weiterer Vorteil liegt darin, daß die Operation unabhängig vom Typ der Operanden wird, da Sie diesen selbstständig erkennt. Der Benutzer kann somit seine Anweisungen auf

einer hohen Abstraktionsebene eingeben. Er braucht sich beispielsweise keine Gedanken mehr darüber zu machen, wie sich ein Datei–Löschen–Kommando von einem Ordner–Löschen–Kommando unterscheidet. Er wählt einfach das zu löschende Element – sei es Datei oder Ordner – aus, und die Sache ist erledigt.

Feineinstellung von Kommandos

Ein ebenfalls beachtenswerter Aspekt ist die Feineinstellung von Kommandos. Wenn der Benutzer beispielsweise einen Kopierbefehl gibt, werden normalerweise die Dateien entsprechend kopiert und eventuelle Fehlermeldungen bzw. Erfolgsmeldungen ausgegeben. Der Filer wurde nun so erweitert, daß der Benutzer nicht nur das Was sondern auch das Warum seines Kommandos angeben kann. Ein einfaches Beispiel soll dies aufzeigen:

Beispiel: Backup mit dem Filer

Stellen Sie sich vor, Sie möchten eine Datei aus Sicherheitsgründen auf ein anderes Laufwerk kopieren. Sie bearbeiten diese Datei des öfteren und konsequenterweise ist auch jedesmal ein Sichern auf ihr Backup–Laufwerk notwendig. Das Problem stellt sich Ihnen nun bei mehreren Dateien, sagen wir ca. 20. Da Sie bei dieser Anzahl nicht mehr den Überblick haben können, welche Dateien alt und welche neu sind, bleibt Ihnen nichts anderes übrig als ständig alle alten Backups zu löschen und die Dateien neu auf Ihr Backup–Laufwerk zu kopieren. Dabei können Sie z.B. versehentlich auch Dateien löschen, die sich nur noch auf Ihren Backup–Laufwerk befinden. Fazit: Sie brauchen ein Backup–Programm oder den FILER. Wenn Sie dem Filer sagen: "Kopieren (=was) wegen Backup (=warum)" sind Sie Ihre Sorgen los. Durch Setzen des BackupFlags der Kopieranweisung wird dem Filer klar, warum Sie kopieren wollen und er reagiert entsprechend: Dateien, die noch nicht auf dem Backup–Laufwerk vorhanden sind, werden ganz normal kopiert. Bei Dateien, die bereits vorhanden sind, wird in Abhängigkeit des Schreibdatums der Dateien kopiert: Ist das Original neuer, wird kopiert, ansonsten nicht. Ist

Ihnen das noch nicht genug, dann können Sie zusätzlich zum Backup noch ein Verify verlangen: Jetzt werden Original und Kopie noch byteweise verglichen.

Mit den Kommandoflags kann somit ein hoher Differenzierungsgrad ein und derselben Operation erreicht werden. Dabei wird für den Benutzer die Aktionseingabe nicht schwieriger sondern eher verständlicher und dies trotz höherer Ablaufkomplexität. Ferner ist der Gesamtablauf der Kommandos über Flags beeinflußbar. So wird der Filer den individuellen Wünschen gerecht (Was für den einen die Sicherheit ist, ist für den anderen die Geschwindigkeit, die Information, etc.).

Performance

Es ist unbestreitbar, daß die Verwaltung einer eleganten Benutzeroberfläche mit "mitdenkenden" Operatoren ihren Preis – nämlich Performance-Einbußen – hat. D.h., es gibt sicher ein Super-Hyper-Kopierprogramm, das vielleicht 20–50% schneller kopiert. Aber kann dieses Programm z.B. auch alle Dateien von Laufwerk C:, D: und E: als Backup auf Laufwerk F: kopieren? Selbst wenn es dies kann, wird es beim zweiten Backup die Segel streichen müssen, da der Filer nur noch vielleicht 10% der Dateien real kopieren muss. Weiterhin sollte man bei Vergleichen, die Zeit, die man selbst braucht um komplexere Aktionen zu starten, mitberücksichtigen. Im Übrigen entscheiden Sie selbst über Kommandoflags, ob Sie den Filer zeit-, arbeits- oder sicherheits-optimierend einsetzen wollen.

Eingabearten

Um den individuellen Arbeitsweisen gerecht zu werden, werden – soweit möglich und sinnvoll – beide Eingabearten, d.h., maus- (menue-) und tastaturorientiert, unterstützt. Die Funktionalität der einzelnen Kommandos wird zumeist anhand der mausorientierten Eingabe beschrieben. Die zugeordneten Tastaturkommandos folgen im Anschluß.

Beenden

Wenn Sie den Filer beenden wollen, so wählen Sie den Menuepunkt 'Quit' im 'Command'-Menue an, oder drücken Sie einfach die Taste Q. Der Filer wird damit beendet. Bitte beachten Sie, daß die SSWiS-Umgebung es Ihnen erlaubt, durch Aktivieren des Fensters eines anderen SSWiS-Programms sofort zu diesem wechseln können, ohne den Filer zu verlassen.

Auswählen – Prinzipielles

Getreu dem Motto "Zuerst Auswählen dann Bearbeiten" wird auch das Erstellen der Auswahl zuerst beschrieben. Wenn Sie den Filer aktiviert haben (siehe dazu xShell) meldet er sich, indem zwei Fenster geöffnet werden. Zusätzlich erscheint im Terminal-Fenster die Versionsnummer des Filers. Es gibt Kommandos mit einem und mit zwei Parametern:

Kommandotypen

- Kommandos mit einem Parameter: z.B. Information über "irgendwas" oder Löschen von "irgendwas"
- Kommandos mit zwei Parametern: z.B. Kopieren von "irgendwas" nach "irgendwohin" oder Bewegen von "irgendwas" nach "irgendwohin"

Selection und Destination

Der erste Parameter, das "irgendwas", wird als Selection und der zweite, das "irgendwohin", als Destination bezeichnet. Um Mißverständnissen bei der Auswahl vorzubeugen, erhält jeder Parameter sein eigenes Auswahlfenster (Filer-Selection, Filer-Destination).

Fensteranwahl

Sie sind vermutlich mit Ihren System soweit vertraut, daß Sie wissen, daß Eingaben nur im obersten, d.h. aktiven Fenster möglich sind. Sie können durch Anklicken mit der Maus ein nicht-aktives Fenster nach oben holen und aktivieren. Zwischen den Filer-Fenstern können Sie zusätzlich auch durch Drücken der Leertaste oder Anwählen des Menuepunktes 'Switch' im 'Command'-Menue umschalten. Die Fenster-Infozeile beinhaltet den aktuellen Pfad mit dem Auswahlkriterium -soweit längenmäßig darstellbar-, die Anzahl der Ordner (Fol:) und die Anzahl der Dateien (Fil:) des eingestellten Pfades. Ist kein Laufwerk geöff-

net, enthält sie nur 'Drives' (Laufwerke). Im Innern der Fenster werden die Namen der Dateien angezeigt, die auf dem aktuellen Pfad dem Auswahlkriterium unterliegen. Die Anzeige beginnt immer mit dem Schließsymbol ".." und nachfolgend den Ordnern des Pfades. Die dargestellten Dateien sind nach Namen, Größe oder Erstellungsdatum sortierbar. Dies geschieht durch Anwählen des entsprechenden Menuepunktes im Menue Param oder durch Drücken der Tasten 1 (Namen), 2 (Datum) oder 3 (Größe) der Tastatur.

Ordner und Laufwerke

Das Maus-Handling des Filers ist dem der GemDos-Oberfläche angepaßt. Sie können Laufwerke und Ordner durch Doppelklick öffnen, wie auf dem Desktop. Im Unterschied zum Desktop befinden sich jedoch keine Laufwerkssymbole auf der Arbeitsoberfläche, sondern diese sind als Urpfad in den Auswahlfenstern zu erreichen. Ist ein Laufwerk oder ein Ordner geöffnet worden, erscheint in der ersten Zeile ein Ordnersymbol mit dem Namen "..". Dies repräsentiert das Laufwerks/Ordnerschließsymbol. Ein Anklicken oder Doppelklicken dieser Zeile schließt den entsprechenden Pfad wieder.

Umschalten zwischen Laufwerken

Wenn Sie beispielweise das Laufwerk D: und darauf einige Ordner geöffnet haben, können Sie das aktuelle Laufwerk durch Anklicken aller erscheinenden Ordnerschließsymbole bis zum Urpfad wieder verlassen, um anschließend ein anderes Laufwerk zu öffnen. Dies ist jedoch im allgemeinen der umständlichere Weg. Einfacher geht es, wenn Sie im Menue Drives, den MenuePunkt "Drives" oder sogar gleich das gewünschte Laufwerk anklicken. Noch schneller geht es mit Hilfe der Tastatur: Wenn Sie auf Laufwerk E: umschalten möchten, geben Sie einfach ein E ein! Analog können Sie auf jedes Laufwerk Ihres Systems gelangen. Durch Drücken der 0 auf der Tastatur (nicht auf dem numerischen Block!) erreichen Sie den Urpfad. Die nächste Frage, die sich stellt ist, wie gelange ich schnell zum Ausgangspfad auf D: zurück ? Auch hier ist es mit einem Klick oder einem Tastendruck ge-

schehen. Der Filer speichert für jedes Laufwerk den zuletzt angewählten Pfad und setzt diesen als Laufwerks–Standardpfad ein. Da dies auch lästig sein kann, wird durch wiederholtes Anwählen der gleichen Laufwerkskennung die oberste Ebene des angewählten Laufwerks erreicht. Im Beispiel: Einmal 𝔻 und Sie sind wieder in Ihren auf D: geöffneten Ordnern, nochmal 𝔻 und Sie sind wieder auf der Laufwerksebene von D:.

sichtbare Dateien einstellen

Da der Umfang der Dateien auf einem Laufwerk oder in einem Ordner sehr groß sein kann, ist es oft günstig der Übersichtlichkeit willen nur eine spezielle Auswahl anzusehen. Der Filer bietet hierzu mehrere Möglichkeiten:

i Diese Möglichkeiten bestehen nur für das Selection–Fenster.

Grundsätzlich bietet der Filer die Typen "*.*" (alle Dateien und Ordner) und "{*.*}" (nur Ordner) an. Der Benutzer kann bis zu 6 (0..5) weitere eigene Auswahltypen definieren. Dies geschieht über einen Dialog, der durch Drücken der Taste 𝕋 oder durch Anklicken des Menuepunktes 'Set Types' im Menue 'Special' aktiviert wird. Ist ein Type definiert worden, so kann er ebenso wie die Standardtypen im Menue 'Types' angewählt werden. Auch hier ist eine tastenorientierte Anwahl möglich: Alle Möglichkeiten liegen auf dem numerischen Tastenblock. Die benutzerdefinierten Typen belegen die Tasten 𝟘..𝟝, die "*.*"–Anwahl erfolgt mit * und die "nur Ordner"–Anwahl mit /. Die definierten Typen werden in einer Environment–Variablen gespeichert und sind beim nächsten Aktivieren des Filers wieder vorhanden.

Standardtypen festlegen

Die erste Möglichkeit ist für das Festlegen von Standardtypen wie z.B. "*.MOD", "*.DOC", "PROJEKT.*" geeignet, die nachfolgende zweite Möglichkeit hingegen unterstützt das interaktive Umschalten im

Fenster. Wenn Sie beim Anklicken eines Dateinamens die CONTROL-Taste halten, können Sie die Typanwahl auf zwei Arten ändern: Entweder Klicken Sie auf die Extension (z.B. 'MOD' in 'HELLO.MOD'), dann werden alle Dateien mit dieser Extension angezeigt (entspricht '*.MOD'), oder Sie Klicken auf den Dateinamen selbst (z.B. 'HELLO'), dann werden alle Dateien mit diesem Namen und beliebiger Extension angezeigt (entspricht 'HELLO.*'). Diese Typanwahl können Sie in beiden Fällen durch Halten der CONTROL-Taste und Klicken auf das Schließsymbol wieder rückgängig machen.

Unterschiede: Selection und Destination

Im Destination-Fenster erfolgt das Öffnen, Schließen und Memorieren von eingestellten Pfaden wie im Selection-Fenster. Im Unterschied zu diesem werden jedoch im Destination-Fenster nur Laufwerke und Ordner dargestellt, da eine Datei als Zielumgebungsauswahl nicht sinnvoll ist.

Dateien auswählen

Das Selektieren von Dateien ist einfach. Sie müssen nur im Selection-Fenster in die Zeile klicken, in der der Name der aus- zuwählenden Datei steht. Als Kennzeichnug der Auswahl erscheint dann am rechten Rand des Fenstereintrags ein Pfeil. Sie können die Selektion wieder rückgängig machen, indem Sie diesen Vorgang wiederholen, der Pfeil am rechten Rand verschwindet dann wieder. Möchten Sie weitere Dateien auswählen, tun Sie dies auf dieselbe Weise. Im Gegensatz zum Desktop wird im Filer die vorangegangene Auswahl nicht zurückgesetzt. Somit können Sie auch Dateien aus verschiedenen Ordnern oder sogar von verschiedenen Laufwerken gleichzeitig auswählen.

erweiterte Auswahl

Wenn Sie mehrere der dargestellten Dateien in Ihre Auswahl aufnehmen wollen, geschieht dies analog zum Desktop: Drücken Sie die Maustaste über der obersten (oder untersten) auszuwählenden Datei und ziehen (draggen) Sie die Maus – bei gehaltener Taste – bis zur untersten (bzw. obersten) auszuwählenden Datei.

XOR Logik

Sie waren erfolgreich, wenn danach alle auszuwählenden Dateien mit einem Pfeil gekennzeichnet sind. Dabei ist zu beachten, daß die Auswahl eine XOR-Logik beinhaltet, d.h., wie bei der einfachen Selektion wird eine Datei, die bereits ausgewählt worden war durch obige Drag-Aktion wieder aus der Auswahl entfernt.

Dateien eines Typs auswählen

Möchten Sie alle Dateien (nicht die Ordner!) oder alle Dateien eines Typs des gerade dargestellten Pfades in Ihre Auswahl aufnehmen, erreichen Sie dies am schnellsten durch Drücken der -Taste. Ein Dialog fragt Sie nach dem Auswahltyp und modifiziert Ihre Auswahl entsprechend. Auch hier gilt, wie bereits oben erwähnt, daß bereits ausgewählte Dateien dadurch wieder aus der Auswahl entfernt werden (XOR-Logik).

Ordner, Laufwerke auswählen

Die umfangreichste Auswahlmöglichkeit des Filers besteht in der Selektion von Ordnern und Laufwerken. Wir wollen uns in der Beschreibung auf Ordner konzentrieren und logische Laufwerke einfach auch als (sehr große) Ordner betrachten; das nachfolgend Gesagte gilt für diese ebenso. Die Selektion eines Ordners erfolgt genauso wie bereits für Dateien beschrieben, also durch Anklicken mit der Maus, oder durch Draggen über mehrere auszuwählende Elemente im Fenster. Wichtig ist jedoch die unterschiedliche Semantik der Auswahl:

- Wenn Sie eine Datei auswählen, ist diese und nur diese gemeint. Wenn Sie einen Ordner auswählen, so ist der Ordner und der gesamte Inhalt mit allen Dateien und Unterordnern gemeint, **falls diese dem Ordnerauswahlkriterium (Foldertype) genügen!**

Ordner-Auswahl-Kriterium (OAK)

Das Ordnerauswahlkriterium ist standardmäßig auf '*.*' (alles) eingestellt. Wählen Sie nun einen Ordner an, sind damit *alle* Dateien und Ordner, die sich darin verbergen gemeint. In der Auflistung der Auswahl erkennen Sie dies am '*.*' hinter dem Ordnernamen. Sie

können das Ordnerauswahlkriterium (OAK) genauso wie die Datei-Anzeigetypen mit 'Set Types' im 'Special'-Menue ändern. Das aktuelle OAK erkennen Sie im Menue 'Type' zwischen den geschweiften Klammern (z.B. {*.*}). Wenn Sie das OAK beispielsweise auf '*.MOD' ändern, bedeutet dies: Es ist mit Ihrer Selektion der gesamte Dateibaum mit allen Ordnern gemeint. Aber Dateien sind nur dann betroffen, wenn Sie das OAK, d.h., im Beispiel '*.MOD' erfüllen. Sie können also mit dieser Auswahl alle '*.MOD'-Dateien und Ihre angelegte Ordnerstruktur bearbeiten.

Beispiel

Ein konkretes Beispiel: Laufwerk D: enthält die Ordner D:\SPC und D:\SOURCE. D:\Source enthalte 4 verschiedene Projektordner PROJ1,.., PROJ4. Mit OAK = '*.MOD' und Selektion von D:\SOURCE haben Sie alle MOD-Dateien, die in SOURCE oder einem beliebigem Unterordner von SOURCE stehen ausgewählt. Wenn Sie nun F:\Backup als Kopierziel angeben, werden alle MOD-Dateien und die gesamte Ordnerstruktur von SOURCE in F:\BACKUP kopiert. Genauso können Sie mit allen 'DEF'-Dateien verfahren. Sie können auch als OAK '*.BAK' angeben und alle BAK-Dateien auf einmal löschen. Vergewissern Sie sich aber vor dem Löschen, daß Sie tatsächlich nur die BAK-Dateien selektiert haben, indem Sie sich die Selektion auflisten lassen! Es sollte dort dann D:\SOURCE***.BAK** stehen.

Zwei wichtige Bemerkungen:

Behandlung von Ordnerstrukturen

☞ Ordnerstrukturen werden nur dann geändert, wenn es sinnvoll erscheint. Wird beispielsweise ein Ordner durch ein Löschkommando komplett geleert, so wird auch der Ordner gelöscht. Beim Kopieren von Ordnern wird die Ordnerstruktur auf dem Kopierziel neu eingerichtet, vorausgesetzt sie ist nicht bereits vorhanden.

Ändern des OAK

☞ Das Ändern des OAK hat keinen Einfluß auf vorherige Selektionen! D.h., Wenn Sie einen Ordner mit '*.*' ausgewählt haben und anschließend das OAK

ändern, ist für diesen Ordner immer noch '*.*' gültig! Dies hat den Vorteil, daß Sie diverse Ordner mit jeweils verschiedenen OAKs selektieren können. Allerdings ist es nicht möglich, einen Ordner gleichzeitig mehrfach mit verschiedenen OAK'en auszuwählen. Dies wurde der Übersichtlichkeit halber unterbunden.

Auswahl anzeigen und rücksetzen

Da die ausgewählten Dateien über mehrere Ordner oder Laufwerke verteilt sein können ist es wichtig, sich die gesamte Auswahl auflisten lassen zu können. Dies geschieht mit dem Menuepunkt 'Show Sel' im Command-Menue. Genauso können Sie ihre gesamte Auswahl wieder rückgängig machen, indem Sie 'Clear Sel' im Command-Menue anwählen. Wenn Sie tastaturorientiert arbeiten möchten, finden Sie diese Funktionen auf der RETURN- (Show Sel) bzw. der DEL-Taste (Clear Sel). Es ist sicher sinnvoll, sich vor größeren Aktionen (z.B. Löschen einer Harddisk-Partition o.ä.) die gesamte Auswahl auflisten zu lassen, insbesonders dann, wenn Sicherheitsabfragen ausgeschaltet werden!

Automatische Auswahllogik

Durch die Vielzahl der Möglichkeiten kann es passieren, daß eine Datei mehrmals ausgewählt wurde (z.B. Ordner und Datei im Ordner). Der Filer besitzt eine zuschaltbare Logik, die die Auswahlliste auf solche Mehrfachauswahlen überprüft und entsprechend verändert. Dabei wird davon ausgegangen, daß bei Mehrdeutigkeiten die spätere Auswahl die maßgebliche ist: Wenn Sie eine Datei in einem Ordner auswählen und anschließend den Ordner selbst, wird die Auswahl der Datei ignoriert. Bei umgekehrter Auswahlreihenfolge wird der Ordner ignoriert. Diese Möglichkeit ist optional über den Menuepunkt 'Set Flags' im 'Special'-Menue an- bzw. abschaltbar.

Ziel auswählen

Für Aktionen wie Kopieren und Verschieben benötigen Sie als zweite Angabe das Ziel (Destination). Um das Ziel für diese Aktionen auszuwählen, müssen Sie zunächst zur Zielauswahl umschalten. Dies geschieht, indem Sie das Destination–Fenster aktivieren. Die Auswahl erfolgt wie im Selection–Fenster, jedoch ist nur genau ein Laufwerk oder ein Ordner als Zielangabe möglich. Daher wird auch durch Auswahl eines Ziel eine eventuelle vorherige Auswahl aufgehoben. Der ausgewählte Ordner oder das ausgewählte Laufwerk wird durch einen Pfeil nach unten gekennzeichnet.

Ziel mit der Tastatur auswählen

Auch die Auswahl kann tastaturorientiert erfolgen. Die gesamte Funktionalität ist auf dem Cursor–Tastenblock untergebracht. Mit der Taste CtrlHome können Sie die tastenorientierte Auswahl an– bzw. abschalten. Wenn Sie die Taste gedrückt haben, erscheint in jedem der beiden Filer–Fenster ein Eintrag invertiert. Der invertierte Eintrag ist die aktuelle Position Ihres Selektionszeigers. Sie können mit Hilfe der Cursortasten den Selektionszeiger auf jede Datei positionieren: Mit der Cursor–Up– und der Cursor–Down–Taste können Sie durch das aktuell dargestellte Inhaltsverzeichnis scrollen. Dabei wird das Fenster automatisch so mitgescrollt, daß der Selektionszeiger immer sichtbar bleibt. Mit der Cursor–Right– oder der Cursor–Left–Taste können Sie Ordner öffnen, bzw durch Anwahl von ".." schließen. Wenn Sie eine Datei oder einen Ordner in Ihre Auswahl aufnehmen wollen, drücken Sie wenn der Selektionszeiger auf dem gewünschten Element steht die Insert–Taste.

Kommandos

Der Filer bietet folgende Grundkommandos:

- Copy F1 Kopieren
- Move F2 Verschieben
- Delete F3 Löschen
- Rename F4 Umbenennen
- Info F5 Informationen
- Tree F6 Datei–Inhaltsverzeichnis
- Compare F7 Vergleichen von Dateien
- Search F8 Suchen von Dateien

Zur Funktionalität der einzelnen Kommandos lesen Sie bitte den entsprechenden Abschnitt. Die Kommandos werden durch Anwählen des Kommandonamens im 'Command'–Menue bzw. durch Drücken der angegebenen Funktionstasten aktiviert. Zunächst wollen wir uns mit den Steuerungsmöglichkeiten beschäftigen.

Kommando–Flags

Um die einzelnen Kommandos an individuelle Gegebenheiten anzupassen, können für jedes Kommando separat Kommando–Flags angegeben werden. Einige der nachfolgenden Flags sind bei jedem Kommando einzustellen, andere sind nur für ein spezielles Kommando vorhanden. Die Flags lassen sich mit dem Menuepunkt 'Set Flags' im 'Special'–Menue setzen.

Die Kommando–Flags haben folgende Bedeutung:

Verbose–Flag

Verbos (Geschwätzig), d.h., der Filer gibt bei den entsprechenden Aktionen des Kommandos Nachricht darüber, was gerade getan wird. Beim 'Info' hat dieses Flag eine gesonderte Bedeutung bei der Ordneranwahl: Es werden dann nicht nur globale Informationen über den Ordner bzw. das Laufwerk ausgegeben, sondern auch über die im Ordner enthaltenen Dateien. Die Ausgabe der Informationen erfolgt entweder ins Terminal–Fenster oder auf eine angegebene Datei. Der Ablauf der Aktion wird dadurch verlangsamt.

Query–Flag

Query (Frag mich). Ist diese Flag gesetzt, wird vom Benutzer vor dem Ausführen einer Aktion nochmals eine Bestätigung verlangt. Wichtig ist hierbei, daß der Benutzer im Bestätigungs–Dialog die meisten Flags für die laufende Aktion modifizieren kann. Diese Flag–Änderungen betreffen aber nur die gerade laufende Aktion und nicht die globale Einstellung der Flags! Darum sollte z.B. bei Lösch–Aktionen das Query–Flag nie global abgeschaltet werden. Wenn Sie sicher sind, daß die laufende Löschaktion korrekt ist, schalten Sie einfach im Bestätigungs–Dialog das Query–Flag aus. Sie werden dann für diese Auswahl nicht mehr belästigt. Bei der nächsten Auswahl ist das Query–Flag wieder vorhanden und schützt Sie so vor ungewollten Aktionen. Der Ablauf der gesamten Aktion wird durch die Bestätigungsanforderungen langsamer aber auch wesentlich sicherer.

Replace–Flag

Replace (Ersetzen). Dieses Flag ist nur beim Kopieren, Verschieben und Umbenennen vorhanden. Ist es gesetzt, werden Namenskonflikte mit existierenden Dateien auf dem Zielpfad ignoriert, d.h., eventuell vorhandene Dateien gleichen Namens werden überschrieben. Ist das Flag nicht gesetzt, gibt der Filer eine Warnmeldung aus und Sie können über einen Dialog entscheiden, wie auf den Namenskonflikt reagiert werden soll.

Buffer–Flag

Buffer (Puffern). Dieses Flag ist nur für Kopiervorgänge relevant. Ist es gesetzt, so wird versucht, zuerst soviel einzulesen wie Speicherplatz vorhanden ist und erst anschließend wird geschrieben. Dieses Flag optimiert damit die Ein/Ausgabeoperationen. Insbesondere wenn Sie von A: nach B: kopieren möchten und das Laufwerk B: nur von Ihrem Rechner emuliert wird, können Sie damit das Disketten–Jonglieren wesentlich reduzieren. Der Ablauf der Gesamtaktion wird schneller, insbesondere dann, wenn Diskettenlaufwerke angesprochen werden. Der allokierte Speicher wird im Anschluß freigegeben, ist innerhalb des SPC–Systems

wieder verfügbar (aber nicht für Accessories!).

Die nachfolgenden Flags sind nur für das Kopieren zuständig und können auch nur global eingestellt werden.

Backup–Flag

Backup (Sicherungskopie). Ist dieses Flag gesetzt, wird der Kopiervorgang als Sicherungsvorgang bewertet, d.h., daß Namenskonflikte durchaus auftreten dürfen. Tritt er auf, so wird angenommen, daß beide Dateien verschiedene Versionen der gleichen Datei repräsentieren. Ist nun die zu kopierende Datei neueren Datums, wird die vorhandene Zieldatei überschrieben. Ist dies nicht der Fall, ist die Sicherungskopie noch aktuell und der Kopiervorgang wird unterbunden.

☞ Dieses Flag kann nur dann korrekt arbeiten, wenn Sie Ihre Systemuhr **immer** setzen.

Ist dies der Fall, können Sie damit Sicherungskopien automatisch und zeitoptimiert erstellen.

Verify–Flag

Verify (Vergewissern). Das Verify–Flag testet erstellte Kopien auf Konsistenz mit dem Original, d.h., Original und Kopie werden byteweise verglichen und eventuelle Unterschiede gemeldet. Dieses Flag ist sinnvoll, wenn Ihnen Ihre Datenkonsistenz sehr wichtig ist. Durch das Einlesen und Vergleichen von Original und Kopie wird die gesamte Aktion natürlich erheblich langsamer die Datensicherheit dafür wesentlich größer.

Check–Flag

Check (Prüfen). Das Check–Flag prüft bevor die eigentliche Aktion anläuft, ob genügend Platz auf dem Speichermedium vorhanden ist, um alle ausgewählten Dateien und Ordner darauf unterzubringen. Dabei wird der Verwaltungsverschnitt mitberücksichtigt. Die Prüfaktion kostet natürlich Zeit. Wird sie abgeschaltet, wird versucht, alles zu kopieren. Eventuell riskieren Sie damit Fehler beim Kopiervorgang. Faustregel: Bei platzkritischen Speichermedien wie Disketten und relativ vollen Harddiskpartitions das Flag setzen, sonst abschalten.

Abbrechen von Kommandos

Wenn Sie das Query-Flag nicht gesetzt haben, arbeitet der Filer das eingegebene Kommando für Ihre gesamte Auswahl ab. Nun kann Ihen aber ein Mißgeschick passiert sein und Sie müssen das Kommando abbrechen. Hierfür können Sie die Undo-Taste benutzten. Der Filer meldet sich mit einem Dialog, in dem Sie die aktuelle Aktion anhalten oder abbrechen können. Der Filer erkennt den Interrupt an vordefinierten Aufsetzpunkten und hält entweder die Aktion an oder bricht sie ab. Dabei werden begonnene Dateioperationen vorher zu Ende geführt (wurde z.B. bereits begonnen eine Kopie zu schreiben, wird diese auch ganz geschrieben). Die Interrupt-Möglichkeit kann auch mit dem Menuepunkt 'Set Flags' an- bzw. abgeschaltet werden.

Wurde eine Aktion angehalten, kann Sie durch nochmaliges Drücken der Undo-Taste wieder fortgesetzt oder abgebrochen werden. Das Anhalten kann dazu dienen andere xShell-Programme zu aktivieren, ohne durch Filer-Aktivitäten Konflikte (z.B Dateizugriffskonflikte) zu provozieren.

Delete

Das Delete-Kommando dient zum Löschen von Dateien und Ordnern. Es werden nur die Directory-Einträge der Dateien, bzw. Ordner gelöscht und der zugeordnete Platz auf dem Speichermedium freigegeben. Um Ihre Datei physikalisch zu löschen, müssen Sie Ihr Speichermedium formatieren. Ordner werden nur dann gelöscht, wenn diese vollständig geleert sind. Zum Beispiel löscht ein Ordner-Delete-Kommando mit Kriterium '*.BAK' diesen nur, wenn in ihm nur 'BAK'-Dateien enthalten sind.

Flags: Verbos [an], Query [an]

Copy

Das Copy-Kommando kopiert Ihre gesamte Auswahl auf den eingestellten Zielpfad (Destination-Auswahl). Es läßt sich durch die Kommandoflags vielseitig einsetzen.

Flags: Verbos [an], Query [an], Replace [aus], Backup [aus], Verify [aus], Check [aus]

Move

Das Move-Kommando verschiebt Dateien und Ordner innerhalb eines logischen Laufwerks. Da nur die Directory-Einträge modifiziert werden und die Dateien physikalisch nicht verschoben werden, ist es im Vergleich zu einem Kopiervorgang mit anschließendem Löschen der Originale wesentlich effizienter. Ein weiterer Vorteil ist, daß - abgesehen von Verwaltungsverschnitt für verschobene Ordnerstrukturen - kein Platz auf dem Speichermedium benötigt wird. So lassen sich auch relativ ausgelastete Speichermedien noch leicht strukturieren.

Flags: Verbos [an], Query [an], Replace [aus]

Rename

Mit dem Rename-Kommando können Sie Dateien und Ordner umbennen. Der neue Name wird über einen Bestätigungsdialog abgefragt. Da das Atari-Betriebsystem (TOS) in der derzeit vorliegenden Version ein reguläres Ordnerumbenennen nicht erlaubt, wird diese Aktion durch Anlegen eines neuen Ordners, Verschieben der Dateien in den neuen Ordner und Löschen des alten Ordners simuliert. Sobald die neue TOS-Version allgemein freigegeben ist, wird auch der Filer das direkte Ordnerumbennen unterstützen.

Flags: Verbos [an], Query [an], Replace [aus]

Info

Mit dem Info-Kommando können Sie sich detaillierte Informationen über Dateien, Ordner oder Laufwerke ausgeben lassen. Die Informationen werden ins Terminal-Fenster bzw. auf Datei ausgegeben und umfassen: Name, Größe, Zugriffsmodus, Erstellungsdatum, Anzahl Unterordner und Dateien (für Ordner u. Laufwerke) sowie freier Speicherplatz (Laufwerke). Bei Ordnern und Laufwerken werden nur die globalen Informationen ausgegeben. Sollen hier auch Einzelheiten über die im Ordner/Laufwerk enthaltenen Elemente ausgegeben werden, ist das Verbos-Flag zu setzen.

Flags: Verbos [aus], Query [an]

Tree

Das Tree-Kommando listet Ihnen das Datei-Inhaltsverzeichnis Ihrer ausgewählten Elemente. Auch hier können Sie durch das Ordnerauswahlkriterium (OAK) das Inhaltsverzeichnis einschränken. Es werden die kompletten Pfadnamen der Dateien auf das Terminal-Fenster bzw. Datei ausgegeben.

Flags: Verbos [an], Query [an]

Compare

Das Compare-Kommando untersucht zwei Dateien auf Identität. Die zu vergleichenden Dateien müssen im Selection-Fenster ausgewählt werden.

Flags: Verbos [an], Query [aus]

Search

Mit dem Search-Kommando können Sie Dateien auf Ihren Laufwerken suchen lassen. Dies ist insbesondere bei verschachtelten Ordnerstrukturen sehr nützlich. Die zu suchende Datei(en) wird durch einen Dialog abge-

fragt. Der eingegebene Name muß folgender Syntax genügen:

<EinzugebenderName> ::= <Laufwerke> ':' <Dateiname>

wobei
<Laufwerke> ::= '*'|<LwKennung>|<LwListe>
<LwKennung> ::= 'A'| ... |'P'
<LwListe> ::= <LwKennung> '-' <LwKennung>
<Dateiname> ::= GEMDOS-Dateiname

Zu beachten ist, daß

- nur existierende Laufwerke angegeben werden sollten.
- durch Angabe der LwListe mehrere nacheinanderfolgende Laufwerke durchsucht werden können.
- durch Laufwerksangabe '*' alle in Ihrem System vorhandenen Laufwerke durchsucht werden.
- der Dateiname der GEMDOS-Konvention entspricht und die ent- sprechenden Wildcards erlaubt sind.
- kein Ordnerpfad angegeben werden darf.

Beispiele

***:HELLO.MOD** : Suche alle HELLOD.MOD Dateien im System.

C-F:*.MOD : Suche alle Dateien mit Extension 'MOD' auf den Laufwerken C:, D:, E: und F:

A:HI*.* : Suche alle Dateien auf Laufwerk A:, deren Namen mit 'HI' beginnen.

Sie erhalten als Ergebnis eine Auflistung der kompletten Pfadnamen der gefundenen Dateien sowie deren Gesamtanzahl. Die Ausgabe erfolgt ins Terminal-Fenster oder auf Datei.

Ausgabe umlenken

Die Ausgabe der verschiedenen Kommandos kann statt in das Terminal–Fensters auch auf eine Datei erfolgen. Dazu wählen Sie im 'Parameter'–Menue den Menuepunkt 'File' an und geben den Namen der gewünschten Ausgabedatei an. Die nachfolgenden Ausgabe der Kommandos werden nun in diese Datei geschrieben. Wenn Sie im gleichen Menue den Punkt 'Terminal' anwählen, wird die Ausgabe wieder ins Terminal–Fenster erfolgen. Die Ausgabedatei ist dann geschlossen und kann beispielsweise mit dem Editor angesehen oder mit dem Print–Utility ausgedruckt werden.

Die nachfolgenden Möglichkeiten können im Menue 'Special' angewählt werden.

Disketten formatieren

Mit 'FormatDrive' können Disketten im Standardformat formatiert werden. Es erscheint eine Dialogbox zur Auswahl des Laufwerks (A oder B). Anschließend kann zwischen einseitigem und doppelseitigem Formatieren gewählt werden. Mit dem Cancel–Button kann das Kommando in der jeweiligen Dialogbox abgebrochen werden.

☞ Vor dem Formatieren vergewissern Sie sich bitte immer, welche Diskette im Laufwerk ist!

neuen Ordner anlegen

Mit 'New Folder' können Sie neue Ordner erzeugen. Im Dialog erscheint der aktuelle Pfad im Editierfeld, so daß Sie nur den Namen Ihres neuen Ordners eingeben müssen. Möchten Sie den neuen Ordner nicht in der aktuellen Umgebung anlegen, so löschen Sie den Pfadnamen und geben Ihren gewünschten Pfad an. Der Ordner wird erzeugt, wenn Name und Pfadname korrekt angegeben wurden.

Pfad explizit ändern

Mit 'ChangePath' kann die Umgebung des aktuellen Laufwerks explizit neu eingestellt werden. Dies ist dann sinnvoll, wenn auf einem Laufwerk von einer tiefen Ordnerschachtelung in eine andere tiefe Ordnerschachtelung gewechselt werden soll.

Beispiel: Von D:\ORDA1\ORDB1\ORDC\ORDD nach D:\ORDA2\ORDB2\ORDX. Ansonsten ist es schneller, die Ordner zu schließen und den neuen Pfad zu öffnen.

Der Linker

Der Linker hat die Aufgabe, aus den einzelnen übersetzten Moduln ein unter GEM ablauffähiges Programm mit der Endung .PRG zu erstellen. Dazu geht er vom Hauptmodul aus und sucht alle unmittelbar und mittelbar benutzten Moduln zusammen, verbindet sie nach bestimmten Regeln untereinander und legt sie in einer Ausgabedatei ab. Letztere hat den gleichen Namensstamm wie der Hauptmodul, jedoch die Endung .PRG.

Daneben wird ein Listing erzeugt, das über die eingebundenen Moduln Aufschluß gibt. Falls der Vorgang nicht erfolgreich war, muß das Listing inspiziert werden, um die Fehlerursache festzustellen. Das Listing hat den Namen LINK.LST und liegt auf dem momentanen Arbeits-Directory.

Starten

Der Linker wird über eine Standard-Kommandozeile aufgerufen und parametriert. Sie hat die Form:
link <Modulname> [-v] [-s<Stack>]

Argumente

Der <Modulname> ist der Name des Hauptmoduls. Er darf eine Dateiendung (z.B. .MOD oder .OBM) enthalten. Es wird in jedem Fall .OBM impliziert und der Modul wird, wie alle weiteren auch, auf den Compiler-Suchpfaden gesucht.

Die Option -v (verbose) veranlaßt den Linker, den Fortgang des Linkens durch Angabe der Modul- bzw. Dateinamen auf dem Terminal zu protokollieren.

Stacksize

Der Linker bestimmt die Stackgröße des fertigen Programms. Diese ist normalerweise 20000 Bytes. Falls ein anderer Wert gewünscht wird, kann er als Zahl direkt hinter der -s Option angegeben werden.

Parametrieren

Der Linker wird interaktiv über die xShell aufgerufen. Der Dateiname wird dann nach dem gewohnten Verfahren von der xShell beigesteuert. Die Optionen können durch Parametrieren des Link-Icons der xShell eingestellt werden.

Pass 1

Das Linken läuft in zwei Durchgängen (Pässen) ab. Im ersten Durchgang werden die Moduln auf den Compiler-Suchpfaden gesucht und registriert. Dabei wird geprüft, ob die Modulschlüssel zueinander passen. Da jeder Modul weitere Moduln importieren kann, läuft der Vorgang solange, bis alle benötigten Moduln registriert sind.

Pass 2

Falls keine Fehler aufgetreten sind, wird der zweite Durchgang gestartet. Durch ihn werden alle registrierten Moduln in eine einzige Ausgabedatei kopiert. Dabei werden gleichzeitig Änderungen an den Moduln durchgeführt, um sie untereinander geeignet zu verbinden. Schließlich wird der Ausgabedatei ein Prolog vorangestellt, der die notwendigsten Initialisierungen durchführt.

Ausgabedatei

Falls auch hier keine Fehler aufgetreten sind, liegt das fertige Programm im gleichen Ordner, in dem auch der Hauptmodul gefunden wurde, allerdings mit der Endung .PRG.

Fehler können mehrere Ursachen haben:

bad syntax

IO-Fehler können durch defekte Eingabedateien verursacht werden. In diesem Fall meldet der Linker 'bad syntax' oder 'io errors occurred'.

io errors occurred

Beim Erzeugen der Ausgabedatei können Probleme dadurch auftreten, dass nicht genügend Platz auf der Diskette vorhanden ist. Dies wird ebenfalls durch die Meldung 'io errors occurred' angezeigt.

module not found

Falls die Compiler-Suchpfade nicht richtig eingestellt sind, können Moduln u.U. nicht gefunden werden. Der Linker meldet dann 'module not found'.

illegal module ke

Der häufigste Fehler wird durch inkonsistente Modulschlüssel verursacht. Der Linker meldet dann 'illegal module key'. Die genaue Ursache ist nur aus dem Listing zu ersehen. Dort kann mit dem Editor nach dem Wort 'illegal' gesucht werden. Da der Fehler

immer einen Konflikt zwischen dem von einem importierenden Modul erwarteten und dem in einem Modul enthaltenen Schlüssel darstellt, kann der Linker nicht bestimmen, welcher Modul fehlerhaft ist. Wenn man aber nach weiteren Fehlerstellen sucht, wird man sehr schnell erkennen, welcher Modul fehlerhaft ist. Dieser muß dan noch einmal übersetzt werden.

Listing

Der Aufbau des Listings wird unten ausschnittsweise gezeigt. Für jeden eingebundenen Modul wird ein Eintrag erzeugt, der Aufschluß gibt über

- die Datei, die den Modul enthält
- den Modulschlüssel
- die Längen des Code-, Daten- und Konstantenbereichs
- die Zahl der exportierten Prozeduren (inklusive der Modulinitialisierung)
- die relative Adresse des Modul-Deskriptors (siehe SYSTEM.DEF)
- die relative Address des Code-Bereichs
- die relative Address des Datenbereichs (StaticBase)
- die importierten Moduln und deren erwartete Modulschlüssel

```
.....
import  System 000000000000

include module CmdLine
file E:\gemdos\SYSLIB.obj\CmdLine.OBM
key 0CC302DDC4D1
code length 1132
data length 426
const length 2
# exported procedures 8
descriptor start DED4
frame start DEF2
StaticBase E508
import Environment 0CC302DDCFFD
import Strings 0CC302DD1478
```

```
import System 000000000000
import GemDos 0CC302E01B8A
import System 000000000000
include module Environment
.....
```

Initialisierungs-reihenfolge

Das Listing wird im Erfolgsfalle abgeschlossen durch eine Liste der Moduln in der Reihenfolge ihrer Initialisierung. Normalerweise wird jeder importierte Modul vor allen ihn importierenden Moduln initialisiert. Dadurch wird gewährleistet, daß auch in der Modulinitialisierung ein importierter Modul schon operabel (d.h. selbst initialisiert) ist.

Zyklen

Wenn aber ein Modul A einen Modul B importiert, und B importiert seinerseits A, dann ist ein sogenannter Zyklus aufgetreten und die Initialisierungsreihenfolge ist undefiniert. Sie hängt dann einzig und allein davon ab, welcher Modul zuerst in einer Importliste genannt wurde. Durch die Hinschreibung der Import-Liste (z.B. des Hauptprogramms) kann also in solchen Fällen die Initialisierungsreihenfolge beeinflußt werden.

Übergang von dynamic linking zu static linking

Da die Anwendung schon unter der xShell getestet wurde, macht der Übergang zu einem eigenständigen Programm normalerweise keine Probleme. Allerdings muß der Hauptprogramm-Modul nun das ganze Laufzeitsystem initialisieren, eine Aufgabe, die bis dahin die xShell übernommen hatte. Durch das o.g. Initialisierungsschema erfolgt dies implizit ohne daß besondere Maßnahmen getroffen werden.

Der Prelinker

Der Prelinker hat die Aufgabe, mehrere .OBM Dateien zu einer einzigen Datei, einer Art Bibliothek zusammenzufassen. Dazu wird von einem Modul, dem Leitmodul, ausgegangen und alle importierten Moduln in die Ausgabedatei kopiert, die davon nicht explizit ausgeschlossen wurden. Alle notwendigen Angaben werden durch eine Steuerdatei gegeben.

Dadurch, daß eine Menge von Moduln in einer einzigen Datei zusammengefaßt (vorgebunden) sind, wird der Ladevorgang beschleunigt. Alle Werkzeuge des Sprachsystems sind in dieser Weise vorgebunden. Bei größeren Projekten bietet es sich für alle Moduln an, die gerade nicht bearbeitet werden.

Starten

Der Prelinker wird über eine Standard-Kommandozeile aufgerufen und parametriert. Sie hat die Form:
prelink <Dateiname> [-v]

Argumente

Der <Dateiname> ist der Name der Steuerdatei. Er hat normalerweise die Endung .LCM) enthalten.

Die Option -v (verbose) veranlaßt den Prelinker, den Fortgang des Linkens durch Angabe der eingeschlossenen Dateinamen auf dem Terminal zu protokollieren.

Aufbau der Steuerdatei

Die Steuerdatei muß folgenden Aufbau haben :

in der ersten Zeile steht der Name des Leitmoduls (z.B. Main)

in der zweiten Zeile steht der Pfadname der Ausgabedatei (z.B. \USER\LNK\MAIN.OBM)

in allen weiteren Zeilen stehen die Namen der Moduln, die vom Binden auszuschließen sind (z.B. InOut).

```
Edit
e:\gemdos\misc\edit.obm
inout
filesystem
environment
```

Protokoll

Während des Bindens zeigt der Prelinker den Namen des Hauptmoduls mit einem vorangestellten * an. Die Ausgabedatei wird mit + markiert. Alle auszuschließenden Moduln werden mit einem % kenntlich gemacht und die importierten Moduln zeigt der Prelinker durch ein vorangestelltes – an.

Starten eines vorgebundenen Moduls

Ein gebundener Modul wird wie jeder andere Programm–Modul gestartet. Man beachte, daß die Ausgabedatei den gleichen Namen wie der zu startende Modul haben muß und auf einem der Compiler–Suchpfade liegen muß.

Make

Make ist eine von UNIX bekannte Utility, die bei der Entwicklung größerer Systeme unterstützt. Make ist für den fortgeschrittenen Anwender und für große Projekte bestimmt. Die Aufgabe von Make ist es, festzustellen, welche abhängigen Dateien eines Softwareprojektes aufgrund von Modifikationen anderer Dateien nicht mehr aktuell sind, und diese dann (meist durch Compilation) neu zu erzeugen. Dazu benötigt Make eine Beschreibung (das Makefile) der gegenseitigen Abhängigkeiten von Dateien, und der Kommandos, mit denen eine abhängige Datei neu erzeugt werden kann. Darüberhinaus ist Make natürlich darauf angewiesen, daß die Uhr des Rechners immer richtig gestellt ist.

Begriffe

Ziel, Target

Prerequisiten

Kommandos

Zum Verständnis von Make müssen zunächst einige Begriffe erklärt werden. Eine abhängige Datei (Ziel, Target) ist eine Datei, die immer dann nicht mehr aktuell ist, wenn sie älter als eine von mehreren Bezugsdateien (Prerequisiten) ist. Dabei werden die Zeitmarkierungen des Filesystems ausgewertet. Tritt dieser Fall ein, dann wird eine Kommandosequenz ausgeführt, welche i.a. die abhängige Datei neu erstellt. Die Kommandosequenz kann sich implizit aus dem Typ der abhängigen Datei ergeben, oder sie kann explizit angegeben sein.

Dieser Mechanismus ist sehr allgemein, und kann mit etwas Übung auch für andere Zwecke eingesetzt werden. Im folgenden wird jedoch beschrieben, wie Make zur Steuerung des Übersetzungsvorgangs eingesetzt wird.

Es sei für die weiteren Erklärungen vorausgesetzt, daß ein Makefile bestehe, welches die Abhängigkeiten in einem Softwaresystem korrekt beschreibt. Es seien verschiedene Dateien editiert worden, und es soll nun Make dazu verwendet werden, die betroffenen Moduln neu zu übersetzen. Wegen der Importe von Schnittstellen in andere Moduln kann die Änderung eines Definitions-Moduls bekanntermaßen weitere Übersetzungen nach sich ziehen.

Starten

Make wird über eine Standard–Kommandozeile gestartet und parametriert. Die Kommandozeile hat die Syntax:
make <Makefile> [-v] [-i] [-n] [-t <Target>]

Arumente

<Makefile> ist der Name des Makefiles, das die Beschreibung des zu bearbeitenden Programmsystems beinhaltet

Optionen

Die –v Option hat wie üblich die Bedeutung 'Verbose' und veranlaßt Make, Meldungen über den Fortgang der Arbeit auf dem Terminal auszugeben.

Die –i Option bestimmt, daß Fehler, die bei der Ausführung von Kommandos auftreten, ignoriert werden sollen.

Die Option –n kann zum Testen eines Makefiles benutzt werden. Make wird dann nur die Kommandozeilen auf dem Terminal ausgeben, ohne die Kommandos tatsächlich zu starten.

Das Ziel des Make–Prozesses kann gleich als Option hinter –t angegeben werden. Falls dies nicht der Fall ist, wird ein Formular ausgegeben, in dem angegeben werden kann, daß nur ein bestimmtes Ziel neu erzeugt werden soll (Default ist: alle Ziele neu erzeugen).

Ablauf

Make wird dann ausgehend von dem oder den Zielen prüfen, ob alle Prerequisiten älter sind. Falls die Prerequisiten selbst wieder als Ziele auftreten, werden auch deren Prerequisiten überprüft, usw. Falls ein Ziel älter ist, als eines seiner Prerequisiten, wird es neu erzeugt. Auf diese Weise wird von einem Softwaresystem nur der Teil übersetzt, der von einer Änderung (z.B. Editierung) wirklich betroffen war.

Wird ein Fehler bei der Ausführung von Kommandos festgestellt, dann wird der gesamte Vorgang abgebrochen, es sei denn, die Option –i war gesetzt.

Das Makefile beschreibt die Abhängigkeiten zwischen

den Dateien des Softwaresystems und die Kommandos, die ausgeführt werden müssen, wenn eine Datei neu erzeugt werden soll.

Aufbau des Makefiles

Ein Makefile wird aus zwei Typen von Einträgen aufgebaut:

- Regeln, die die Abhängigkeiten zwischen Zielen und Prerequisiten und die eventuell auszuführenden Kommandos beschreiben.
- Makros, die zur Verkürzung der Hinschreibung eingesetzt werden können.

Jede Zeile des Makefile darf hinter einem # Kommentare enthalten.

Regeln

Regeln bestehen aus einer oder mehreren Zielen gefolgt von einem Doppelpunkt (:) hinter dem dann die Prerequisiten folgen. In den folgenden Zeilen können, sofern die Zeilen mit einem Leerzeichen beginnen, Kommandos stehen, die ausgeführt werden sollen, falls das Ziel neu generiert werden soll. Die Liste der Prerequisiten kann auch leer sein. In diesem Fall wird die Anweisungsfolge immer ausgeführt.

Makrodefinition

Häufig wiederkehrende Sequenzen können als Makro definiert werden. Eine Makrodefinition beginnt mit dem Namen des Makros gefolgt von einem Gleichzeichen (=) hinter dem eine Zeichenkette steht. Der Makroname darf aus einem oder mehreren alphanumerischen Zeichen (einschließlich des _) bestehen.

Makroaufruf

Ein Makro wird verwendet, indem anstelle der definierten Zeichenkette ein Dollar-Zeichen ($) gefolgt vom Makronamen geschrieben wird. Makronamen, die aus mehreren Zeichen bestehen, müssen in runde Klammern eingeschlossen werden.

spezielle Makros

In den Kommandozeilen können einige spezielle Makros verwendet werden:

- @ enthält Pfad und Namen (ohne Typ) des Ziels.
- < enthält den kompletten Namen des gerade bearbeiteten Ziels.
- @D enthält den Pfad des Ziels
- <D wie @D
- @F enthält den Namen (ohne Typ) des Ziels
- <F enthält den Namen (mit Typ) des Ziels

Beispiel

Die Datei \SPC\USER\SSWISEXA.MAK enthält ein Beispiel, mit dem die Beispielprogramme erzeugt werden können.

implizite Regeln

Da Make im SPC MODULA-2 Sprachsystem hauptsächlich zur Steuerung des Übersetzungsvorgangs eingesetzt wird, sind der Utility zwei Regeln implizit bekannt:

- eine .SBM Datei wird durch den Compiler aus seinen Prerequisiten erzeugt.
- eine .OBM Datei wird durch den Compiler aus seinen Prerequisiten erzeugt.

Print

Das Ausdrucken von Textdateien wird über die xShell durch die Print-Utility unterstützt. Print erlaubt bestimmte Formatierungen des auszugebenden Textes. Es ist z.B. möglich, die MODULA-2 Schlüsselwörter hervorzuheben, die Breite des Heftrandes einzustellen, etc.

Starten

Print wird durch das Kommando gestartet:
`print [<Filename>]`

ist der Name einer Datei die ausgedruckt werden soll. Falls keine Datei angegeben ist, läuft Print im interaktiven Modus als SSWiS-Applikation. Die Utility hat ihr eigenes Fenster, in dem der Status ausgegeben wird, sowie einen eigenen Menübalken. Über die Menüs können verschiedene Einstellungen vorgenommen werden, die weiter unten erläutert werden.

Anpassung an den Drucker

Print verwendet zur Ausgabe den Standard-Modul Printer. Dieser wird an den jeweils vorhandenen Drucker durch eine Wordplus-Konfigurationsdatei (*.CFG) angepaßt. Der Name der Printer-Konfigurationsdatei wird als Environment-Variable "PRINTERCONF" gespeichert. Gleichfalls wird die Anzahl der Spalten pro Zeile und die Anzahl der Zeilen pro Seite in dieser Variablen festgehalten. Printer-Konfigurationsdateien sind für alle gängigen Druckermodelle in der Public-Domain erhältlich.

Option-Menü

Über die Menüs kann Print in verschiedenen Parametern eingestellt werden. Das Option-Menü bietet die Möglichkeit, Die Zeilennumerierung ein- oder auszuschalten, den Schlüsselwort-Spezialdruck zu aktivieren und die Konfigurationsdatei einzustellen. Alle Parameter werden als Environment-Variable für den nächsten Aufruf gespeichert.

Font-Menü

Das Font-Menü bietet die Möglichkeit, die Schriftart zu wählen, sowie die Druckerparameter Zeichen/Zeile, Zeilen/Seite und Breite des Heftrandes einzustellen. Die Druckerparameter werden vom Modul Printer als

Environment-Variable gespeichert, und werden auch für nachfolgende Aufrufe des Moduls Printer verwendet.

File-Menü

Das File-Menü dient schließlich zur Steuerung des Druckvorgangs. Dazu kann auch noch die Drucker-Konfigurationsdatei gewählt werden. Alsdann wird über Print die auszugebende Datei gewählt. Der Druckvorgang kann zwischenzeitig unter- oder abgebrochen werden.

Konfigurations-Datei

Print kann über eine Konfigurations-Datei weitergehend eingestellt werden. Der Name der Konfigurationsdatei wird als Environment-Variable "PRINTFLAGS" zusammen mit anderen Einstellungen gespeichert. Eine Beispieldatei wird unter dem Namen \SPC\MISC\PRINT.INF mitgeliefert.

Aufbau der Konfigurations-Datei

Der Aufbau der Konfigurations-Datei ist zeilenorientiert. Das Format ist unten beschrieben. Jede Zeile wird durch einen * abgeschlossen. Danach kann ein beliebiger Kommentar folgen. Eine einzelne Zeile darf nicht länger als 80 Zeichen sein.

In der Konfigurationsdatei können beliebige Schlüsselwörter angegeben werden. Sofern keine Konfigurationsdatei zur Verfügung steht, werden die MODULA-2 Schlüsselwörter verwendet. Alle Schlüsselwörter werden durch Leerzeilen getrennt. Es können maximal 200 Schlüsselwörter mit nicht mehr als 1000 Zeichen insgesamt angegeben werden. Ein einzelnes Schlüsselwort darf maximal 80 Zeichen lang sein.

☞ Man beachte, daß die Indizierung keine syntaktische Analyse beinhaltet und deshalb i.a. nur korrekt ist, sofern auch der auszugebende Text und die Schlüsselwort-Liste korrekt strukturiert sind.

Indizierung

Für die Indizierung kann jedes Schlüsselwort mit einem optionalen Suffix versehen werden. Dazu kommen in Frage:

- (Klammer auf, beginnt eine neue Ebene und erhöht den Index.
-) Klammer zu, beendet eine Ebene und erniedrigt den Index.
- = Gleichzeichen, beläßt die Ebene und den Index.

Syntax

1. Zeile:	<Kennung> *
2. Zeile:	<Schrift> <Zeilen> <Zeichen> <Heftrand> *
3. Zeile:	<Kopf> <Nummern> <Spezial> <Index> <Anzahl SW>
4. Zeile ff.	enthalten Schlüsselwörter

<Kennung>	2434
<Schrift>	0..3 (Pica, Elite, Schmal, Breit)
<Zeilen>	Anzahl Zeilen pro Seite
<Spalten>	Anzahl Spalten pro Zeile
<Heftrand>	Breite des Heftrandes in Spalten
<Kopfzeile>	0: Kopfzeile aus, 1: Kopfzeile an
<Nummern>	Breite der Zeilennummern, negative Werte: keine Zeilennummern
<Spezial>	Art der Hervorhebung von Schlüsselwörtern. 1: fett, 2: unterstrichen, negative Werte: keine Hervorhebung
<Index>	Art der Indizierung von Schlüsselwörtern: 1: Hochstellung, 2: Tiefstellung, negative Werte: keine Indizierung
<Anzahl SW>	Anzahl der Schlüsselwörter, die ab Zeile 4 folgen

Paths

Die vom Compiler und anderen Werkzeugen des Sprachsystems verwendeten Suchpfade werden als Environment–Variablen gespeichert. Die Pfade für die Quellformen der Moduln haben die Namen PATH<N>, wobei <N> die Präferenz angibt. Pfade mit niedrigerer Präferenz werden zuerst durchsucht. Die Suche hört auf, wenn eine Variable mit der nächst höheren Präferenz nicht existiert.

Quellen und ableitbare Dateien trennen

Sollen Objektformen auf anderen Directories gehalten werden, als die Quellformen, dann kann zu jedem PATH<N> ein OBJPATH<N> angegeben werden. Andernfalls werden die Objektformen ebenfalls auf PATH<N> abgelegt und dort gesucht.

Bedienung

Paths unterstützt die Pflege der entsprechenden Environment–Variablen. Dabei sorgt Paths dafür, daß nach dem Löschen eines Pfades die Kette wieder geschlossen wird und erlaubt das Einfügen an einer bestimmten Position (Präferenz) in die Kette.

Paths kennt die Kommandos :

L – Anzeigen aller Pfade mit aufsteigender Präferenz.

M – Modifizieren einer Pfadkombination (PATH und OBJPATH). Es muß die Präferenz angegeben werden. Die neuen Pfadnamen werden über eine GEM–Fileselektor–Box erfragt.

D – Löschen einer Pfadkombination.

I – Einfügen einer Pfadkombination. Es wird die Präferenz der einzugebenden Pfade erwartet. Die Pfade selbst werden wieder über eine Fileselektor–Box erfragt.

Q – Quit, verläßt Paths.

Die Pfade, auf denen die Dateien gesucht werden, dürfen natürlich auch auf verschiedenen Laufwerken liegen. Dies ist wichtig, wenn Sie mit zwei Diskettenlaufwerken oder einer RAM–Disk arbeiten wollen. In

diesem Fall müssen Sie den entsprechenden Laufwerksbuchstaben mit angeben.

Wenn Sie Probleme haben, eine Datei zu übersetzen, oder der Compiler eine zu importierende Datei nicht findet, sind i.d.R. die Suchpfade nicht richtig eingestellt.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

SSWiS

Kapitel 8

Übersicht

SSWiS steht für Small Systems Windowing Standard und bedeutet, daß es sich hierbei um eine Gruppe von Funktionen handelt, die üblicherweise und gerade von Fenstersystemen auf kleinen Rechnern bereitgestellt werden. Die Funktionen von SSWiS werden also i.a. auf die Funktionen eines unterliegenden Fenstersystems abgebildet.

Vorzüge

Da keine Besonderheiten eines bestimmten Fenstersystems voraussgesetzt werden, können SSWiS Anwendungen leichter portiert werden, als Anwendungen, die das jeweils vorhandene Fenstersystem direkt ansprechen. Damit erbringt SSWiS vor allem zwei wichtige Leistungen:

- die Programmierung von fensterorientierten Applikationen wird einfacher, da SSWiS höhere Funktionen bereitstellt.
- SSWiS Applikationen können leichter portiert werden, da zumindest die Ansprache des Fenstersystems vereinheitlicht ist.

Einschränkungen

Da alles seinen Preis hat, soll dieser auch hier nicht verschwiegen werden:

- SSWiS stellt spezielle Funktionen eines Fenstersystems nicht zur Verfügung, um die Portabilität nicht zu gefährden und das ganze System kompakt und einfach zu halten.
- Vereinfachungen der Fensterverwaltung, die mitunter innerhalb einer bestimmten Applikation zulässig sind, können von SSWiS nicht angewendet werden, da unter SSWiS mehrere Applikationen quasiparallel ablaufen können.

Pseudo-Multitasking

Die inzwischen vorhandenen und mit der Version 1.4 ausgelieferten SSWiS Applikationen (z.B. Editor, Filer, xShell) beweisen jedoch, daß die Vorteile von SSWiS die Nachteile bei weitem aufwiegen. Vor allem wegen der Möglichkeit, mehrere Anwendungen gleichzeitig zu betreiben, ohne ein Multitasking Betriebssystem voraussetzen zu müssen, ist die Verwendung von SSWiS für interaktive Anwendungen in SPC-MODULA-2 hochinteressant.

Es ist die Philosophie von SSWiS, die Oberfläche des jeweils vorhandenen Systems nicht zu verfremden, sondern sie aus Gründen der Ergonomie so zu übernehmen, wie sie der Benutzer auch aus anderen Anwendungen kennt.

Wie die meisten Fenstersysteme, so bietet auch SSWiS neben der Verwaltung von Fenstern Zusatzdienste, wie Menüs und Formulare an.

In den folgenden Abschnitten wird zum einen die Bedienung von SSWiS unter GEM auf dem ATARI ST beschrieben. Zum zweiten wird die Programmierung von SSWiS Applikationen erläutert.

Bedienung

Die Bedienung einer SSWiS–Applikation (z.B. der Editor von SPC–MODULA–2) ist im Detail natürlich von der Applikation selbst und ihrer Funktionalität abhängig. Soweit es jedoch darum geht, Fenster, Menüs und Formulare zu bedienen, sind alle SSWiS Applikationen gleich. Dies ist Gegenstand der folgenden Abschnitte.

Fenster

Ein Fenster (engl. Window) ist ein Bereich des Bildschirmes, in dem eine Applikation einen Teil seiner Ausgabe abwickelt. Ob diese grafisch oder textuell ist, ist für das Fenster zunächst unerheblich. Die Größe und die Position des Ausgabebereiches ist sowohl von der Applikation, als auch vom Benutzer in bestimmten Grenzen einstellbar. Es können mehrere Fenster eingerichtet werden, die sich i.a. auch gegenseitig überlappen dürfen.

Fenstertitel

Jedes Fenster hat einen Titel, durch den die Bedeutung des Fensterinhaltes umschrieben wird. Normalerweise kann aus dem Titel auch auf die für das Fenster zuständige Applikation geschlossen werden. Unter SSWiS können zur gleichen Zeit mehrere Applikationen aktiv sein. Der Titel wird als Textzeile am oberen Rand des Fensters dargestellt.

Fenster bewegen

Das Fenster kann bewegt werden, indem man es mit der Maus an der Titelzeile anfaßt (linken Mausknopf drücken und und gedrückt halten) und den daraufhin erscheinenden Rahmen an eine andere Stelle des Bildschirms bewegt. Sobald der Rahmen losgelassen wird (linken Mausknopf loslassen), wird das Fenster an die neue Stelle gebracht.

Desktop

Der Hintergrund, auf dem ein Fenster abliegt, wird Desktop (Tischfläche) oder Desk genannt, da durch die ganze Benutzeroberfläche ein normaler Schreibtisch nachempfunden werden soll, auf dem beschriftete Blätter (Fenster) übereinanderliegen und hin– und her bewegt werden. Der Desktop wird durch eine graue Fläche dargestellt, die fast den ganzen Bildschirm überspannt.

Fenster-Icons

Der Desktop steht den Applikationen nicht für Ausgaben zur Verfügung, sondern wird von SSWiS für die Verwaltung von Fenstern benötigt. Für jedes vorhandene Fenster wird in der linken unteren Ecke des Desktop beginnend ein beschrifteter Balken angelegt. Die Beschriftung ergibt sich aus den ersten Buchstaben des Fenstertitels. Wozu werden diese Balken gebaucht? SSWiS kann bis zu 20 Fenster verwalten. GEM jedoch beschränkt die Anzahl von Fenstern auf 4 bzw. 6. Wenn nun mehr als 6 Fenster über SSWiS angelegt werden, dann schließt SSWiS jeweils das hinterste Fenster (mehr als 6 geöffnete Fenster sind auf dem 14" Monitor des ATARI ST auch nicht sinnvoll). Das geschlossene Fenster wird von SSWiS jedoch weiter verwaltet und kann jederzeit wieder geöffnet werden. Dies geschieht, indem man den Balken des Fensters mit der Maus anklickt. Falls schon 6 Fenster geöffnet sind, muß natürlich das hinterste wieder geschlossen werden, usw.

das aktive Fenster

Das oberste Fenster hat bei GEM eine Sonderstellung, die unter SSWiS so beibehalten wird. Das oberste Fenster wird das aktuelle Fenster genannt. Alle Tastatureingaben fließen immer der Applikation zu, der das oberste Fenster gehört. Um das aktuelle Fenster kenntlich zu machen, wird die Titelzeile des aktuellen Fensters grau dargestellt, während die Titelzeilen aller anderen Fenster einen weißen Hintergrund haben. Ein Fenster kann zum aktuellen (obersten) Fenster gemacht werden, indem entweder, wie oben beschrieben, der zu dem Fenster gehörende Balken angeklickt wird, oder, falls das Fenster schon geöffnet ist, einfach in das Fenster geklickt wird.

Fenster wieder schließen

Ein geöffnetes Fenster kann vom Benutzer geschlossen werden, indem die sogenannte Schließ-Box links neben der Titelzeile angeklickt wird. Fenster zuschliessen kann sinnvoll sein, um z.B. mehr Übersicht auf dem Desktop zu bekommen. Man beachte, daß mit dem Schließen eines Fensters unter SSWiS keines-

wegs die dazugehörige Applikation beendet wird, sondern lediglich das sichtbare Fenster vom Desktop verschwindet. Wie oben beschrieben, kann das Fenster jederzeit wieder geöffnet werden. Die zum Fenster gehörende Applikation merkt davon nichts.

Randelemente von Fenstern

Weitere Bedienungsmöglichkeiten hängen davon ab, welche Randelemente für die einzelnen Fenster konfiguriert sind. Neben der Titelzeile und der Schließ–Box können noch weiterhin konfiguriert werden:

- eine Meldungszeile zur Ausgabe von einzeiligen Meldungen durch die Applikation.
- ein vertikaler und/oder ein horizontaler Scroll–Balken, um den Fensterinhalt zu verschieben, falls das Fenster zu klein ist, diesen vollständig darzustellen. Ihre Funktion wird unten weiter erklärt.

Scroll–Balken

- eine Größen–Box, um die Größe des Fensters innerhalb konfigurierter Grenzen zu verändern. Dazu wird das Fenster an der Größen–Box (in der rechten unteren Ecke) angefaßt (linke Maustaste drücken und gedrückt halten) und das daraufhin erscheinende Rechteck auf die gewünschte Größe gebracht. Nach dem Loslassen, wird das Fenster entsprechend vergrößert oder verkleinert, jedoch nur innerhalb der durch die Applikation zugelassenen Grenzen.

Größen–Box

- eine Maximalgrößen–Box, um das Fenster auf die maximal zulässige Größe und wieder zurück zu bringen. Die maximale Fenstergröße wird von der Applikation bestimmt.

Maximalgrößen–Box

Fensterinhalt

Innerhalb des Fensters wird von der Applikation ein Bild ausgegeben, welches i.a. sehr viel größer ist, als das Fenster selbst. Jede, in einem Fenstersystem laufende Applikation muß deshalb dafür sorgen, daß sie keinesfalls Ausgabe außerhalb der Bildschirmbereiche erzeugt, die ihr durch das Fenstersystem (in diesem Fall SSWiS) zugeteilt werden. Das Bild, das innerhalb des Fensters ausgegeben wird, wird die Welt genannt. Die Applikation beschreibt die Ausgabe in Weltkoordinaten und empfängt alle Eingaben in Weltkoordina–

Weltkoordinaten

ten. Diese sind von der Position des Fensters und damit von den Bildschirmkoordinaten unabhängig.

Größe des Weltbildes

Die Scroll-Balken signalisieren unter GEM die Größe des Bildes, indem die Schieber im Verhältnis zum Fenster genauso groß sind, wie das Fenster im Verhältnis zur Welt. Mit anderen Worten: wenn der Schieber halb so groß ist wie das Fenster, dann ist das Fenster halb so groß wie die Welt. Indem der Schieber mit der Maus bewegt wird, kann das Fenster quasi

das Fenster scrollen

über die Welt bewegt werden. Anklicken des grauen Scroll-Balkens selbst blättert in die entsprechende Richtung. Die Pfeile können dazu benutzt werden, das Fenster schrittweise in die gewünschte Richtung über die Welt zu bewegen. Die Schrittweite wird von der jeweiligen Applikation bestimmt.

Menüs

Um Eingaben an eine Applikation zu machen, können neben der Tastatur in einem fensterorientierten System auch Menüs verwendet werden. Menüs sind zu Menütiteln strukturiert. Ein Menütitel enthält mehrere, thematisch zusammengehörige Menüeinträge. Wählt man

Titel und Einträge

einen Menüeintrag aus, dann gibt man einer Applikation ein Kommando, ähnlich als wenn man eine Funktionstaste drückt. Da Menüs konfiguriert werden können, sind sie wesentlich flexibler, als Funktionstasten, jedoch werden die am häufigsten benötigten Menüfunktion meist auch über Funktionstasten zugänglich gemacht.

Menüeinträge selektieren

Unter GEM sind alle Menütitel zu einer Menüzeile am oberen Bildschirmrand zusammengefaßt. Fährt man mit der Maus in einen der Menütitel, dann klappt das eigentliche Menü mit seinen Menüeinträgen herunter. Ein Menüeintrag wird durch Anklicken ausgewählt. Das Menü verschwindet danach wieder. Der Menübalken

Menübalken

kann nur die Menütitel einer Applikation enthalten. Da unter SSWiS mehrere Applikationen nebeneinander laufen, müssen mehrere Menübalken verwaltet werden.

Steuerung des Menübalkens

Es ist immer der Menübalken der Applikation sichtbar, der das aktuelle Fenster gehört. Wenn also das aktuelle Fenster wechselt, ändert sich i.a. auch der Menübalken, es sei denn, das neue aktuelle Fenster gehört der gleichen Applikation. Der Name der Applikation, deren Menübalken gerade sichtbar ist, ist unter SSWiS gleichzeitig der Titel des ersten (links außen) Menüs. Dieses Menü steht den Applikationen nicht zur Verfügung, sondern ist unter GEM für sogenannte Accessories reserviert.

Attribute von Menüeintragen

Menüeinträge können dauernd oder vorübergehend ausgeschaltet (disabled) werden. Ausgeschaltete Menüeinträge werden grau dargestellt. Ein anderes Attribut (abgehakt) wird durch einen kleinen Haken am linken Rand des Menüeintrages dargestellt. Ausgeschaltete Menüeinträge können nicht gewählt werden. Die Bedeutung des Hakens hängt von der jeweiligen Applikation ab und ist dort dokumentiert.

Identifikation von Applikationen

Unter GEM wird der erste Eintrag des ersten Menüs dafür verwendet, die Applikation zu identifizieren. SSWiS Applikationen tun dies über ein vereinheitlichtes Formular, in dem Programmversion, Autor und ein Copyright-Vermerk zusammengefaßt sind.

Funktionstasten

☞ Es ist gute Praxis, häufig verwendete Menüfunktionen auch über die Tastatur zugänglich zu machen. Dafür bieten sich Funktionstasten und alternativ belegte Tasten (Taste in Verbindung mit Alternate oder Control betätigen) an. Die zu einem Menüeintrag gehörenden Tastaturkommandos sollten der leichteren Erlernbarkeit wegen im Menüeintrag selbst angedeutet sein (z.B. ALT A, F1, etc.)

Formulare

Formulare sind ein weiterer Service von Fenstersystemen, um den Dialog zwischen Applikation und Benutzer zu verbessern. Der Aufbau und die Verwaltung von Formularen sind in den verschiedenen Fenstersystemen sehr unterschiedlich. Allerdings benötigt eine typische Anwendung nur wenige Grundtypen von Formularen. Diese werden von SSWiS zur Verfügung gestellt.

Meldungsformular

Der erste Formulartyp dient dazu, eine Meldung oder Mitteilung auszugeben, und sich vom Benutzer eine Quittung geben zu lassen. Der Meldungstext darf eine Zeile lang sein. Die Quittung besteht darin, daß einer von maximal vier konfigurierbaren Knöpfen zum Verlassen des Formulars gewählt wird. Solange ein Formular aufgeschaltet ist, kann unter GEM keine andere Eingabe gemacht werden, außer der Formularbedienung.

Abfrageformular

Ein weiterer Formulartyp ermöglicht es, eine Meldung auszugeben, und eine Quittung anzufordern. Der Mechanismus ist der gleiche wie beim Meldungsformular. Zusätzlich kann aber ein Text eingegeben werden und bis zu vier konfigurierbare Optionen gewählt werden. Um den Text einzugeben, muß zuerst die Textzeile selektiert werden. Sie enthält i.a. schon einen Vorschlagswert, der mit den üblichen Editierfunktionen geändert werden kann. Die Optionen können durch Anklicken mit der Maus ein- und ausgeschaltet werden. Der Formulardialog wird durch Anklicken einer der Quittungstasten wie oben beschrieben beendet.

Identifikationsformular

Dieses Formular wird verwendet, um auf einfache Weise eine Identifikation der Applikation zu ermöglichen. Die einzige Bedienung ist das Anklicken des OK-Knopfes.

Mit den genannten Formulartypen wurden inzwischen eine Reihe von Applikationen erstellt und es hat sich gezeigt, daß die Formulare für portable Applikationen ausreichend sind. Zudem werden die von den verschiedenen Applikationen geführten Dialoge etwas ver-

einheitlicht, was letztlich wieder der Einfachheit des Gesamtsystems zugute kommt. Die Bedienung von SSWiS–Applikationen ist, soweit dies ohne Bezug auf die Applikation selbst möglich ist, hiermit beschrieben.

Programmier-Schnittstelle

Die folgenden Abschnitte befassen sich mit der Programmierung von SSWiS Applikationen. Da die Möglichkeit, mehrere Applikationen unter SSWiS nebeneinander laufen zu lassen, inzwischen von vielen Programmierern gerne genutzt wird, liegt es nahe, neue interaktive Applikationen ebenfalls als SSWiS Applikation zu realisieren. Allerdings sollte am Anfang der Realisierung eine gründliche Analyse stehen, um festzustellen, ob die Möglichkeiten von SSWiS für das Vorhaben ausreichend sind.

Struktur von SSWiS Applikationen

SSWiS Applikationen (im folgenden nur noch Applikationen genannt) haben einen bestimmten, durch die Funktionsweise von SSWiS vorgegebenen Aufbau. Nachdem sich eine Applikation initialisiert hat, läuft sie in einer Schleife, innerhalb derer sie ständig die Kontrolle an SSWiS abgibt, um Ereignisse (Events) zu bearbeiten. Die Schleife wird durchlaufen, bis das Programm beendet werden soll. Danach folgt i.a. eine geordnete Terminierung, in der alle Resourcen freigegeben werden. Die Grundstruktur ist durch das folgende Programmstück wiedergegeben:

PollEvents Loop

```
MODULE Application;

....

PROCEDURE Initialise;

BEGIN
  SSWiS.Register (..., 'Application', ...);
END Initialise;

BEGIN
  Initialise;
  REPEAT
    SSWiS.PollEvents;
  UNTIL Xit;
  Terminate;
END Application;
```

Applikation anmelden

Man beachte, daß sich Applikationen bei SSWiS anmelden müssen (Register), den SSWiS verwaltet mehrere Applikationen nebeneinander, wobei jede ihre eigenen Fenster und Menüs haben kann. Während der Terminierung erfolgt natürlich eine entsprechende Abmeldung, die hier nicht gezeigt wurde.

Application Handle

Bei der Anmeldung vergibt SSWiS eine Kennung (Handle) für jede Applikation. Unter dieser Kennung ist die Applikation als Klient (Client) von SSWiS registriert. Bei fast allen Aufrufen ist die Kennung als Parameter wieder zu übergeben, so daß SSWiS den richtigen Satz von Resourcen (z.B. Fenster) zuordnen kann.

Eventverarbeitung

Events

EventReport

Accept–Prozedur

Nun ergibt sich natürlich die Frage, woher die Eingaben des Benutzers für die Applikation kommen, und wo sie verarbeitet werden. Eingaben haben zunächst die Form von Ereignissen (Events). Das sind Signale, die anzeigen, daß etwas passiert ist. Was nun genau passiert ist, wird durch eine Datenstruktur beschrieben, die EventReport genannt wird. Alle Ereignisse gelangen zunächst an SSWiS. Dort sind sie noch in einer sehr rohen Form und für Applikationen noch nicht leicht zu verarbeiten. Außerdem steht noch nicht fest, welche Applikation das Ereignis empfangen soll. Die Zuordnung und eine weitgehende Vorverarbeitung übernimmt SSWiS. Alsdann wird der Event an die richtige Applikation ausgeliefert, die dann ihrerseits das Ereignis weiterverarbeitet. Die Prozedur der Applikation, welche die Events aufnehmen und verarbeiten soll wird die Accept–Prozedur genannt.

Event–Typen

Zur Verarbeitung von Ereignissen wird sie zunächst feststellen, um welchen Typ von Ereignis es sich handelt, denn es gibt nur eine Accept–Prozedur, über den alle Ereignisse mitgeteilt werden. Typen von Ereignissen sind z.B. Mausklicks oder Tastatureingaben. Je nach Event–Typ stehen die benötigten Zusatzinformationen (Koordinaten, etc.) im Event–Report, anhand dessen die Applikation die auszuführenden Funktionen geeignet parametrieren kann.

Accept anmelden

Die Accept-Prozedur wird als sogenannte Callback Procedure realisiert. Eine solche Prozedur wird typischerweise von einem Modul, in diesem Fall SSWiS, zu einem späteren Zeitpunkt zurückgerufen, eben dann, wenn ein Ereignis eingetreten ist. Dazu muß sie natürlich bei SSWiS bekannt gmacht werden. Dies geschieht während der Anmeldung bei SSWiS. Das folgende Programmstück soll den wichtigen Mechanismus verdeutlichen. Alle momentan nicht wichtigen Parameter wurden weggelassen.

```
MODULE Application;

....

PROCEDURE Accept (..., Report);

BEGIN
  CASE Report.Type OF
  ...
  END;
END Accept;


PROCEDURE Initialise;

BEGIN
  SSWiS.Register (..., Accept);
END Initialise;


BEGIN
  Initialise;
   REPEAT
    SSWiS.PollEvents;
  UNTIL Xit;
  Terminate;
END Application;
```

Die Accept-Prozedur wird während der Anmeldung der Applikation bekanntgegeben. Hieraus folgt unmittelbar, daß jede Applikation eine Accept-Prozedur haben muß, also in irgendeiner Weise auf die Events von

SSWiS reagiert. Wie die Ereignisse im Detail zu verstehen und zu behandeln sind, wird in einem späteren Abschnitt erklärt.

Fenster

Um den prinzipiellen Aufbau einer SSWiS-Applikation zu vervollständigen, ist es noch notwendig, die Verwendung von Fenstern zu erklären. Fenster sind die Basis für die Verteilung der vorhandenen Bildschirmfläche an die registrierten Applikationen. Eine Applikation darf deshalb nur innerhalb ihrer eigenen Fenster Ausgabe erzeugen.

Anstöße für Ausgaben

Der Anstoß zur Erzeugung von Ausgabe kann auf verschiedene Ursachen zurückgehen. Zunächst ist klar, daß Ausgabe erzeugt werden muß, wenn sich das im Fenster dargestellte Bild geändert hat. Weiterhin kann eine neuerliche Ausgabe eines Fensters dadurch notwendig werden, daß das Fenster vergrößert wurde, oder aber, weil sich die gegenseitige Überlappung von Fenstern geändert hat. In einem solchen Fall kommt der Anstoß nicht von der Applikation selbst, sondern wird durch äußere Ereignisse hervorgerufen.

Clip-Gebiete

Redraw-Prozedur

In beiden Fällen kann das Gebiet, das den Fensterinhalt beschreibt, komplex sein, da das Fenster durch andere Fenster teilweise verdeckt sein kann. In jedem Fall besteht dieses Gebiet aber aus einer Reihe von Rechtecken, deren Kanten parallel zu den Achsen des Koordinatensystems sind. Da die Lage aller Fenster und damit das Gebiet, das zu einem bestimmten Fenster gehört, ist nur SSWiS als koordinierende Stelle bekannt. Die Applikation stellt lediglich eine Redraw-Prozedur bereit, die einen Ausschnitt der Ausgabe erzeugt. Die Restore-Prozedur wird ebenfalls als Callback-Prozedur ausgeführt. Sie wird beim Erzeugen des Fensters angegeben und von SSWiS bei Bedarf aufgerufen. Daraus folgt, daß zu jedem Fenster eine eigene Restore-Prozedur gehört (die u.U. gleich sein können) und daß eine Applikation mehrere Restore-Prozeduren haben kann.

Aufgabe der Restore–Prozedur

Es ist wichtig, da die Restore–Prozedur nichts anderes tut, als den von SSWiS angeforderten Fensterausschnitt zu erneuern. Da die gegenseitige Verdeckung von Fenstern kompliziert sein kann, ist es u.U. nötig, daß SSWiS sehr oft hintereinander die Restore–Prozedur aufrufen muß. Wann und wie oft dies der Fall ist, kann die Applikation nicht kontrollieren.

Das obige Beispiel wird im folgenden um ein erstes Fenster erweitert. Der Einfachheit halber wird das Fenster während der Initialisierung angelegt. Natürlich können auch zu jedem anderen Zeitpunkt zwischen der Initialisierung und der Terminierung weitere Fenster erzeugt werden.

```
MODULE Application;

PROCEDURE Accept (..., Report);
BEGIN
  CASE Report.Type OF
  ...
  END;
END Accept;

PROCEDURE Restore (...);

BEGIN
END Restore;

PROCEDURE Initialise;
BEGIN
  SSWiS.Register (..., Accept);
  SSWiS.CreateWindow (..., Restore);
END Initialise;

BEGIN
  Initialise;
  REPEAT
   SSWiS.PollEvents;
  UNTIL Xit;
  Terminate;
END Application;
```

Parameter der Restore–Prozedur

Die Parameter der Restore–Prozedur werden in einem späteren Abschnitt erklärt. Der prinzipielle Aufbau einer SSWiS–Applikation ist damit vollständig. Zusätzliche Funktionen wie Menüs und Formulare haben auf die Struktur des Programms keinen Einfluß mehr.

Koordinatensysteme

Eine Applikation erzeugt grafische Ausgabe um am Bildschirm ein Bild auszugeben. Die Koordinaten, in denen dieses Bild beschrieben wird, werden allgemein als Welt–Koordinaten bezeichnet. Das Bild wird deshalb auch oft als Welt–Bild oder kurz als Welt bezeichnet.

Welt–Koordinaten

Geräte–Koordinaten

Auf dem Bildschirm, den sich mehrere Applikationen teilen müssen, kann i.a. nur ein Ausschnitt des Welt–Bildes dargestellt werden. Dazu muß eine Grafik in Bildschirm–Koordinaten (Gerätekoordinaten) beschrieben werden. Das Gerätekoordinatensystem ist durch die Größe des Bildschirms meist sehr eingeschränkt, z.B. auf 640x400 diskrete Koordinatenwerte.

Transformationen

Durch die unterschiedliche Lage der Welt–Bilder im Verhältnis zum Bildschirm müssen die Welt–Koordinaten zum Zwecke der Ausgabe von der Anwendung in Bildschirmkoordinaten transformiert werden. SSWiS sieht dabei lediglich vor, daß das Welt–Koordinatensystem gegenüber dem Bildschirm–Koordinatensystem verschoben ist, so daß die Transformation durch Additionen beschrieben werden kann. Eine Skalierung oder Rotation ist damit nicht möglich.

Datentypen

SSWiS exportiert die Datentypen sowohl für Welt–Koordinaten, als auch für Bildschirm–Koordinaten. Man beachte, daß die Welt–Koordinaten durch 32–Bit Werte beschrieben sind. Dadurch ist auf dem MC68000 ein gewisser Mehraufwand bei Divisionen und Multiplikationen nötig. Die Koordinaten–Datentypen werden durch Datentypen für Punkte und Strecken ergänzt. Da ein Rechteck durch seine Diagonale eindeutig beschrieben ist, wird der Datentyp Lines auch für Rechtecke verwendet.

Restore–Prozedur

Die Restore–Prozedur eines Fensters ist dafür zuständig, in einem angeforderten Bildschirmbereich einen bestimmten Teil des Weltbildes auszugeben. Da keine Skalierung und keine Rotation des Weltkoordinatensystems gegenüber dem Bildschirm–Koordinatensystem vorgesehen ist, reicht es aus, nur die Verschiebung der beiden Koordinatensysteme als Vektor anzugeben. Der Parameter heißt Offset und bezeichnet den Punkt des Weltkoordinatensystems, der auf die linke obere Fensterecke abgebildet wird. Der Parameter Area beschreibt die Position und die Größe eines Rechtecks im Weltkoordinatensystem, das durch den aktuellen Aufruf auszugeben ist. Auf jede durch die Restore–Prozedur auszugebende Koordinate muß also der Vektor Offset hinzuaddiert werden, um Bildschirm–Koordinaten zu erhalten.

Clipping

Weiterhin ist es die Aufgabe der Restore–Prozedur, dafür zu sorgen, daß keinesfalls außerhalb des angeforderten Rechtecks Ausgabe erzeugt wird, denn das Gebiet außerhalb gehört i.a. zu einem anderen Fenster, oder zu Teilen, die nicht von der Applikation verwaltet werden (z.B. Fenstertitel). Normalerweise wird man dafür ein Clip–Rechteck verwenden, jedoch sind auch alle anderen, gleichwertigen Methoden zulässig.

erzeugen grafischer Ausgabe

Nun bleibt noch die Frage, wie die grafische Ausgabe selbst erzeugt wird. SSWiS macht darüber keine Annahmen oder Vorschriften, jedoch werden als Ergänzung zu SSWiS Moduln angeboten, die Ausgabefunktionen und systemunabhängiger Weise bereitstellen (TextWindows, ...). Dadurch wird die mit SSWiS gewonnene Portabilität auch auf die Restore–Prozedur ausgedehnt. Allerdings ist es auch akzeptabel, die Ausgabe in geräte– oder systemabhängiger Weise zu beschreiben, da die ganze Ausgabe auf die Restore–Prozedur konzentriert ist. Unter GEM sind VDI, AES und die Line–A Routinen Beispiele solcher systemabhängiger Grundfunktionen. Das folgende Programmfragment demonstriert eine einfache Restore–Prozedur,

die auf einem hypothetischen Grafiksystem mit den Aufrufen SetClipping und Circle aufsetzt. Natürlich ist der Kreis nur sichtbar, wenn tatsächlich Teile davon in das Clip–Rechteck fallen.

```
PROCEDURE Restore (   Owner     : ModuleHandles;
                      Window    : WindowHandles;
                      WorldArea : Lines;
                      Offset    : Points);

VAR Clip : ScreenLines;

BEGIN
  Clip.A.X:= WorldArea.A.X + Offset.X;
  Clip.A.Y:= WorldArea.A.Y + Offset.Y;
  Clip.B.X:= WorldArea.B.X;
  Clip.B.Y:= WorldArea.B.Y;
  SetClipping (Clip);
  Circle (100000D+Offset.X,100000D+Offset.y,100);
END Restore;
```

Falls sich die grafische Ausgabe jedoch aufwendiger gestaltet, dann ist es sinnvoll, die systemabhängigen Teile mindestens durch einen Modul zu isolieren. Text–Windows ist ein Beispiel für einen solchen Modul. Er verbirgt die aktuelle Implementierung von Textausgabe und macht die rufende Restore–Prozedur damit portabel.

Ereignisse

Nachdem nun der prinzipielle Aufbau einer SSWiS–Applikation beschrieben ist, müssen noch die Ereignisse erklärt werden, die eine Applikation zu verarbeiten hat. Ereignisse werden von SSWiS vorverarbeitet, klassifiziert, mit Detailinformation versehen und einer bestimmten Applikation zugeordnet. Das Ereignis wird alsdann der Applikation zur Verarbeitung übergeben, indem ihre Accept–Prozedur aufgerufen wird. Als Parameter werden von SSWiS ein sogenannter Event–Report übergeben, der die benötigte Detail– und Begleitinformation enthält.

Zuordnung von Ereignissen

Da die meisten Events danach zugeordnet werden, in welchem Fenster gerade die Maus steht, wird das

Handle des entsprechenden Fensters ebenfalls als Parameter der Accept–Prozedur übergeben. Unter GEM werden Ereignisse immer dem obersten (aktiven) Fenster zugeordnet. Entsteht ein Ereignis außerhalb des aktiven Fensters, dann erzwingt GEM zunächst, daß dieses Fenster aktiv wird.

Ereignisbehandlung

Die Accept–Prozedur führt daraufhin, als Teil der Applikation, eine Ereignisbehandlung durch. Dafür ist zunächst der Ereignistyp maßgebend. Abhängig vom Ereignistyp enthält der EventReport unterschiedliche Detailinformationen, deren Bedeutung im folgenden erklärt wird.

Keyboard–Events

Alle Tasteneingaben werden als Tastenereignis klassifiziert. Der EventReport enthält die gedrückten Tasten. Dabei werden bis zu 20 Tastenereignisse zu einem EventReport zusammengefaßt. Von der Applikation wird erwartet, daß sie darauf entsprechend reagiert. Die Blockung mehrerer Ereignisse ist immer dann interessant, wenn dadurch die nachfolgende Verarbeitung effizienter gestaltet werden kann. Je mehr eine Applikation bei der Verarbeitung von Tastenereignissen in Rückstand kommt, desto mehr kann sie dann durch die größere Blockung sparen und so wieder aufholen.

Meta–Tasten

Gleichzeitig wird mit den Tastencodes der Status der Meta–Tasten (Shift, Alternate und Control) übergeben. Dabei soll angenommen werden, daß die Meta–Tasten bei allen übergebenen Tastencodes gedrückt waren.

Mausposition

Schließlich wird noch die momentane Mausposition übergeben. Die Mausposition wird in Welt–Koordinaten des Fensters übergeben, über das das Ereignis zugeordnet wurde.

Tasten–Codes

Alle Funtions– und Editier–Tasten, die von SSWiS verarbeitet werden, sind als Konstanten deklariert, um ihre Auswertung zu vereinfachen. Die Codes unterhalb 128 sind durch den ASCII–Zeichensatz festgelegt; die Codes zwischen 128 und 255 sind den nationalen

Sonderzeichen vorbehalten und müssen applikationsspezifisch verarbeitet werden.

Mouse-Events

Die Maus verursacht verschiedene Ereignisse, die von Betätigungen der Maustaste (SSWiS geht von einer Maus mit nur einer Taste aus) herrühren. In jedem Fall wird der Status der Meta-Tasten, die Position der Maus (wie oben) und der Status der Maus-Taste übergeben.

Mausaktivität

Die ereignisverursachende Aktivität ist die wichtigste Information des Maus-Ereignisses. Einfaches und doppeltes Klicken der Maustaste liefern je eine Ereignis. Wenn die Maustaste gedrückt wird, wird ein Ereignis bearbeitet, genauso, wenn sie wieder losgelassen wird.

Motion-Event

Das Maus-Echo (s.u.) kann so eingestellt sein, daß die Applikation selbst für dessen Generierung zuständig ist. In diesem Falle wird ein Mausereignis immer dann erzeugt, wenn die Maus ein vorgebbares Raster (s.u.) verläßt. Ein Beispiel dafür ist die Markierung der Selektion im SPC Editor.

Window-Violation

Wenn bei gedrückter Maustaste das Fenster verlassen wird, dem das Drücken der Maustaste gemeldet wurde, dann entstehen mit einer bestimmten Frequenz ständig weitere Ereignisse, die anzeigen, daß sich die Maus aus dem Fenster entfernt hat. Die Applikation kann in diesem Fall das Fenster weiterscrollen, um anschließende Bereiche sichtbar zu machen.

Menü-Events

Über das aktive Fenster bestimmt SSWiS, welcher Menübalken sichtbar sein soll. Beim Wechseln des aktiven Fensters wechselt i.a. deshalb auch der Menübalken. Menüs enthalten bis zu sieben Titel und jeder Titel bis zu 8 Einträge. Ein Menü-Ereignis ist durch die Angabe des Titels und des Eintrages vollständig spezifiziert.

Message–Events

(noch nicht implementiert)

Nachrichten können zwischen Applikationen ausgetauscht werden, um sich gegenseitg Anstöße für bestimmte Aktivitäten zu geben. Es wird ein standardisiertes Nachrichtenformat vorgegeben.

Timer–Events

Wenn SSWiS auf Ereignisse wartet und keine Ereignisse anfallen, geht SSWiS davon aus, daß der Benutzer vorübergehend untätig ist. Die dadurch frei verfügbare Prozessorleistung wird dann in Form eines Timer–Events an die bei SSWiS registrierten Applikationen reihum vergeben. Ein Timer–Event kann nicht dazu verwendet werden, zeitkritische Aktivitäten zu steuern, sondern nur, um in einfacher Weise die Komponente Zeit in eine Applikation zu bringen. Es dürfen keine langwierigen Aktivitäten innerhalb des Timer–Events durchgeführt werden, da dadurch die Reaktionsfähigkeit des Systems leiden würde.

Identification–Events

Die Aufforderung an die Applikation ist deshalb ein spezielles Feature von SSWiS, da unter SSWiS verschiedene Applikationenen koexistent sein können. Es ist deshalb nicht mehr damit getan, eine Startup–Meldung auszugeben, etc. Vielmehr ist die ganze Benutzeroberfläche unter SSWiS sehr stark standardisiert, und die Herkunft eines Programms ist nicht ohne weiteres ersichtlich. Als Folge des Ereignisses sollte eine Applikation die SSWiS Funktion Identify aufrufen und Programmnamen, Autor, Copyright, etc. zur Verfügung stellen.

Unter GEM wird dieses Ereignis natürlich durch selektieren des About–Eintrages im Menü erzeugt.

PollEvents Loop

Aus den angegebenen Programmfragmenten ist ersichtlich, daß eine Applikation, sobald sie ihre Initialisierung durchgeführt hat, in eine Schleife eintritt, in der nur noch SSWiS.PollEvents aufgerufen wird. Innerhalb von SSWiS werden dann, als Folge von Ereignissen und Programmaktivität die angeschlossenen Applika–

tionen aktiviert, indem deren Accept- bzw. Restore-Prozeduren aufgerufen werden.

Diese Programmstruktur ist typisch für Applikationen eines Fenstersystems und unterscheidet sich vollständig vom klassischen, sequentiellen Programmaufbau. Bei letzterem sind die dialogführenden Programmstellen in einem sonst sequentiellen Programmablauf eingebettet.

Unter SSWiS wird die ganze Arbeit aller aktiven Applikationen in den Accept-Prozeduren geleistet. Die Funktion SSWiS.PollEvents wird i.a. nur in der Schleife des Hauptprogramms aufgerufen, um Rekursionen zu vermeiden.

Dynamic Linking

SPC-MODULA-2 bietet die Möglichkeit, Programme zuzuladen. Dieses als Dynamic Linking bezeichnete Feature ist an anderer Stelle erklärt. Unter SSWiS ist jedoch wichtig zu wissen, daß das Zuladen mittelbar oder unmittelbar von einer Accept-Prozedur ausgeht und dadurch natürlich ein neues Hauptprogramm zum Ablauf kommt. Dieses läuft, nachdem es die neue Applikation initialisiert hat, ebenfalls in eine PoolEvents Schleife. SSWiS wird ab dann von deren PollEvents-Aufruf angetrieben (und nicht mehr von der startenden Applikation).

Starten und Terminieren von Applikationen

Einfacher ausgedrückt: Die PollEvents Schleife der zuletzt zugeladenen Applikation ist die SSWiS treibende Kraft. Damit wird auch klar, warum die Applikationen nur in der umgekehrten Startreihenfolge terminiert werden. (z.B. beendet sich die xShell erst, wenn alle Applikationen, die von ihr gestartet wurden, ebenfalls terminiert wurden).

Benutzung des Stack

Eine weitere, sich hieraus ergebende Tatsache ist, daß alle Applikationen auf dem gleichen Stack laufen. Durch das Nachladen von Applikationen innerhalb einer Accept-Prozedur erhöht sich natürlich der Stack-Bedarf, und zwar um den der zugeladenen Applikation

und den der nachladenden Accept–Prozedur.

Operatoren

Die Operatoren von SSWiS werden in den folgenden Abschnitten mit der thematischen Ordnung diskutiert, die auch im DEFINITION MODULE SSWiS angegeben ist. Wenn die zuvor genannten Konzepte klar geworden sind, dann sollte es keine Schwierigkeiten bereiten, die knappen Kommentare des Definitionsmoduls zu lesen.

Steuerung von SSWiS

SSWiS initialisiert sich innerhalb des Modulrumpfes. Die ordnungsgemäße Terminierung erfolgt über den vom–System bereitgestellten Terminierungsmechanismus. Applikationsprogramme brauchen sich deshalb nicht darum zu kümmern.

Die SSWiS treibende Funktion ist PollEvents. Damit kehrt der Programmablauf immer wieder zu SSWiS zurück. Mitunter ist es wünschenswert, SSWiS nur die Möglichkeit zu geben, die Fenster zu restaurieren (z.B. nachdem eine Dateiauswahl–Box wieder vom Bildschirm verschwunden ist). Dafür gibt es die Funktion Resync. Reinit veranlaßt SSWiS, den Desktop, den Menübalken, etc. zu reinstallieren, nachdem aus einer fremden Applikation zurückgekehrt wurde (z.B. Tempus).

Anmelden der Applikation

Applikationen müssen sich bei SSWiS an– und abmelden, da SSWiS für jede Applikation einen Satz an Resourcen vorhält

Fenster Anlegen und kontrollieren

Fenster müssen von der Applikation angelegt und wieder gelöscht werden. Die Applikation (nicht SSWiS) gibt jedem Fenster eine Nummer, über die später auf das Fenster wieder Bezug genommen werden kann.

Fenster können ganz nach vorn oder ganz nach hinten gebracht werden. Ein Fenster, welches nach hinten gebracht wird, wird von SSWiS automatisch geschlossen.

Die Randelemente eines Fensters können in einem weiten Bereich konfiguriert werden. Weiter können der

Fenstertitel und die Meldezeile (falls konfiguriert) gesetzt werden.

SSWiS steuert die Restore–Prozeduren so, daß immer dann, wenn sich das Fenster–Arrangement verändert hat, alle Fenster wieder geeignet restauriert werden. Änderungen, die sich innerhalb des Fensterinhaltes ergeben (z.B. durch Eingeben von Zeichen in einem Texteditor) müssen SSWiS explizit mitgeteilt werden (ExplicitRestore), damit SSWiS die notwendigen Restore–Aufrufe absetzt.

Fensterposition und –größe

SSWiS erlaubt der Applikation, die Position und die Größe ihrer Fenster zu beeinflussen. Jedoch müssen unter Umständen weitere Randbedingungen eingehalten werden, so daß SSWiS den Wünschen der Applikation nur teilweise nachkommen kann. Sowohl die Fensterposition, als auch die Fenstergröße (die Größe des Fensterinhaltes) können von der Applikation abfragt werden.

Parameter des Bildes

Die Größe des (Welt–) Bildes muß SSWiS bekannt sein, da sich daraus die Parameter der Scrollelemente ergeben. Alle Angaben über das Bild erfolgen in Welt–Koordinaten. Ein Raster kan angegeben werden, um das Inkrement beim Feinscrolling zu steuern.

Maus und Caret

Die Position der Maus wird durch ein Echo auf dem Bildschirm signalisiert. Das Echo kann verschiedene Formen (z.B. RubberLine) annehmen. Zudem ist i.a. noch ein sogenanntes Sprite sichtbar, also eine kleine Figur, die den augenblicklichen Standort der Maus anzeigt. Das Sprite und die Form des Echos sind von der Applikation einstellbar. Insbesondere kann eine applikationsabhängige Form gewählt werden. In diesem Fall ist die Applikation selbst dafür verantwortlich, das Echo zu erzeugen. Die Bewegungen der Maus werden dann über die Accept–Prozedur gemeldet. Echoform und Sprite können für jedes Fenster einer Applikation separat konfiguriert werden. Sie werden jeweils dann aktiviert, wenn das Fenster aktiv wird.

Das Caret ist eine zweite Markierung einer Position, ähnlich dem Mausecho, jedoch wird die Position des Caret ausschließlich durch die Applikation kontrolliert. Das Caret wird z.B. von TextWindws verwendet, um die aktuelle Cursorposition anzuzeigen.

Menüs

Jede Applikation kann bei SSWiS ein eigenes Menü bestehend aus bis zu 7 Titeln anmelden. Jeder Titel wird mit einer Beschriftung versehen. Jeder Titel kann bis zu 8 Einträge haben, die wieder jeweils mit einer Beschriftung versehen werden. Bei Menüeintragen können über die Beschriftung gleichzeitig Attribute des Eintrages konfiguriert werden. So kann ein Eintrag durch ein der Beschriftung vorangestelltes M maskiert werden. Es ist dann nicht möglich, den Eintrag zu selektieren. Ein C markiert den Eintrag bei GEM durch einen vorangestellten Haken. Menüeinträge und -titel können zu jeder Zeit von der Applikation wieder umkonfiguriert werden.

Formulare

Die Funktionalität von Formularsystemen ist so uneinheitlich, daß für SSWiS nur einige einfache Mechanismen vorgesehen werden, die aber doch recht universell verwendet werden können.

Ein Formulartyp dient dazu, eine Meldung auszugeben und eine von maximal 4 Antworten zu erhalten. Die für die Antwort zu konfigurierenden Buttons (virtuelle Tasten) werden als String angegeben. Die genaue Syntax ist im Definitionsmodul angegeben. Der Defaultbutton ergibt sich aus dem Eingangswert des Ergebnisses.

Ein weiteres Formular erlaubt, einen editierbaren Text und 4 zusätzliche, wählbare Optionen anzugeben. Die Optionen werden in der gleichen Syntax wie die Buttons konfiguriert. Jedoch liefern die Optionen ein BITSET als Ergebnis, da auch mehrere Optionen gleichzeitig gewählt werden können.

Das dritte Formular wird dazu benützt, die Applikation

zu identifizieren. Neben dem Program, der Version und dem Autor kann ein Copyright–Vermerk angegeben werden.

Diese Seite wurde aus satztechnischen Gründen freigelassen

Das Laufzeitsystem

Kapitel 9

Übersicht

Unter dem Begriff Laufzeitsystem werden die Funktionalitäten einer Sprachimplementierung zusammengefaßt, die den Ablauf von Programmen auf einer Hardware und einem Betriebssystem unterstützen, ohne daß sie für den Programmierer direkt sichtbar sind. Hierzu gehören u.a. die Repräsentierung von Datentypen als Bitmuster im Speicher der Maschine, die Verwaltung der diversen Speicherbereiche (Stack, Code, Daten, Konstanten) und Mechanismen zur Verwaltung des Speichers selbst.

Maschinenabhängigkeit

Das Laufzeitsystem ist damit hochgradig maschinenabhängig. Weiterhin sind die Funktionalitäten des Laufzeitsystems nicht als Modulschnittstelle (der SYSLIB) exportiert. Die im folgenden gegebenen Informationen haben deshalb hauptsächlich den Zweck, die interne Organisation von SPC MODULA-2 zu dokumentieren.

☞ Die an dieser Stelle gegegbenen Informationen sollen nicht in Programmen ausgenützt werden.

Falls sich die Organisation einmal ändern würde, wären solche Programme nicht mehr lauffähig. Weiterhin sind solche Programme fast nie portabel.

Es versteht sich von selbst, daß Programme niemals die Funktionalitäten des Laufzeitsystems stören dürfen.

Datentypen

Die Repräsentierung von MODULA-2 Datentypen in Bits, Bytes und Worten des Hauptspeichers ist durch den CPU-Typ bestimmt. Unterschiedliche Prozessoren bedingen i.a. unterschiedliche Repräsentierungen. Die hier gegebenen Informationen beziehen sich auf den CPU-Typ MOTOROLA MC68000 sowie auf seine kompatiblen Familienmitglieder. Programme, die sich auf die bitweise Repräsentierung der Datentypen beziehen, sind i.a. zwischen Implementierungen auf dem gleichen Prozessortyp portabel.

INTEGER

Der INTEGER Datentyp ist 16 Bit breit. Negative Werte werden im Zweierkomplement dargestellt. Der Zahlenberich reicht mithin von -32768 bis +32767. Die Datenbreite orientiert sich an der Breite des Maschinenwortes, also der Speichereinheit, die von arithmetischen und logischen Operationen standardmäßig unterstützt wird.

CARDINAL

Der CARDINAL Datentyp ist ebenfalls 16 Bit breit, jedoch umfaßt sein Wertebereich nur die positiven Zahlen von 0 bis 65535. Die Datenbreite orientiert sich am Maschinenwort.

CHAR

Der CHAR Datentyp ist 8 Bit breit und umfaßt die Werte von CHR(0) bis CHR(255), wobei die Bedeutung der Werte von CHR(0) bis CHR(127) durch den ASCII-Zeichensatz bestimmt ist.

BOOLEAN

Der BOOLEAN Datentyp ist ebenfalls 8 Bit breit. ORD(FALSE) ist 0, ORD(TRUE) ist 1. Bei der Auswertung von Ausdrücken werden jedoch alle von FALSE verschiedenen Werte als TRUE ausgewertet.

BITSET

Der BITSET Datentyp belegt ein Maschinenwort, hier 16 Bit, und ist als SET OF [0..15] definiert. Das Bitset 0 entspricht dem INTEGER Wert 1.

Enumerationen

Enumerationstypen (Aufzählungstypen) sind 8 Bit breit, d.h. ihr Wertebereich kann 256 Elemente umfassen.

SET OF

Set-Typen sind 16 oder 32 Bit breit, d.h. ihr Basistyp kann maximal 32 Elemente umfassen. Dabei entspricht der niederste Wert des Basistyps dem Bit 0 des Sets.

POINTER TO

Pointer-Typen sind 32 Bit breit. Der Wert NIL entspricht der Adresse 0, die i.a. zum Systembereich gehört und geschützt ist.

Alignment

Alle 8 Bit breiten Datentypen werden im Speicher auf Byte-Grenzen gelegt. Alle 16 und 32 Bit breiten Datentypen werden auf Wortgrenzen gelegt. Bei RECORD Datentypen können deshalb unbelegte "Löcher" entstehen. ARRAY Elemente mit höheren Indices sowie später deklarierte RECORD Elemente stehen im Speicher an höheren Adressen. Strukturierte Datentypen werden immer auf Wortgrenzen gelegt.

strukturierte Datentypen

REAL

Der REAL Datentyp ist 32 Bit breit. Das Format entspricht dem IEEE Standard (single precision). Die kleinste darstellbare, echt positive Zahl ist 3.3E-38, die größte darstellbare Zahl ist 3.3E38 (Dynamikbereich). Da die Mantisse nur 23 Bit breit ist, ist die Darstellungsgenauigkeit auf 4 bis 5 Dezimalziffern beschränkt. Die größte darstellbare, echt negative Zahl ist -3.3E-38, die kleinste darstellbare Zahl ist -3.3E38. Der Exponent hat einen sogenannten Versatz (engl. Bias) von 127, d.h. ein Wert von 128 entspricht einem Exponenten von 0. Negative Zahlen werden durch das gesetzte Vorzeichenbit gekennzeichnet. Da der REAL Datentyp vom MC68000 nicht direkt unterstützt wird, emuliert SYSTEM die notwendigen Instruktionen.

LONGINT

Der LONGINT Datentyp ist 32 Bit breit, negative Werte werden im Zweierkomplement dargestellt. Die Multiplikation und die Division stehen nicht als Maschineninstruktion zur Verfügung und werden deshalb von SYSTEM emuliert.

LONGCARD

Der LONGCARD Datentyp ist 32 Bit breit und umfaßt die positiven Zahlen von 0 bis 4294967295.

LONGBITSET

Der LONGBITSET Datentyp ist 32 Bit breit, und ist als SET OF [0..31] definiert. Das Set LONGBITSET 0 entspricht dem LONGINT-Wert 1.

LONGREAL

Der LONGREAL Datentyp ist 64 Bit breit. Sein Format entspricht dem IEEE Standard (double precision). Die Mantisse ist 52 Bit breit, der Exponent ist 11 Bit breit. Der Exponent hat einen Versatz von 1023. Die kleinste darstellbare, echt positive Zahl ist 1.79E-307, die größte darstellbare Zahl ist 1.79E308. Die größte darstellbare, echt negative Zahl ist -1.79E-307, die kleinste darstellbare Zahl ist -1.79E308.

Modul-organisation

In diesem Abschnitt wird die Organisation von Moduln im Hauptspeicher erklärt. Weitere Informationen finden sich im Definitionsmodul System, der für die Organisation zuständig ist, und die unten genannten Strukturen teilweise exportiert,

Threads

Jedes SPC MODULA-2 Programm besteht aus einem Hauptprogramm (-modul) und mehreren importierten-Moduln. Alle zu einem Programm gehörenden Modul werden ein Thread genannt. Selbst ein Standalone-Programm (gelinktes Programm, keine Shell) besteht aus mindestens einem Thread, kann aber, falls es den Loader importiert, auch mehrere haben. Threads sind numeriert. Die Nummer dient gleichzeitig als Kennung.

Moduln

Jeder SPC-MODULA-2 Modul besteht im Hauptspeicher aus mehreren Teilen. Die Einzelteile sind untereinander verzeigert. Für gebundene Programme (.PRG) wird die Verzeigerung vom Linker vorbereitet und vom Lader des Betriebssystems zur Ladezeit auf die aktuelle Ladeadresse bezogen (Loadtime Relocation). Der dynamische Linker (Loader) von SPC MODULA-2 kann die jeweils schon im Speicher vorhandene Struktur um weitere, nachzuladende Moduln erweitern.

Moduldescriptor

Jeder im Hauptspeicher vorhandene Modul ist durch einen Moduldeskriptor beschrieben. Die Struktur des-Moduldeskriptors wird vom Modul System exportiert. Der Moduldeskriptor enthält die folgenden Komponenten :

- Next, Zeiger auf den nächsten Deskriptor, letzter Zeiger ist NIL
- Frame, Zeiger auf das Code-Segment
- StaticBase, Zeiger auf das Daten-Segment
- ImportedMods, Anzahl der importierten Moduln
- ExportedProcs, Anzahl der exportierten Prozeduren
- CodeLength, Länge des Code-Segmentes
- DataLength, Länge des Datenbereichs

- ConstLength, Länge des Konstantenbereichs
- Thread, Nummer des Programms, zu dem der Modul gehört (0..15).
- References, Menge der geladenenen Programme, die den Modul importieren.
- reservierte Adresse

Hilfsdeskriptor

Desweiteren gibt es einen Hilfsdeskriptor (AuxDescr), der im Datenbereich des Moduls liegt und folgende Komponenten hat:

- Descr, Rückverweise auf den Deskriptor
- Name, Modulname
- Key, Modulschlüssel
- Flags, 0 falls Modul noch nicht initialisiert

Code-Segment

Das Code-Segment beginnt mit einem Zeiger auf das Daten-Segment (StaticBase), über den PC-relativ aus dem Code-Segment heraus das Datensegment erreicht werden kann. Danach folgt im Code-Segment der Code aller Prozeduren des Moduls, einschließlich der Initialisierungsprozedur.

Daten-Segment

Das Datensegment eines Moduls wird über einen Zeiger erreicht, der immer im Register A4 gehalten wird (StaticBase), solange eine Prozedur des Moduls ausgeführt wird. Jede exportierte Prozedur lädt beim Eintritt den Zeiger vom Anfang des Code-Segmentes (s.o.). Von A4 aus können dann mit positiven Offsets erreicht werden :

- Prozedurtabelle
- Modultabelle
- Modulkonstanten

sowie mit negativen Offsets :

- Hilfs-Deskriptor
- globale Modulvariablen

Prozedurtabelle

Die Prozedurtabelle enthält die Anfangsadressen aller exportierten Prozeduren, inklusive der Initialisierungsprozedur, die sich aus dem Hauptprogrammteil des Moduls ergibt. Die Prozedurtabelle wird vom Compiler aus dem Definitionsmodul abgeleitet. Prozeduren eines Moduls werden von außen über den Index in die Prozedurtabelle erreicht. Jedem importierenden Modul sind die Indizes aus dem importierten Definitionsmodul bekannt. Die absoluten Anfangsadressen werden vom Lader erst zur Ladezeit in die Tabelle eingetragen. Diesen Vorgang nennt man Fixup.

Modultablle

Die Modultabelle eines geladenen Moduls enthält die StaticBases der importierten Moduln. Über den Index des importierten Moduls gelangt man an die Datensegmente aller importierten Moduln und dabei insbesondere an deren Prozedurtabellen. Die StaticBases werden vom Lader zur Ladezeit eingetragen (Fixup).

Konstanten

Die Konstanten eines Moduls sind von der entsprechenden StaticBase mit positiven Offsets erreichbar. Sie liegen im Speicher über der Modultabelle.

Hilfsdeskriptor

Der Hilfsdeskriptor wurde schon oben erwähnt. Die Modulflags enthalten den Wert 0, falls der Modul noch nicht initialisiert wurde, sonst einen von 0 verschiedenen Wert. Hiermit wird verhindert, daß ein Modul mehrmals seine Initialisierungsprozedur durchläuft. Der Modulschlüssel ist für den Lader interessant. Aus ihm wird ermittelt, ob der zu ladende Modul zu allen geladenen Moduln paßt. Andernfalls wird der Fehler 'illegal module key' zurückgegeben. Der Modulname ist wichtig, um beim Nachladen von Moduln die importierten Moduln zu finden.

Modulvariablen

Die globalen Modulvariablen sind teils exportiert, teills sind sie nicht von außen sichtbar. Sie liegen deshalb am tiefsten unter der StaticBase.

Zusammenfassend läßt sich bemerken, daß die Teile, deren Ausdehnung von außen nicht bestimmbar ist

(namentlich die Konstanten und die Variablen), am oberen und unteren Ende des Datensegmentes angesiedelt sind. Die folgende Grafik veranschaulicht die genannten Verzeigerungen.

der Ladevorgang

Das dynamische Binden im SPC MODULA-2 Sprachsystem wird dadurch ermöglicht, daß nachzuladende Moduln mit den bereits geladenen Moduln verzeigert werden. Dieser Vorgang ist recht komplex und wird vom Lader bereitgestellt.

Modulliste aufbauen

Der Lader baut zunächst im Hauptspeicher eine Liste von Deskriptoren auf, an denen die dazugehörigen Daten- und Code-Segmente hängen. Dazu müssen die .OBM Dateien gelesen und interpretiert werden. Es ist zulässig, daß in einer Datei mehrere Moduln zusammengefaßt sind. Moduln, die aktuell nicht benötigt werden, werden vom Lader im Eingabestrom übersprungen. Verbindungen zu bereits geladenen Moduln stellt der Lader her, indem er sich von System einen Zeiger auf einen geladenen Modul geben läßt.

Thread starten

Wenn die neue Modulliste (Thread) komplett ist, wird sie System übergeben, um in die Liste der Threads eingehängt zu werden. Danach kann das Hauptprogramm gestartet werden.

Modulliste abbauen

Nach Beendigung des Programms werden die Moduln und Deskriptoren wieder aus dem Speicher entfernt, es sei denn, es ist die Hold-Option gesetzt. In diesem Fall bleiben die Moduln im Speicher. Es wird jedoch vermerkt. daß sie erneut initialisiert werden müssen, wenn später geladene Moduln darauf wieder Bezug nehmen.

gemeinsam benützte Datenbereiche

In diesem Konzept ist es möglich, daß verschiedene Programme die gleichen Moduln (insbesondere die gleichen Datenbestände) verwenden. Der Modul InOut wird z.B. von allen Moduln verwendet. Dadurch ist es auch möglich, eine Kommunikation zwischen verschiedenen Programmen herzustellen.

Stack-organisation

Datenbestände, die zur Abarbeitung von Prozeduren benötigt werden, werden wie üblich auf dem Stack angelegt. Der Stack wächst von höheren zu niedrigeren Addressen. Das Ende des Stacks (Top Of Stack) wird durch das Register A7 des MC68000 bezeichnet.

Prozedurrahmen

Für jede aufgerufene Prozedur wird auf dem Stack ein Datenbereich angelegt, der einem festen Aufbau folgt. Die darin enthaltenen Daten sind:

- die Parameter
- interne Verzeigerungen
- lokale Variablen der Prozedur
- Zwischenergebnisse

Der Aufruf von Prozeduren folgt Konventionen, die in bestimmten Fällen der systemnahen Programmierung von Interesse sein könnten. Sie werden deshalb im folgenden offen gelegt.

Parameter

Die Prozedurparameter werden beim Aufruf in der Reihenfolge ihrer Hinschreibung (von links nach rechts) auf den Stack gelegt. Der am weitesten links stehende Parameter kommt deshalb an der höchsten Adresse zu liegen. Die Größe von Datenstrukturen ist in vorangegangenen Abschnitten dokumentiert worden. Parameter, die nur ein Byte belegen, müssen auf dem Stack dennoch als Wortgröße abgelegt werden.

Rücksprung-Adresse

Beim Aufruf von Prozeduren wird im Anschluß an die Parameter die Rücksprungadresse auf den Stack gelegt und die Programmausführung mit dem ersten Befehl der Prozedur fortgesetzt. Die ersten Befehle einer jeden Prozedur gehören zum sogenannten Prozedur-Prolog und werden vom Compiler automatisch generiert.

globale vs. lokale Prozeduren

Dabei wird unterschieden, ob eine Prozedur auf der Modulebene, also der äußersten Sichtbarkeitsebene eines Moduls deklariert ist, oder ob sie innerhalb einer anderen Prozedur erklärt wurde.

Im ersten Fall wird durch den Prolog zunächst der alte Wert von A4 (Zeiger auf das Datensegment, Static-Base) auf den Stack gerettet und der Zeiger auf das Datensegment des die Prozedur enthaltenden Moduls gesetzt. Dieser wird vom Anfang des Code-Segmentes des Moduls beschafft.

FramePointer

In beiden Fällen (globale und lokale Prozedur) wird anschließend einheitlich weiter verfahren. Als nächstes wird der alte Wert des Registers A6 auf den Stack gerettet, der Inhalt des Stackpointers in das Register A6 übernommen und der Stackpointer um soviele Worte weitergesetzt (zu tieferen Adressen), wie die lokalen Variablen der Prozedur in Anspruch nehmen. Das Register A6 zeigt nun in den sogenannten Stack-Frame der Prozedur und wird deshalb auch Frame-Pointer genannt. Der oben beschriebene Aufbau wird durch die Instruktion LINK des Prozessors unterstützt. Von A6 aus sind jetzt mit positiven Offsets zugänglich:

- die Parameter
- die Rücksprungadresse (ReturnAddress)
- der alte Wert von A4 (OldStaticBase)
- der alte Wert von A6 (OldFramePointer)

sowie mit negativen Offsets:

- die lokalen Variablen

Der Stackpointer zeigt unter die lokalen Variablen, sodaß der weitere Aufbau des Stacks den Stack-Frame nicht zerstört.

Prozedur verlassen

Beim Verlassen der Prozedur wird zunächst der alte Wert von A6 wieder in das Register übernommen und der Stackpointer auf die Adresse oberhalb des alten A6-Wertes gesetzt (UNLINK). Falls es sich um eine globale Prozedur handelt, wird danach der alte Wert von A4 wieder in das Register übernommen. In beiden Fälle wird die Rücksprungadresse in das Register A0 übernommen und der Stackpointer über die Parameter gesetzt. Die Prozedur korrigiert also selbst den Stack

um die Größe der Parameter (im Gegensatz zu C). Der Rücksprung erfolgt über das Register A0.

Prolog und Epilog

Der erzeugte Code geht davon aus, daß über den Prozeduraufruf die Register A4, A5 und A6 erhalten bleiben. Weiter wird davon ausgegangen, daß die gerufene Prozedur die Parameter vom Stack nimmt.

Zur Konkretisierung werden im folgenden die Codesequenzen für den Prozedurein- und -austritt angegeben:

```
global:
MOVE.L    A4,-(SP)
MOVE.L    -d(PC),A4
LINK      A6,#ls

; Prozedurrumpf

UNLK      A6
MOVE.L    (SP)+,A4
MOVE.L    (SP)+,A0
ADDQ.L    #ps,SP
JMP       (a0)
```

```
lokal:
; entfällt, da A4
; schon geladen ist
LINK      A6,#ls

; Prozedurrumpf

UNLK      A6
; entfällt
MOVE.L    (SP)+,A0
ADDQ.L    #ps,SP
JMP       (a0)
```

Hierbei bedeuten d der Abstand zum Anfang des Code-Segmentes, ls die Größe der lokalen Variablen und ps die Größe der Parameter.

Ein Funktionsergebnis wird als 0-ter Parameter behandelt. Es ist also ein enstprechend großer Bereich über der Parametern der Prozedur vorhanden (2 oder 4 Bytes).

Compiler Fehlermeldungen

Anhang A

Syntax

10 Identifier erwartet

11 Komma (,) erwartet

12 Semikolon (;) erwartet

13 Doppelpunkt (:) erwartet

14 Punkt (.) erwartet

15 schließende Klammer ()) erwartet

16 schließende eckige Klammer (]) erwartet

17 schließende geschweifte Klammer ( ) erwartet

18 Gleichheitszeichen (=) erwartet

19 Zuweisung (:=) erwartet

20 END erwartet

21 Ellipse (..) erwartet

22 öffnende Klammer (() erwartet

23 OF erwartet

24 TO erwartet

25 DO erwartet

26 UNTIL erwartet

27 THEN erwartet

28 MODULE erwartet

29 unzulässige Ziffer oder Zahl zu groß

	30	IMPORT erwartet
	31	Faktor beginnt mit unzulässigem Symbol
	32	Identifier, (oder [erwartet
	33	Identifier, ARRAY, RECORD, SET, POINTER, PROCEDURE, (oder [erwartet
	34	Type wird von unzulässigem Symbol gefolgt
	35	Statement beginnt mit unzulässigem Symbol
	36	Deklaration wird von unzulässigem Symbol gefolgt
	37	Statement Teil ist in DEFINITION MODULE nicht erlaubt
	38	Exportliste ist in IMPLEMENTATION MODULE nicht erlaubt
	39	EXIT ist nur innerhalb von LOOP Statements erlaubt
	40	unzulässiges Zeichen in einer Zahl
	41	Zahl ist zu groß
	42	Schließende Kommentarklammer *) fehlt
	44	Ausdruck darf nur konstante Operanden enthalten
	45	Steuerzeichen innerhalb eines Strings
nicht definiert	50	Identifier ist nicht erklärt oder nicht sichtbar
Klasse und Typ	51	Objekt sollte eine Konstante sein
	52	Objekt sollte ein Typ sein

53 Objekt sollte eine Variable sein

54 Objekt sollte eine Prozedur sein

55 Objekt sollte ein Modul sein

56 Objekt sollte ein Unterbereich sein

57 Objekt sollte ein RECORD sein

58 Objekt sollte ein ARRAY sein

59 Objekt sollte ein SET sein

60 unzulässiger Basistyp eines SETs

61 unzulässiger Typ einer Marke oder Unterbereichsgrenze

62 mehr definierte CASE Marke

63 untere Grenze ist größer als obere Grenze

64 mehr aktuelle als formale Parameter

65 weniger aktuelle als formale Parameter

66 mehr Parameter im IMPLEMENTATION MODULE als im DEFINITION MODULE

67 Parameter mit gleichen Typen im IMPLEMENTATION MODULE haben unterschiedliche Typen im DEFINITION MODULE

68 Diskrepanz zwischen VAR Spezifikationen

69 Diskrepanz zwischen Typ Spezifikationen

70 mehr Parameter in DEFINITION MODULE als in IMPLEMENTATION MODULE

71 Diskrepanz zwischen Ergebnistyp-Spezifikationen

72 Funktion im DEFINITION MODULE, normale Prozedur im IMPLEMENTATION MODULE

73 Prozedur hat im DEFINITION MODULE Parameter, jedoch nicht im IMPLEMENTATION MODULE

74 CODE Prozeduren dürfen nicht im DEFINITION MODULE deklariert werden

75 unzulässiger Typ der Kontrollvariable eines FOR Statements

76 Prozeduraufruf einer Function

77 Identifier in Kopf und nach END passen nicht zusammen

78 neuerliche Definition eines Typs, der im DEFINITION MODULE deklariert wurde

79 importierter Modul konnte nicht gefunden werden

80 offener EXPORT Listeneintrag

81 unzulässiger Typ des Prozedurergebnisses

82 unzulässiger Basistyp eines Unterbereichs

83 unzulässiger Typ eines CASE Ausdruckes

84 Ausgabe der Symboldatei (.SBM) war nicht erfolgreich

85 Schlüssel der importierten Moduln sind nicht konsistent

86 fehlerhaftes Format der Symboldatei (.SBM)

88 Symboldatei konnte nicht geöffnet werden

89 Prozedur ist im DEFINITION MODULE erklärt, aber nicht im IMPLEMENTATION MODULE

90	in der Konstruktion a..b muß, falls a eine Konstante ist, b ebenfalls eine Konstante sein	Implementierung
91	Überlauf des Identifier Puffers	
92	zu viele CASEs	
93	zu viele EXIT Statements	
94	Indextyp eines ARRAYs muß ein Unterbereich sein	
95	Unterbereichsgrenzen müssen kleiner 32535sein	
96	zu viele globale Moduln	
97	zu viele Prozeduren im DEFINITION MODULE	
98	zu viele strukturierten Elemente im DEFINITION MODULE	
100	mehrfache Definition innerhalb des gleichen Sichtbarkeitsbereichs	mehrfache Definition
101	unzulässige Verwendung eines Typs	Inkompatibilität
102	unzulässige Verwendung einer Prozedur	
103	unzulässige Verwendung einer Konstanten	
104	unzulässige Verwendung eines Typs	
105	unzulässige Verwendung einer Prozedur	
106	unzulässige Verwendung eines Ausdrucks	
107	unzulässige Verwendung eines Moduls	
108	konstanter Index außerhalb des zulässigen Bereichs	

109 indizierte Variable ist kein ARRAY oder der Index hat den falschen Typ

110 RECORD Selektor ist kein Feld Identifier

111 dereferenzierte Variable ist kein Pointer

112 der Typ des Operanden ist inkompatibel mit dem Operator Vorzeichenumkehr (–)

113 der Typ des Operanden ist inkompatibel mit dem Operator NOT

114 in dem Ausdruck x IN y ist der Typ von x verschieden vom Basistyp von y

115 in dem Ausdruck x IN y ist der Typ von x kein Basistyp eines SETs oder y ist kein SET

116 in dem Ausdruck a..b ist a oder b verschieden vom Basistyp des SETs

117 inkompatible Operandentypen

118 der Typ des Operanden ist nicht kompatibel mit dem Operator *

119 der Typ des Operanden ist nicht kompatibel mit dem Operator /

120 der Typ des Operanden ist nicht kompatibel mit dem Operator DIV

121 der Typ des Operanden ist nicht kompatibel mit dem Operator MOD

122 der Typ des Operanden ist nicht kompatibel mit dem Operator AND

123 der Typ des Operanden ist nicht kompatibel mit dem Operator +

124 der Typ des Operanden ist nicht kompatibel mit dem Operator –

125 der Typ des Operanden ist nicht kompatibel mit dem Operator OR

126 der Typ des Operanden ist nicht kompatibel mit einem Vergleichsoperator

127 die Prozedur muß auf Ebene 0(Modulebene) deklariert werden

128 der Ergebnistyp einer Prozedur paßt nicht zum Ergebnistyp eines Prozedurtyps

129 ein Parameter einer Prozedur paßt nicht zur formalen Typenliste eines Pozedurtyps

130 eine Prozedur hat weniger Parameter als der Prozedurtyp

131 eine Prozedur hat mehr Parameter als der Prozedurtyp

132 Zuweisung einer negativen Zahl an eine CARDINAL Variable

133 inkompatible Zuweisung

134 Zuweisung an ein Objekt, das keine Variable ist

135 Typ des Ausdrucks in IF, WHILE oder UNTIL muß BOOLEAN sein

136 Aufruf eines Objektes, das keine Prozedur ist

137 der Typ des VAR Parameters ist nicht kompatibel mit dem des aktuellen Parameters

139 der Typ des Ausdrucks nach RETURN ist verschieden vom Ergebnistyp der Prozedur

140 unzulässiger Typ eines CASE Ausdrucks

141 die Schrittweite eines FOR Statements darf nicht 0 sein

	142	unzulässiger Typ einer Kontrollvariablen
	143	Zuweisung an ein dynamisches Array ist unzulässig
	144	unzulässiger Typ eines Parameters einer Standardprozedur
	145	dieser Parameter sollte ein Typ Identifier sein
	146	der String ist zu lang
	147	unzulässige Spezifikation einer Priorität
Namenskollision	150	ein exportierter Identifier kollidiert mit einem bereits deklarierten Identifier
System	201	Integer ist zu negativ für Vorzeichenumkehr
	202	ein SET Element ist außerhalb der Wortlänge
	203	Überlauf in einer Multiplikation
	204	Überlauf in einer Division
	205	Division durch 0oder Modulo mit negativem Wert
	206	Überlauf in einer Addition
	207	Überlauf in einer Subtraktion
	208	der CARDINAL Wert, der einer INTEGER Variablen zugewiesen wird, ist zu groß
	209	die Größe eines SETs ist zu groß
	210	die Größe eines ARRAYs ist zu groß
	212	eine Komponente eines Character ARRAYs kann kein VAR Parameter werden
	214	Elemente eines SETs müssen Konstanten sein

215 der Ausdruck ist zu komplex

222 eine Ausgabedatei konnte nicht geöffnet werden (Directory voll?)

223 eine Ausgabedatei konnte nicht vollständig geschrieben werden (Diskette voll?)

224 zu viele externe Referenzen

225 zu viele Strings

226 das Programm ist zu lang

Maschine

230 der Ausdruck ist nicht ladbar

231 der Ausdruck ist nicht addressierbar

232 der Ausdruck ist nicht zulässig

233 der Ausdruck ist nicht zulässig

234 Fehler bei der Registerzuteilung

235 unzulässiger Selektor

236 zu viele geschachtelte WITH Statements (mehr als 4)

237 unzulässiger Operand

238 unzulässige Operandengröße

239 Typ sollte LONGREAL sein

240 der Parameter sollte ein dynamisches Array sein

241 unzulässiger Typ einer Gleitkomma-Operation

244 unzulässiger Gleitkomma-Vergleich

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

MODULA–2 Syntax

Anhang B

```
                ident = letter {letter | digit}.

               number = integer | real.

              integer = digit {digit} ["D"] | octalDigit
                        {octalDigit} ("B"|"C") | digit {hexDigit}
                        "H".

                 real = digit {digit} "." {digit} [ScaleFactor].

          ScaleFactor = "E" ["+"|"–"] digit {digit}.

             hexDigit = digit |"A"|"B"|"C"|"D"|"E"|"F".

                digit = octalDigit | "8"|"9".

           octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".

               string = "'" {character} "'" | '"' {character} '"'.

            qualident = ident {"." ident}.

      ConstDeclaration = ident "=" ConstExpression.

     ConstExpression = expression.

     TypeDeclaration = ident "=" type.

                type = SimpleType | ArrayType | RecordType
                       | SetType | PointerType |
                       ProcedureType.

          SimpleType = qualident | enumeration |
                       SubrangeType.

         enumeration = "(" IdentList ")".
```

IdentList = ident {"," ident}.

SubrangeType = [qualident] "[" ConstExpression ".." ConstExpression "]".

ArrayType = **ARRAY** SimpleType {"," SimpleType} **OF** type.

RecordType = **RECORD** FieldListSequence **END**.

FieldListSequence = FieldList {";" FieldList}.

FieldList = [IdentList ":" type | **CASE** [ident] ":" qualident **OF** variant {"|" variant} [**ELSE** FieldListSequence] **END**].

variant = [CaseLabelList ":" FieldListSequence].

CaseLabelList = CaseLabels {"," CaseLabels}.

CaseLabels = ConstExpression [".." ConstExpression].

SetType = **SET OF** SimpleType.

PointerType = **POINTER TO** type.

ProcedureType = **PROCEDURE** [FormalTypeList].

FormalTypeList = "(" [[**VAR**] FormalType {"," [**VAR**] FormalType}] ")" [":" qualident].

VariableDeclaration = IdentList ":" type.

designator = qualident {"." ident | "[" ExpList "]" | "^"}.

ExpList = Expression {"," expression}.

expression = SimpleExpression [relation SimpleExpression].

relation = "=" | "#" | "<" | "<=" | ">" | ">=" | **IN**

```
SimpleExpression = ["+"|"-"] term {AddOperator term}.

AddOperator = "+" | "-" | OR .

term = factor {MulOperator factor}.

MulOperator = "*" | "/" | DIV | REM | MOD | AND |
              "&" .

factor = number | string | set | designator
         [ActualParameters] | "(" expression ")"
         | NOT factor | "~" factor.

set = [qualident] "{" [element {"," element}]
      "}".

element = ConstExpression [".."
          ConstExpression].

ActualParameters = "(" [ExpList] ")" .

statement = assignment | ProcedureCall |
            IfStatement | CaseStatement |
            WhileStatement | RepeatStatement |
            LoopStatement | ForStatement |
            WithStatement | EXIT | RETURN
            [expression] ].

assignment = designator ":=" expression.

ProcedureCall = designator [ActualParameters].

StatementSequence = statement {";" statement}.

IfStatement = IF expression THEN
              StatementSequence {ELSIF
              expression THEN
              StatementSequence} [ELSE
              StatementSequence] END.

CaseStatement = CASE expression OF case {"|" case}
                [ELSE StatementSequence] END.
```

```
                case = [CaseLabelList ":"
                       StatementSequence].

      WhileStatement = WHILE expression DO
                       StatementSequence END.

     RepeatStatement = REPEAT StatementSequence UNTIL
                       expression.

        ForStatement = FOR ident ":=" expression TO
                       expression [BY ConstExpression] DO
                       StatementSequence END.

       LoopStatement = LOOP StatementSequence END.

        WithStatement = WITH designator DO
                       StatementSequence END .

ProcedureDeclaration = ProcedureHeading ";" (block ident |
                       FORWARD).

   ProcedureHeading = PROCEDURE ident
                       [FormalParameters].

               block = {declaration} [BEGIN
                       StatementSequence] END.

         declaration = CONST {ConstantDeclaration ";"} |
                       TYPE {TypeDeclaration ";"} | VAR
                       {VariableDeclaration ";"} |
                       ProcedureDeclaration ";" |
                       ModuleDeclaration ";".

  FormalParameters = "(" [FPSection {";" FPSection}] ")" [":"
                       qualident].

         FPSection = [VAR] IdentList ":" FormalType.

        FormalType = [ARRAY OF] qualident.

 ModuleDeclaration = MODULE ident [priority] ";" {import}
                       [export] block ident.

          priority = "[" ConstExpression "]".
```

```
export =           EXPORT [QUALIFIED] IdentList ";".

import =           [FROM ident] IMPORT IdentList ";".

DefinitionModule = DEFINITION MODULE ident ";"
                   {import} {definition} END ident ".".

definition =       CONST {ConstantDeclaration ";"} |
                   TYPE {ident ["=" type] ";"} | VAR
                   {VariableDeclaration ";"} |
                   ProcedureHeading ";" .

ProgramModule =    MODULE ident [priority] ";" {import}
                   block ident "." .

CompilationUnit =  DefinitionModule | [IMPLEMENTATION]
                   ProgramModule.
```

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Literaturhinweise

Anhang C

Niklaus Wirth

"Programming in MODULA-2", Third Corrected Edition

Springer, 1985, ISBN 0-387-15078-1

Dieses Werk, vom Erfinder der Sprache selbst verfaßt, dient als Standardwerk und als Referenz über MODULA-2. Es erklärt in kurzer und prägnanter Form die Syntax und die Semantik von MODULA-2. Bis zur Erreichung einer Normung durch das BSI werden sich Diskussionen über die Sprache an diesem Werk orientieren. Die deutsche Übersetzung ist ebenfalls erhältlich. Nicht zuletzt wegen des relativ geringen Preises gehört es in das Bücherregal jedes ernsthaften MODULA-2 Programmierers. Allerdings muß eingeschränkt werden, daß es zum Erlernen des Programmierens an sich nicht geeignet ist.

Herbert Schildt

MODULA-2 Made Easy

McGraw-Hill, 1986, ISBN 0-07-881241-0

Was N.Wirth in seinem Standardwerk vermissen läßt, nämlich eine leicht verständliche Einführung für Programmierneulinge ist Herbert Schildt in hervorragender Weise gelungen. Das Werk ist etwas umfänglicher, ohne die Übersicht zu erschweren. Besonders der Anfänger wird zu schätzen wissen, daß inzwischen auch eine deutsche Fassung im Handel ist. Das Buch ist reichlich mit Beispielen versehen und enthält einige Übungsaufgaben, die langsam

aber sicher an des Besondere von MODULA-2 heranführen. Eine ideale Ergänzung zum SPC Sprachsystem.

ATARI Corp.

ATARI ST Bedienungshandbuch

ATARI Corp., 1985

Jankowski, Reschke, Rabich

ATARI ST Profibuch

SYBEX, 1987, ISBN 3-88745-501-0

Wer auf dem ATARI ST Computer programmieren will, kommt ohne zusätzliche Literartur nicht aus. Unter den Büchern über den ATARI ST und sein Betriebssystem, die inzwischen im Handel sind, ist diese Werk eines der umfassendsten. Die Darstellung der Betriebssystemschnittstellen sind präzise und sowohl in C- als auch in Assembler-Notation angegeben. Das Buch kann durchaus zwei oder mehrere andere Bücher zum gleichen Thema ersetzen.

Hilf, Nausch

M68000 Familie, Teil 1, Grundlagen und Architektur

Te-Wi, 1984

Der Verfasser ist als Mitarbeiter der Firma MOTOROLA ein intimer Kenner der Architektur des 68000 und der dazugehörigen

Bausteine. Wer in die Assemblerprogrammierung einsteigen will, oder sich an die elementaren Fähigkeiten des Chip herantrauen will, wird an diesem Buch nicht vorbeikommen. Für den Einstieg in MODULA-2 sind die Details der Hardware jedoch weniger interessant, sodaß man im ersten Moment auf die Anschaffung verzichten kann.

Jürgen Geiß, Dieter Geiß

Software-Entwicklung auf dem ATARI ST

Hüthig, 1986, ISBN 3-7785-1339-7

Im Gegensatz zum Profibuch geht es in diesem als Taschenbuch ausgeführten Werk ausschließlich um die Entwicklung von Software für den ST. Dazu wird auf einige Aspekte der Programmierung der GEM-Oberfläche genauer eingegangen und einigen Beispielen die anzuwendenden Techniken vorgeführt. Daß die Autoren auf diesem Gebiet fachkundig sind, haben sie schon eindrucksvoll vorgeführt.

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

Beispielprogramme

Anhang D

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

```
MODULE  Hello;

(* Codesize is limited about 1k Bytes within the demo version. *)

FROM    InOut  IMPORT   WriteString, WriteInt, WriteLn;
FROM    SYSTEM IMPORT   VAL, ADDRESS;

VAR     x     : ARRAY [1..10] OF CHAR;
        c     : CHAR;
        p     : POINTER TO CHAR;  999


PROCEDURE Wait;

VAR     i, j : INTEGER;

BEGIN
  FOR j:=-10000 TO MAX(INTEGER) DO
    i:= 1;
  END;
END Wait;


PROCEDURE Count         (    Number     : INTEGER);

VAR     i : INTEGER;

BEGIN
  WriteInt (Number, 2); WriteLn;
  c:= x[Number];
  Wait;
  IF Number = 0 THEN RETURN ELSE Count (Number-1) END;
END Count;



BEGIN
  WriteLn; WriteString ('Hello World'); WriteLn;

  (* To demo the debugger compile this program giving options : *)
  (* compile hello.mod r o                                      *)
  p:= VAL(ADDRESS,99);
  WriteString ('Count Down'); WriteLn;
  Count (10);

END Hello.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) MODULE
(* .      Name          : *) Empty;
(* .      Function      : Empty Test Programm (minimal PRG size)        *)
(* .      Version  Date : 21: 4  16. 1.1989                             *)
(* Authors              : R.Huetter                                     *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1989, Ronald Huetter, D7500 Karlsruhe     *)


(*----- Category : Types and Data ----------------------------------------*)

IMPORT Terminal;


VAR     ch : CHAR;

BEGIN
  REPEAT
    Terminal.WriteString ('Hello World');
    Terminal.WriteLn;
    Terminal.BusyRead (ch);
  UNTIL ch # 0C;

  Terminal.WriteString ('got ');
  Terminal.Write       (ch);
  Terminal.WriteLn;

  Terminal.Read (ch);
  Terminal.WriteString ('got ');
  Terminal.Write       (ch);

END Empty.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) MODULE
(* .      Name          : *) Dump;
(* .      Function      : File Dump Utility                              *)
(* .      Version/Date  : 1.0  28.12.87                                  *)
(* Authors              : R.Huetter                                      *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987, Ronald Huetter, D7500 Karlsruhe      *)


(*----- Category : Types and Data -------------------------------------*)

FROM    SYSTEM  IMPORT  LONG, SHIFT, VAL, ADDRESS;
FROM    InOut   IMPORT  WriteString, WriteLn, WriteInt;

IMPORT  ByteStreams, CmdLine, Environment, HFS, Strings, TextStreams, XStr;


CONST   Version         = 'Dump V1.0';


VAR     i, j, k, kBytes : INTEGER;
        ch              : CHAR;
        Name, s         : ARRAY [0..80] OF CHAR;
        bs              : ByteStreams.Streams;
        ts              : TextStreams.Streams;
        a               : ADDRESS;
        v, f, d, t      : ARRAY [0..60] OF CHAR;
        asc             : ARRAY [0..15] OF CHAR;


(*----- Category : Main Program ---------------------------------------*)

BEGIN
  CmdLine.UtilityName (s);

  IF NOT CmdLine.FileArg (Name)
  THEN CmdLine.ResultIs (FALSE, 'usage : Dump <filename>');
       RETURN;
  END;

  ByteStreams.Open (bs, Name, ByteStreams.FileIn);
  IF bs.Result = ByteStreams.Done
  THEN HFS.Decode (Name, v,f,d,t);
       HFS.Encode (v,f,d,'.LST', s);
       TextStreams.Open (ts, s, TextStreams.FileOut);
       IF ts.Result # TextStreams.Done
       THEN CmdLine.ResultIs (FALSE, 'cannot open list file');
            RETURN;
       END;
  ELSE CmdLine.ResultIs (FALSE, 'file not found');
       RETURN;
  END;


  i:= 0; j:= 0; a:= 0; kBytes:= 1000;

  LOOP
    IF j = 256
    THEN TextStreams.WriteLn (ts);
         j:= 0;
         DEC (kBytes); IF kBytes <= 0 THEN EXIT END;
    END;
```

```
    IF j = 0
    THEN TextStreams.WriteLn (ts);
         TextStreams.WriteString (ts, '          ');
         FOR i:= 0 TO 15 DO
           IF (i MOD 8) = 0 THEN TextStreams.Write (ts, ' ') END;
           TextStreams.WriteHex (ts, i, 3);
         END;
         TextStreams.WriteLn (ts);
    END;

    IF (j MOD 16) = 0
    THEN TextStreams.WriteLn (ts);
         TextStreams.WriteAddress (ts, a, 8);
         TextStreams.WriteString (ts, '  ');
    ELSIF (j MOD 8) = 0
    THEN TextStreams.Write (ts, ' ');
    END;


    ByteStreams.ReadByte (bs, ch);
    IF bs.Result # ByteStreams.Done
    THEN TextStreams.WriteLn (ts);
         EXIT;
    END;

    TextStreams.WriteHex (ts, ORD(ch),3);
    IF (ch >= ' ') & (ch <= 'z')
    THEN asc[j MOD 16]:= ch;
    ELSE asc[j MOD 16]:= '.';
    END;

    INC (j); INC (a);

    IF (j MOD 16) = 0
    THEN TextStreams.WriteString (ts, '   ');
         TextStreams.WriteString (ts, asc);
    END;
  END;

  TextStreams.Close (ts);
  ByteStreams.Close (bs);

  CmdLine.ResultIs (FALSE, 'done');
END Dump.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) IMPLEMENTATION MODULE
(* .      Name          : *) GemDos;
(* .      Function      : GEMDOS Interface to Modula-2                  *)
(* .      Version  Date :  0:38   2. 6.1988                             *)
(* Authors              : R.Huetter                                     *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, Ronald Huetter, D7500 Karlsruhe     *)


(*----- Category : Types and Data -------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS, INLINE, ADR, VAL, REG, SHIFT, SHORT;

CONST   Trap            = 04E41H;
        AddSP           = 0DFFCH; (* HighCount, LowCount *)


(*----- Category : GemDos Interface -----------------------------------*)

PROCEDURE None          (   Code        : INTEGER); CODE Trap;

PROCEDURE I             (   p1          : INTEGER;
                            Code        : INTEGER); CODE Trap;

PROCEDURE C             (   p1          : INTEGER;
                            Code        : INTEGER); CODE Trap;

PROCEDURE A             (   p1          : ADDRESS;
                            Code        : INTEGER); CODE Trap;

PROCEDURE L             (   p1          : LONGINT;
                            Code        : INTEGER); CODE Trap;

PROCEDURE IL            (   p1          : INTEGER;
                            p2          : LONGINT;
                            Code        : INTEGER); CODE Trap;

PROCEDURE IA            (   p1          : INTEGER;
                            p2          : ADDRESS;
                            Code        : INTEGER); CODE Trap;

PROCEDURE ALT           (   p1          : ADDRESS;
                            p2          : LONGINT;
                            p3          : INTEGER;
                            Code        : INTEGER); CODE Trap;

PROCEDURE IIL           (   p1, p2      : INTEGER;
                            p3          : LONGINT;
                            Code        : INTEGER); CODE Trap;

PROCEDURE IIA           (   p1, p2      : INTEGER;
                            p3          : ADDRESS;
                            Code        : INTEGER); CODE Trap;

PROCEDURE II            (   p1, p2      : INTEGER;
                            Code        : INTEGER); CODE Trap;

PROCEDURE LA            (   p1          : LONGINT;
                            p2          : ADDRESS;
                            Code        : INTEGER); CODE Trap;

PROCEDURE AAAI          (   p1, p2, p3  : ADDRESS;
                            p4          : INTEGER;
```

```
                                Code        : INTEGER); CODE Trap;

PROCEDURE AA                (   p1, p2      : ADDRESS;
                                Code        : INTEGER); CODE Trap;

PROCEDURE AAI               (   p1, p2      : ADDRESS;
                                p3          : INTEGER;
                                Code        : INTEGER); CODE Trap;

PROCEDURE ALI               (   p1          : ADDRESS;
                                p2          : LONGINT;
                                p3          : INTEGER;
                                Code        : INTEGER); CODE Trap;

PROCEDURE IAI               (   p1          : INTEGER;
                                p2          : ADDRESS;
                                p3          : INTEGER;
                                Code        : INTEGER); CODE Trap;


(*----- Category : GemDos Functions ------------------------------------*)

PROCEDURE Term0;

BEGIN
  None (0); INLINE (AddSP, 0,2);
  Result:= REG(0);
END Term0;


PROCEDURE ConIn             (VAR Ch         : CHAR;
                             VAR ScanCode   : ScanCodes);

BEGIN
  None (1); INLINE (AddSP, 0,2);
  Result:= REG(0);
  Ch       := CHR(Result MOD 256D);
  ScanCode:= SHIFT(Result,-16) MOD 256D;
END ConIn;


PROCEDURE ConOut            (   Ch          : CHAR);

BEGIN
  C (ORD(Ch), 2); INLINE (AddSP, 0,4);
  Result:= REG(0);
END ConOut;


PROCEDURE AuxIn             (VAR Ch         : CHAR);

BEGIN
  None (3); INLINE (AddSP, 0,2);
  Result:= REG(0);
  Ch:= CHR(Result MOD 256D);
END AuxIn;


PROCEDURE AuxOut            (   Ch          : CHAR);

BEGIN
  C (ORD(Ch), 4); INLINE (AddSP, 0,4);
  Result:= REG(0);
END AuxOut;
```

```
PROCEDURE PrnOut        (    Ch          : CHAR);

BEGIN
  C (ORD(Ch), 5); INLINE (AddSP, 0,4);
  Result:= REG(0);
END PrnOut;


PROCEDURE ConRawIO      (VAR Ch          : CHAR);

BEGIN
  C (ORD(Ch), 6); INLINE (AddSP, 0,4);
  Result:= REG(0);
  IF ORD(Ch) = 0FFH
  THEN
    Ch:= CHR(Result MOD 256D);
  END;
END ConRawIO;


PROCEDURE ConRawIn      (VAR Ch          : CHAR);

BEGIN
  None (7); INLINE (AddSP, 0,2);
  Result:= REG(0);
  Ch:= CHR(Result MOD 256D);
END ConRawIn;


PROCEDURE ConNegIn      (VAR Ch          : CHAR);

BEGIN
  None (8); INLINE (AddSP, 0,2);
  Result:= REG(0);
  Ch:= CHR(Result MOD 256D);
END ConNegIn;


PROCEDURE ConWriteString(    Line        : ARRAY OF CHAR);

BEGIN
  A (ADR(Line), 9); INLINE (AddSP, 0,6);
  Result:= REG(0);
END ConWriteString;


PROCEDURE ConReadString (VAR Line        : ARRAY OF CHAR);

BEGIN
  A (ADR(Line), 10); INLINE (AddSP, 0,6);
  Result:= REG(0);
END ConReadString;


PROCEDURE ConInStat     () : BOOLEAN;

BEGIN
  None (11); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN Result # 0D;
END ConInStat;


PROCEDURE SetDrv        (    Drive       : Drives)
```

```
                        : SetOfDrives;

BEGIN
  I (Drive, 14); INLINE (AddSP, 0,4);
  Result:= REG(0);
  RETURN VAL(SetOfDrives, Result);
END SetDrv;


PROCEDURE ConOutStat    () : BOOLEAN;

BEGIN
  None (16); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN TRUE;
END ConOutStat;


PROCEDURE PrnOutStat    () : BOOLEAN;

BEGIN
  None (17); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN Result = -1D;
END PrnOutStat;


PROCEDURE AuxInStat     () : BOOLEAN;

BEGIN
  None (18); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN Result = -1D;
END AuxInStat;


PROCEDURE AuxOutStat    () : BOOLEAN;

BEGIN
  None (19); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN Result = -1D;
END AuxOutStat;


PROCEDURE GetDrive      () : Drives;

BEGIN
  None (25); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN VAL(Drives, Result);
END GetDrive;


PROCEDURE SetDTA        (VAR Dta        : DTA);

BEGIN
  A (ADR(Dta), 26); INLINE (AddSP, 0,6);
  Result:= REG(0);
END SetDTA;


PROCEDURE Super         (VAR Stck       : LONGINT);

BEGIN
```

```
  A (ADR(Stck), 32); INLINE (AddSP, 0,6);
  Result:= REG(0);
END Super;


PROCEDURE GetDate       (VAR Today      : DosDate);

BEGIN
  None (42); INLINE (AddSP, 0,2);
  Result:= REG(0);
  Today := Result;
END GetDate;


PROCEDURE SetDate       (    Today      : DosDate);

BEGIN
  I (Today, 43); INLINE (AddSP, 0,4);
  Result:= REG(0);
END SetDate;


PROCEDURE GetTime       (VAR Now        : DosTime);

BEGIN
  None (44); INLINE (AddSP, 0,2);
  Result:= REG(0);
  Now:= Result;
END GetTime;


PROCEDURE SetTime       (    Now        : DosTime);

BEGIN
  I (Now, 45); INLINE (AddSP, 0,4);
  Result:= REG(0);
END SetTime;


PROCEDURE GetDTA        () : DTAPtr;

BEGIN
  None (47); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN VAL(ADDRESS, Result);
END GetDTA;


PROCEDURE Version       () : CARDINAL;

BEGIN
  None (48); INLINE (AddSP, 0,2);
  Result:= REG(0);
  RETURN VAL(CARDINAL, Result);
END Version;


PROCEDURE TermResident  (    Memory     : LONGCARD;
                             Return     : INTEGER);

BEGIN
  IL (Return, Memory, 49); INLINE (AddSP, 0,8);
  Result:= REG(0);
END TermResident;
```

```
PROCEDURE DiskFree      (VAR Info       : DiskInfo;
                             Drive      : Drives);

BEGIN
  IA (Drive, ADR (Info), 54); INLINE (AddSP, 0,8);
  Result:= REG(0);
END DiskFree;


PROCEDURE DirCreate     (    Name       : ARRAY OF CHAR);

BEGIN
  A (ADR(Name), 57); INLINE (AddSP, 0,6);
  Result:= REG(0);
END DirCreate;


PROCEDURE DirDelete     (    Name       : ARRAY OF CHAR);

BEGIN
  A (ADR(Name), 58); INLINE (AddSP, 0,6);
  Result:= REG(0);
END DirDelete;


PROCEDURE SetPath       (    Name       : ARRAY OF CHAR);

BEGIN
  A (ADR(Name), 59); INLINE (AddSP, 0,6);
  Result:= REG(0);
END SetPath;


PROCEDURE Create        (    Name       : ARRAY OF CHAR;
                             Attribute  : SetOfAttributes)
                        : Handles;

BEGIN
  IA (VAL(INTEGER, Attribute), ADR(Name), 60);
  INLINE (AddSP, 0,8);
  Result:= REG(0);
  RETURN SHORT(Result);
END Create;


PROCEDURE Open          (    Name       : ARRAY OF CHAR;
                             Mode       : OpenModes)
                        : Handles;

BEGIN
  IA (ORD(Mode), ADR(Name), 61); INLINE (AddSP, 0,8);
  Result:= REG(0);
  RETURN SHORT(Result);
END Open;


PROCEDURE Close         (    Handle     : Handles);

BEGIN
  I (Handle, 62); INLINE (AddSP, 0,4);
  Result:= REG(0);
END Close;
```

```
PROCEDURE Read          (    Handle     : Handles;
                             Buffer     : ADDRESS;
                             Size       : LONGINT)
                        : LONGINT;

BEGIN
  ALI (ADR(Buffer), Size, Handle, 63); INLINE (AddSP, 0,12);
  Result:= REG(0);
  RETURN Result;
END Read;


PROCEDURE Write         (    Handle     : Handles;
                             Buffer     : ADDRESS;
                             Size       : LONGINT)
                        : LONGINT;

BEGIN
  ALI (ADR(Buffer), Size, Handle, 64); INLINE (AddSP, 0,12);
  Result:= REG(0);
  RETURN Result;
END Write;


PROCEDURE Delete        (    Name       : ARRAY OF CHAR);

BEGIN
  A (ADR(Name), 65); INLINE (AddSP, 0,6);
  Result:= REG(0);
END Delete;


PROCEDURE Seek          (    Handle     : Handles;
                             Mode       : SeekModes;
                             Position   : LONGINT);

BEGIN
  IIL (ORD(Mode), Handle, Position, 66); INLINE (AddSP, 0,10);
  Result:= REG(0);
END Seek;


PROCEDURE Attribute     (    Name       : ARRAY OF CHAR;
                             Mode       : GetModes;
                         VAR Attrib     : SetOfAttributes);

BEGIN
  IIA (VAL(INTEGER, Attrib), ORD(Mode), ADR(Name), 67);
  INLINE (AddSP, 0,10);
  Result:= REG(0);
  Attrib:= VAL(SetOfAttributes, Result);
END Attribute;


PROCEDURE Dup           (    StdHandle  : Handles)
                        : Handles;

BEGIN
  I (StdHandle, 69); INLINE (AddSP, 0,4);
  Result:= REG(0);
  RETURN SHORT(Result);
END Dup;


PROCEDURE Force         (    StdHandle  : Handles;
```

```
                                NonStdHndl : Handles);

BEGIN
  II (StdHandle, NonStdHndl, 70); INLINE (AddSP, 0,6);
  Result:= REG(0);
END Force;


PROCEDURE GetPath       (VAR Path       : Paths;
                             Drive      : Drives);

BEGIN
  IA (Drive, ADR(Path), 71); INLINE (AddSP, 0,8);
  Result:= REG(0);
END GetPath;


PROCEDURE MemAlloc      (    Amount     : LONGINT)
                        : LONGINT;

BEGIN
  L (Amount, 72); INLINE (AddSP, 0,6);
  Result:= REG(0);
  RETURN Result;
END MemAlloc;


PROCEDURE MemFree       (    Block      : ADDRESS);

BEGIN
  A (Block, 73); INLINE (AddSP, 0,6);
  Result:= REG(0);
END MemFree;


PROCEDURE Shrink        (    Block      : ADDRESS;
                             Size       : LONGINT);

BEGIN
  LA (Size, Block, 74); INLINE (AddSP, 0,10);
  Result:= REG(0);

END Shrink;


PROCEDURE Exec          (    Mode       : LoadModes;
                             Path       : ARRAY OF CHAR;
                             CmdLine    : ARRAY OF CHAR;
                             Environment: ARRAY OF CHAR)
                        : LONGINT;

BEGIN
  AAAI (ADR(Environment), ADR(CmdLine), ADR(Path), ORD(Mode), 75);
  INLINE (AddSP, 0,16);
  Result:= REG(0);
  RETURN Result;
END Exec;


PROCEDURE Term          (    Return     : INTEGER);

BEGIN
  I (Return, 76); INLINE (AddSP, 0,4);
  Result:= REG(0);
END Term;
```

```
PROCEDURE SearchFirst   (    Spec       : ARRAY OF CHAR;
                             Attr       : SetOfAttributes);

BEGIN
  IA (VAL(INTEGER, Attr), ADR(Spec), 78); INLINE (AddSP, 0,8);
  Result:= REG(0);
END SearchFirst;


PROCEDURE SearchNext;

BEGIN
  None (79); INLINE (AddSP, 0,2);
  Result:= REG(0);
END SearchNext;


PROCEDURE Rename        (    OldName    : ARRAY OF CHAR;
                             NewName    : ARRAY OF CHAR);

BEGIN
  AAI (ADR(NewName), ADR(OldName), 0, 86); INLINE (AddSP, 0,12);
  Result:= REG(0);
END Rename;


PROCEDURE Timestamp     (VAR DatTim     : FileTimes;
                             Handle     : Handles;
                             Mode       : GetModes);

VAR     i : INTEGER;

BEGIN
  IF Mode = Set THEN i:= 1 ELSE i:= 0 END;
  IIA (i, Handle, ADR(DatTim), 87);
  INLINE (AddSP, 0,10);
  Result:= REG(0);
END Timestamp;


END GemDos.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) MODULE
(* .      Name          : *) SetEnv;
(* .      Function      : Maintain Environment Variable                 *)
(* .      Version/Date  : 1.00  27.1.88                                 *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract ---------------------------------------**

  Set, Modify, Delete and List environment variables. *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    InOut             IMPORT WriteString, WriteLn, Write,
                                 ReadString, Read, ReadInt, ReadLn;
IMPORT  ASCII, Environment, Strings;

CONST   Version         = 'SetEnv V1.0';

VAR     VarName         : ARRAY [0..30] OF CHAR;
        TempStr         : ARRAY [0..80] OF CHAR;
        i               : INTEGER;
        ch              : CHAR;


(*----- Category : Commands ----------------------------------------------*)

PROCEDURE Set;

BEGIN
  WriteString ('set variable : '); ReadString (VarName);
  WriteLn;
  WriteString ('current value : ');

  IF Environment.Get (VarName, TempStr)
  THEN WriteString (TempStr);
  ELSE WriteString ('<empty, new will be allocated>');
  END;
  WriteLn;

  WriteString ('new or <ret>  : '); ReadLn (TempStr);
  IF TempStr[0] # 0C
  THEN WriteLn;
       Environment.Set (VarName, TempStr);
       WriteString ('changed       : '); WriteString (TempStr);
  ELSE WriteString ('no changes');
  END;
  WriteLn;
END Set;


PROCEDURE Delete;

VAR     TmpName : ARRAY [0..30] OF CHAR;

BEGIN
  WriteString ('del variable : '); ReadString (VarName);
  WriteLn;
  WriteString ('current value : ');

  IF Environment.Get (VarName, TempStr)
```

```
  THEN WriteString (TempStr);
       WriteLn;
       WriteString ('y to confirm  : '); Read (ch);
       WriteLn;
       IF CAP(ch) = 'Y'
       THEN Environment.Set (VarName, '');
            WriteString ('deleted       : ');
            WriteString (VarName);
       END;
  ELSE WriteString ('not found     : ');
       WriteString (VarName);
  END;
  WriteLn;
END Delete;


PROCEDURE List;

VAR     VarName, String : ARRAY [0..80] OF CHAR;
        i               : INTEGER;

BEGIN
  i:= 1;
  WHILE Environment.GetIndexed (i, VarName, String) DO
    WriteString (VarName);
    WriteString (' = ');
    WriteString (String);
    WriteLn;
    INC (i);
  END;
END List;


(*----- Category : Main Program ----------------------------------------*)

BEGIN
  WriteString (Version); WriteLn; WriteLn;

  LOOP
    WriteString ('SetEnv [L] ');
    Read (ch); IF ch = ASCII.EOL THEN ch:= 'L' END;
    Write (' ');

    CASE CAP(ch) OF
    | 'S' : Set;
    | 'D' : Delete;
    | 'L' : List;
    | 'Q' : WriteLn; WriteLn;
            EXIT;
    | ' ' : ;
    | 'H' : WriteString ('commands are: ');
            WriteLn;
            WriteString ('L_ist   to list all variables');
            WriteLn;
            WriteString ('S_et    to set a new or to modify an existing variable');
            WriteLn;
            WriteString ('D_elete to delete a variable');
            WriteLn;
            WriteString ('Q_uit   to return to Shell');
            WriteLn;
    | ELSE  WriteString ('unknown command, type H for help');
            WriteLn;
    END;

    WriteLn;
```

```
  END;
END SetEnv.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) MODULE
(* .      Name          : *) Shell;
(* .      Function      : Standard Command Interface for VERSAdos       *)
(* .      Version/Date  : 1.25 1.1.88                                   *)
(* Authors              : R.Huetter                                     *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Types and Data ---------------------------------------*)

FROM    SYSTEM  IMPORT  LONG;
FROM    InOut   IMPORT  Done, Write, WriteString, WriteLn,
                        WriteInt, WriteCard, WriteOct, WriteHex,
                        Read, ReadCard, ReadInt, ReadString, ReadLn;
IMPORT  ASCII, Strings, Loader, CmdLine;


CONST   Version         = 'Shell V1.40';

VAR     ch              : CHAR;
        ExitShell       : BOOLEAN;


(*----- Category : Program Call Primitives -----------------------------*)

PROCEDURE Call          (    Tool       : ARRAY OF CHAR;
                             Files      : BOOLEAN);

VAR     s, msg  : ARRAY [0..132] OF CHAR;
        b       : BOOLEAN;

BEGIN
  Strings.Copy   (Tool, 1,99, msg);
  WriteString (msg); WriteString (' ');

  Strings.Concat (Tool, '  ', s);
  IF Files
  THEN ReadLn (msg); Strings.Concat (s, msg, s);
  END;

  WriteLn;
  CmdLine.Set (s); CmdLine.ResultIs (TRUE, 'no message');

  IF NOT Loader.Call (Tool, FALSE, msg)
  THEN WriteString ('Loader: '); WriteString (msg); WriteLn;
  ELSE CmdLine.Result (b, msg);
       WriteString (msg); WriteLn;
  END;
END Call;


PROCEDURE Run;

VAR     s, msg  : ARRAY [0..132] OF CHAR;
        b       : BOOLEAN;
        Tool    : ARRAY [0..40] OF CHAR;

BEGIN
  WriteString ('un ');
  ReadLn (s);  WriteLn;

  CmdLine.Set (s); CmdLine.ResultIs (TRUE, 'no message');
```

```
  CmdLine.UtilityName (Tool);

  IF NOT Loader.Call (Tool, FALSE, msg)
  THEN WriteString ('Loader: '); WriteString (msg); WriteLn;
  ELSE CmdLine.Result (b, msg);
       WriteString (msg); WriteLn;
  END;
END Run;


PROCEDURE Quit () : BOOLEAN;

BEGIN
  WriteString ('You are about to QUIT SHELL'); WriteLn;
  WriteString ('    Type Q again to confirm');
  Read (ch);
  RETURN CAP(ch) = 'Q';
END Quit;


PROCEDURE Help;

BEGIN
  WriteLn;
  WriteString (Version); WriteLn;
  WriteString ('Commands are : '     ); WriteLn;
  WriteString ('  C ... Compile'     ); WriteLn;
  WriteString ('  L ... Link'        ); WriteLn;
  WriteString ('  R ... Run'         ); WriteLn;
  WriteString ('  P ... Prelink'     ); WriteLn;
  WriteString ('  X ... Run Domain'  ); WriteLn;
  WriteString ('  Q ... Quit'        ); WriteLn;
END Help;


(*----- Category : Main Loop ----------------------------------------*)

BEGIN
  ExitShell:= FALSE;

  WriteLn; WriteString (Version); WriteLn;

  REPEAT
    WriteString ('spc: '); Read (ch);

    CASE CAP(ch) OF
    | 'C' : Call ('compile', TRUE);
    | 'H' : Help;
    | 'P' : Call ('prelink', TRUE);
    | 'L' : Call ('link',    TRUE);
    | 'X' : Call ('domain',  FALSE);
    | 'R' : Run;
    | 'Q' : ExitShell:= Quit();
    | '?' : Help;
    ELSE    WriteString ('unknown command, type H for HELP');
            WriteLn;
    END;
  UNTIL ExitShell;

  WriteLn;
END Shell.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) IMPLEMENTATION MODULE
(* .      Name          : *) Terminal;
(* .      Function      : Window Based Standard Terminal              *)
(* .      Version  Date : 12:59  22. 1.1989                           *)
(* Authors              : R.Huetter                                   *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987, Ronald Huetter, D7500 Karlsruhe   *)


(*----- Category : Types and Data -------------------------------------*)

FROM    SYSTEM
IMPORT  SHORT;
IMPORT  SSWiS, ASCII, Environment, TextFiles, TextWindows, XStr, System;

CONST   ModuleName      = 'Terminal';

VAR     Script          : TextFiles.File;
        HotLine         : TextFiles.Text;
        HotLineLen      : INTEGER;
        HotLinePos      : INTEGER;
        InpLine         : ARRAY [0..40] OF CHAR;
        InpLineLen      : INTEGER;
        InpLinePos      : INTEGER;
        Client          : SSWiS.ModuleHandles;
        Digits          : ARRAY [0..15] OF CHAR;


(*----- Category : Utility Level --------------------------------------*)

PROCEDURE Update;

VAR     XY, WH : TextWindows.Points;

BEGIN
  IF HotLineLen > HotLinePos
  THEN XY.X:= HotLinePos;
       XY.Y:= TextFiles.TotalLinesOf (Script);
       WH.X:= HotLineLen-HotLinePos+1;
       WH.Y:= 1;
       TextWindows.ExplicitRestore (Client, 0, XY, WH);
  END;
END Update;


(*----- Category : Input and Output -----------------------------------*)

PROCEDURE Read          (VAR ch         : CHAR);

VAR     p : TextWindows.Points;

BEGIN
  IF InpLinePos >= InpLineLen
  THEN
    Expose; Update;

    p.X:= HotLineLen;
    p.Y:= TextFiles.TotalLinesOf (Script);
    TextWindows.SetCaret (Client, 0, p);
    InpLinePos:= 0;
    InpLineLen:= 0;

    WHILE InpLineLen = 0 DO
```

```
      SSWiS.PollEvents;
    END;

    p.X:= -1;
    TextWindows.SetCaret (Client, 0, p);
  END;
  ch:= InpLine[InpLinePos]; INC(InpLinePos);
END Read;


PROCEDURE BusyRead      (VAR ch         : CHAR);

VAR     p : TextWindows.Points;

BEGIN
  IF InpLinePos >= InpLineLen
  THEN
    Update;
    p.X:= HotLineLen;
    p.Y:= TextFiles.TotalLinesOf (Script);
    TextWindows.SetCaret (Client, 0, p);
    InpLinePos:= 0;
    InpLineLen:= 0;

    SSWiS.PollEvents;

    p.X:= -1;
    TextWindows.SetCaret (Client, 0, p);
  END;
  IF InpLinePos < InpLineLen
  THEN ch:= InpLine[InpLinePos]; INC(InpLinePos);
  ELSE ch:= 0C;
  END;
END BusyRead;


PROCEDURE Write         (    ch         : CHAR);

VAR     s       : ARRAY [0..0] OF CHAR;
        i       : INTEGER;
        XY, WH : TextWindows.Points;

BEGIN
  IF (ch = 10C) & (HotLineLen > 0)
  THEN DEC (HotLineLen);
       WriteString ('');

  ELSIF ch = ASCII.CR
  THEN WriteLn;

  ELSIF ch = ASCII.FF
  THEN i:= TextFiles.TotalLinesOf (Script);
       WHILE i > 0 DO
         TextFiles.Position (Script, i);
         TextFiles.Delete   (Script);
         DEC (i);
       END;
       HotLine[0]:= 0C; HotLineLen:= 0; HotLinePos:= 0;

       WH.X:= 100; WH.Y:= 100; XY.X:= 0; XY.Y:= 0;
       TextWindows.ExplicitRestore (Client, 0, XY,WH);

       WH.X:= 100; WH.Y:= 1; XY.X:= 0; XY.Y:= 0;
       TextWindows.SizeWorld       (Client, 0, WH);
       TextWindows.PositionWorld   (Client, 0, XY);
```

```
  ELSIF ch >= ' '
  THEN s[0]:= ch;
       WriteString (s);
  END;
END Write;


PROCEDURE WriteString   (    Text       : ARRAY OF CHAR);

VAR     i, j   : INTEGER;
        XY, WH : TextWindows.Points;

BEGIN
  j:= 0;
  WHILE (j <= HIGH(Text)) & (HotLineLen < HIGH(HotLine)) & (Text[j] # 0C) DO
    HotLine[HotLineLen]:= Text[j];
    INC (HotLineLen); INC (j);
  END;
  HotLine[HotLineLen]:= 0C;
END WriteString;


PROCEDURE WriteLn;

VAR     LastLine  : TextWindows.Coordinates;
        i         : INTEGER;
        XY, WH    : TextWindows.Points;
        Done      : TextFiles.Results;

BEGIN
  HotLineLen:= 999; Update;

  LastLine:= TextFiles.TotalLinesOf (Script);
  TextFiles.Position (Script, 999);
  TextFiles.Insert   (Script, HotLine, Done);
  HotLine[0]:= 0C;
  HotLineLen:= 0;
  HotLinePos:= 0;

  TextWindows.WorldOf    (Client, 0, XY,WH);
  TextWindows.InteriorOf (Client, 0, WH);

  IF LastLine >= XY.Y+WH.Y-3
  THEN
    IF LastLine > 140
    THEN WHILE LastLine > 100 DO
           TextFiles.Position (Script, 1);
           TextFiles.Delete   (Script);
           DEC (LastLine);
         END;
    END;

    XY.X:= 0; XY.Y:= LastLine-WH.Y+3;
    TextWindows.PositionWorld (Client, 0, XY);
  END;
END WriteLn;


PROCEDURE WriteLong     (    Arg        : LONGINT;
                             Length     : CARDINAL);

CONST   Base     = 16D;
        BaseChar = 'H';
```

```
VAR     b, d     : ARRAY [0..30] OF CHAR;
        i, j     : INTEGER;
        a        : LONGINT;

BEGIN
  d[0]:= ' ';
  IF Arg < 0D
  THEN d[1]:= '-';
       a:= -Arg;
       j:= 2;
  ELSE a:= Arg;
       j:= 1;
  END;
  i:= 0;
  REPEAT
    b[i]:= Digits[SHORT(a MOD Base)];
    a   := a DIV Base;
    INC (i);
  UNTIL a = 0D;
  WHILE i > 0 DO
    DEC (i); d[j]:= b[i]; INC (j);
  END;
  d[j]:= BaseChar; INC (j);
  d[j]:= 0C;
  WriteString (d);
END WriteLong;


(*----- Category : Window Driven Terminals -----------------------------*)

PROCEDURE Expose;

BEGIN
  SSWiS.PlaceWindowOnTop (Client, 0);
END Expose;


PROCEDURE Hide;

VAR     XY, WH  : SSWiS.ScreenPoints;
        TempStr : ARRAY [0..60] OF CHAR;
        j       : CARDINAL;

BEGIN
  TempStr:= ''; j:= 0;
  SSWiS.PositionOfWindow    (Client, 0, XY);
  SSWiS.SizeOfWindowContent (Client, 0, WH);

  XStr.Integer (XY.X , 5, 10, TempStr, j);
  XStr.Integer (XY.Y , 5, 10, TempStr, j);
  XStr.Integer (WH.X , 5, 10, TempStr, j);
  XStr.Integer (WH.Y , 5, 10, TempStr, j);
  XStr.Char    (0C,           TempStr, j);

  Environment.Set   ('TERMINALFLAGS', TempStr);

  SSWiS.IconiseWindow (Client, 0);
END Hide;


(*----- Category : Operating the Window --------------------------------*)

PROCEDURE AcceptEvent   (    Owner      : SSWiS.ModuleHandles;
                             Window     : SSWiS.WindowHandles;
                         VAR Report     : SSWiS.EventReports);
```

```
VAR     i, j   : INTEGER;
        XY, WH : TextWindows.Points;

BEGIN
  WITH Report DO
    IF Type = SSWiS.Keyboard
    THEN
      i:= 0;
      LOOP
        j:= Strokes.Keys[i];
        IF (j < 0) OR (InpLineLen >= HIGH(InpLine)) THEN EXIT END;
        IF j > 256
        THEN CASE j OF
             | SSWiS.NumLeftBracket : j:= ORD('(');
             | SSWiS.NumRightBracket: j:= ORD(')');
             | SSWiS.NumSlash       : j:= ORD('/');
             | SSWiS.NumAsterisk    : j:= ORD('*');
             | SSWiS.NumMinus       : j:= ORD('-');
             | SSWiS.NumPlus        : j:= ORD('+');
             | SSWiS.NumEnter       : j:= ORD(ASCII.CR);
             | SSWiS.NumDot         : j:= ORD('.');
             | SSWiS.Num0           : j:= ORD('0');
             | SSWiS.Num1           : j:= ORD('1');
             | SSWiS.Num2           : j:= ORD('2');
             | SSWiS.Num3           : j:= ORD('3');
             | SSWiS.Num4           : j:= ORD('4');
             | SSWiS.Num5           : j:= ORD('5');
             | SSWiS.Num6           : j:= ORD('6');
             | SSWiS.Num7           : j:= ORD('7');
             | SSWiS.Num8           : j:= ORD('8');
             | SSWiS.Num9           : j:= ORD('9');
             | ELSE
             END;
        END;
        IF j < 256
        THEN InpLine[InpLineLen]:= CHR(j);
             INC (InpLineLen);
        END;
        INC (i);
      END;

    ELSIF Type = SSWiS.Timer
    THEN Update;

    ELSIF Type = SSWiS.Identification
    THEN
      SSWiS.Identify ('SSWiS', '', '', '');
    END;
  END;
END AcceptEvent;


PROCEDURE Restore       (    Owner      : SSWiS.ModuleHandles;
                             Window     : SSWiS.WindowHandles;
                             XY, WH     : TextWindows.Points);

VAR     i, j, k: INTEGER;
        p      : TextFiles.TextPtr;

BEGIN
  XY.X:= 0;
  TextWindows.Position (XY);
  j:= TextFiles.TotalLinesOf (Script);
```

```
  WHILE WH.Y >= 0 DO
    IF XY.Y < j
    THEN TextFiles.Position  (Script, XY.Y+1);
         p:= TextFiles.PointerOf (Script);
         TextWindows.WriteString (p^);
    ELSIF XY.Y = j
    THEN TextWindows.WriteString (HotLine);
    END;
    TextWindows.WriteLn;
    INC (XY.Y); DEC (WH.Y);
  END;
  TextWindows.Clear;

  HotLinePos:= HotLineLen;
END Restore;


(*----- Category : Initialisation ---------------------------------------*)

PROCEDURE Init;

VAR     Done     : TextFiles.Results;
        TempStr  : ARRAY [0..60] OF CHAR;
        j        : CARDINAL;
        tp       : TextWindows.Points;
        XY, WH   : SSWiS.ScreenPoints;
        min      : SSWiS.ScreenPoints;
        b        : BOOLEAN;

BEGIN
  TextFiles.Create (Script, Done);
  HotLine[0]:= 0C;
  HotLineLen:= 0;
  InpLine[0]:= 0C;
  InpLinePos:= 0;
  InpLinePos:= 0;
  tp.X      := 100;
  tp.Y      := 150;

  SSWiS.Register            (Client, ModuleName, AcceptEvent);
  TextWindows.Create        (Client, 0, Restore);
  TextWindows.SizeWorld     (Client, 0, tp);
  SSWiS.SetWindowTitle      (Client, 0, 'Terminal');
  SSWiS.SetWindowElements   (Client, 0, SSWiS.SetOfWindowElements{
                                        SSWiS.Iconiser..SSWiS.YScroller});

  XY.X:= 20; XY.Y:= 20; WH.X:= 400; WH.Y:= 200; j:= 0;
  IF Environment.Get ('TERMINALFLAGS', TempStr)
  THEN
    b:= XStr.InvInteger (XY.X, TempStr, j);
    b:= XStr.InvInteger (XY.Y, TempStr, j);
    b:= XStr.InvInteger (WH.X, TempStr, j);
    b:= XStr.InvInteger (WH.Y, TempStr, j);
  END;
  SSWiS.PositionWindow    (Client, 0, XY);
  min.X:= 100; min.Y:= 100;
  SSWiS.SizeWindowContent (Client, 0, min, WH, SSWiS.ScreenSize);
  SSWiS.PlaceWindowOnTop  (Client, 0);
  SSWiS.Resync;
END Init;


PROCEDURE Term;

BEGIN
```

```
  Hide;

  TextWindows.Delete (Client, 0);
END Term;


BEGIN
  Digits:= '0123456789ABCDEF';

  Init;  IF System.OnModuleTerminationDo (Term) THEN END;
END Terminal.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) IMPLEMENTATION MODULE
(* .      Name          : *) Terminal;
(* .      Function      : Terminal Based Standard Terminal              *)
(* .      Version  Date : 21:16  16. 1.1989                             *)
(* Authors              : R.Huetter                                     *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1989, Ronald Huetter, D7500 Karlsruhe     *)


(*----- Category : Implementation Notes ---------------------------------**

  Der folgende Modul stellt eine alternative Implementierung des meist
  benoetigten Terminals zur Verfuegung. Die Groesse des fertigen Programms
  kann drastisch gesenkt werden, falls nicht aus anderen Gruenden auf
  SSWiS zurueckgegriffen werden muss.

  Um den Modul zu verwenden, kann er einfach gegen den normalen Modul
  TERMINAL.OBM ausgetauscht werden.

*)

(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM
IMPORT  SHORT;
IMPORT  ASCII, GemDos, System;

VAR     Digits          : ARRAY [0..15] OF CHAR;


(*----- Category : Input and Output --------------------------------------*)

PROCEDURE Read          (VAR ch         : CHAR);

BEGIN
  GemDos.ConNegIn (ch);
END Read;


PROCEDURE BusyRead      (VAR ch         : CHAR);

BEGIN
  IF GemDos.ConInStat ()
  THEN Read (ch);
  ELSE ch:= 0C;
  END;
END BusyRead;


PROCEDURE Write         (    ch         : CHAR);

VAR     s : ARRAY [0..1] OF CHAR;

BEGIN
  s[0]:= ch; s[1]:= 0C;
  WriteString (s);
END Write;


PROCEDURE WriteString   (    Text       : ARRAY OF CHAR);

BEGIN
  GemDos.ConWriteString (Text);
```

```
END WriteString;


PROCEDURE WriteLn;

VAR     s : ARRAY [0..2] OF CHAR;

BEGIN
  s[0]:= ASCII.CR; s[1]:= ASCII.LF; s[2]:= 0C;
  WriteString (s);
END WriteLn;


PROCEDURE WriteLong     (    Arg        : LONGINT;
                             Length     : CARDINAL);

CONST   Base     = 16D;
        BaseChar = 'H';

VAR     b, d     : ARRAY [0..30] OF CHAR;
        i, j     : INTEGER;
        a        : LONGINT;

BEGIN
  d[0]:= ' ';
  IF Arg < 0D
  THEN d[1]:= '-';
       a:= -Arg;
       j:= 2;
  ELSE a:= Arg;
       j:= 1;
  END;
  i:= 0;
  REPEAT
    b[i]:= Digits[SHORT(a MOD Base)];
    a   := a DIV Base;
    INC (i);
  UNTIL a = 0D;
  WHILE i > 0 DO
    DEC (i); d[j]:= b[i]; INC (j);
  END;
  d[j]:= BaseChar; INC (j);
  d[j]:= 0C;
  WriteString (d);
END WriteLong;


(*----- Category : Window Driven Terminals ----------------------------*)

PROCEDURE Expose;

BEGIN

END Expose;


PROCEDURE Hide;

BEGIN

END Hide;


(*----- Category : Initialisation -------------------------------------*)
```

```
BEGIN
  Digits:= '0123456789ABCDEF';
END Terminal.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) IMPLEMENTATION MODULE
(* .      Name          : *) Watch;
(* .      Function      : SPC Desktop Clock                              *)
(* .      Version  Date : 21:35  11. 1.1989                              *)
(* Authors              : R.Huetter                                      *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe        *)


(*----- Category : Types and Data -------------------------------------*)

FROM    SYSTEM          IMPORT LONG, SHORT;
FROM    MathLib         IMPORT sin, cos;
IMPORT  Clock, SSWiS, XStr, System,
        AESGraphics, VDIAttributes, VDIControls, VDIOutputs;


CONST   HalfSize        = 36;
        Size            = 2*HalfSize;

VAR     Client          : SSWiS.ModuleHandles;
        VDIHandle       : INTEGER;
        Sin, Cos        : ARRAY [0..59] OF REAL;
        Time            : Clock.Time;
        FacePat         : ARRAY [0..11] OF VDIOutputs.Coordinate;
        Day             : CARDINAL;


(*----- Category : Utility Level --------------------------------------*)

PROCEDURE OpenVirtWorkstation ()
                        : INTEGER;

VAR     In  : VDIControls.WorkstationInitRec;
        Out : VDIControls.WorkstationDescription;
        i   : INTEGER;

BEGIN
  WITH In DO
    DeviceId        := 1;
    LineStyle       := VDIAttributes.Solid;
    LineColour      := 1;
    MarkerType      := VDIAttributes.Dot;
    MarkerColour    := 1;
    Font            := VDIAttributes.BigFont;
    TextColour      := 1;
    FillStyle       := VDIAttributes.Filled;
    FillIndex       := 1;
    FillColour      := 1;
    CoordinateSystem:= VDIAttributes.RasterCoords;
  END;

  i:= AESGraphics.Handle (i,i,i,i);
  VDIControls.OpenVirtualWorkstation (In, i, Out);
  RETURN i;
END OpenVirtWorkstation;


(*----- Category : Restore Proc ---------------------------------------*)

PROCEDURE Restore       (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                            WorldArea   : SSWiS.Lines;
```

```
                                Offset     : SSWiS.Points);

VAR     Clip   : VDIOutputs.VDIRectangle;
        t      : Clock.DecodedTime;
        i      : INTEGER;
        j, f   : CARDINAL;
        hs     : LONGINT;
        Finger : ARRAY [0.. 2] OF VDIOutputs.Coordinate;
        Face   : ARRAY [0..11] OF VDIOutputs.Coordinate;
        s      : ARRAY [0..20] OF CHAR;

BEGIN
  Clip.LowerLeft.x := WorldArea.A.X + Offset.X;
  Clip.LowerLeft.y := WorldArea.A.Y + Offset.Y;
  Clip.UpperRight.x:= WorldArea.A.X + Offset.X + WorldArea.B.X-1D;
  Clip.UpperRight.y:= WorldArea.A.Y + Offset.Y + WorldArea.B.Y-1D;
  VDIControls.SetClipping (VDIHandle, TRUE, Clip);

  VDIOutputs.FillRectangle (VDIHandle, Clip);

  Clock.Decode (Time, t);
  hs:= HalfSize;

  IF t.Day # Day
  THEN j:= 0; Day:= t.Day;
       XStr.Cardinal (t.Day  , 1,10, s,j);
       XStr.Char     ('.'          , s,j);
       XStr.Cardinal (t.Month, 1,10, s,j);
       XStr.Char     ('.'          , s,j);
       XStr.Char     (0C           , s,j);
       SSWiS.SetWindowTitle (Client, 0, s);
  END;

  FOR i:= 0 TO 11 DO
    WITH Face[i] DO
      x:= SHORT(Offset.X) + FacePat[i].x;
      y:= SHORT(Offset.Y) + FacePat[i].y;
    END;
  END;
  VDIOutputs.PolyMarker (VDIHandle, 12, Face);

  WITH Finger[1] DO
    x:= Offset.X + hs;
    y:= Offset.Y + hs;
  END;
  WITH Finger[0] DO
    x:=  TRUNC(Sin[t.Minute]*FLOAT(HalfSize-8 )) + Finger[1].x;
    y:= -TRUNC(Cos[t.Minute]*FLOAT(HalfSize-8 )) + Finger[1].y;
  END;
  WITH Finger[2] DO
    i:= ((t.Hour MOD 12) * 60 + t.Minute) DIV 12;
    x:=  TRUNC(Sin[i]*FLOAT(HalfSize-16)) + Finger[1].x;
    y:= -TRUNC(Cos[i]*FLOAT(HalfSize-16)) + Finger[1].y;
  END;
  VDIOutputs.PolyLine (VDIHandle, 3, Finger);
END Restore;


PROCEDURE Accept        (    Owner      : SSWiS.ModuleHandles;
                             Window     : SSWiS.WindowHandles;
                         VAR Report     : SSWiS.EventReports);

VAR     t : Clock.Time;

BEGIN
```

```
  Clock.Get (t);
  IF t.Millisec >= Time.Millisec+60000D
  THEN Time:= t;
       SSWiS.ExplicitRestore (Client, 0, SSWiS.NeverClip);
  END;
END Accept;


(*----- Category : Initialisation -----------------------------------*)

PROCEDURE Init;

VAR     i, j, k, l : INTEGER;
        Style      : VDIAttributes.FillStyles;
        Color      : VDIAttributes.ColourRange;
        xy, wh     : SSWiS.ScreenPoints;
        x          : REAL;

BEGIN
  FOR i:=0 TO 59 DO
    x:= FLOAT(i)*0.1047197; Sin[i]:= sin(x); Cos[i]:= cos (x);
  END;
  Clock.Get (Time); j:= 0;
  FOR i:= 0 TO 11 DO
    WITH FacePat[i] DO
      x:=  TRUNC(Sin[i*5]*FLOAT(HalfSize-4)) + HalfSize;
      y:= -TRUNC(Cos[i*5]*FLOAT(HalfSize-4)) + HalfSize;
    END;
  END;
  Day:= 0;

  VDIHandle:= OpenVirtWorkstation ();

  Style:= VDIAttributes.SetFillInteriorStyle
             (VDIHandle, VDIAttributes.Filled);
  Color:= VDIAttributes.SetFillColour (VDIHandle, 0);
  Color:= VDIAttributes.SetLineColour (VDIHandle, 1);

  SSWiS.Register         (Client, 'Clock', Accept);
  SSWiS.CreateWindow     (Client, 0, Restore);
  SSWiS.SetWindowTitle   (Client, 0, 'Clock');

  wh.X:= Size; wh.Y:= Size;
  xy.X:= SSWiS.ScreenSize.X - Size-8; xy.Y:= 8;
  SSWiS.SizeWindowContent (Client, 0, wh,wh,wh);
  SSWiS.PositionWindow    (Client, 0, xy);
  SSWiS.SetWindowElements (Client, 0, SSWiS.SetOfWindowElements
                                          {SSWiS.Iconiser});

  SSWiS.PlaceWindowOnTop (Client, 0);

END Init;


PROCEDURE Term;

BEGIN
  VDIControls.CloseVirtualWorkstation (VDIHandle);

  SSWiS.IconiseWindow(Client, 0);
  SSWiS.DeleteWindow (Client, 0);
  SSWiS.Deregister   (Client);
END Term;
```

```
BEGIN
  Init; IF System.OnModuleTerminationDo (Term) THEN END;
END Watch.
```

Der Pseudomodul SYSTEM Anhang E

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) SYSTEM;
(* .      Function      : Pseudo modul to import system details        *)
(* .      Version/Date  : 1.00  1.1.88                                 *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract -----------------------------------**

  The pseudo module system covers some system dependent functions and
  datatypes known by the compiler. There is no object form of module
  SYSTEM, but its declaration is given below as a normal definition
  module. Do not confuse the pseudo module SYSTEM with the runtime
  system module System! *)


(*----- Category : Types and Data ------------------------------------*)

TYPE    ADDRESS  = POINTER TO BYTE; (* compatible with all pointer types*)
                                    (* and with type LONGCARD          *)
        BYTE     ;                  (* smallest addressable unit of ... *)
                                    (* main memory. SIZE (BYTE) = 1     *)
        WORD     ;                  (* to consecutive BYTEs beginning ..*)
                                    (* at an even address.             *)


(*----- Category : Functions -----------------------------------------*)

PROCEDURE ADR           (VAR aVariable  : AnyType)
                        : ADDRESS;

(* Answer the address of aVariable, which may be of any type. *)


PROCEDURE INLINE        (    WordList   : WORD);

(* WordList may be a list of WORDs, separated by commas. The given WORDs
   are inserted into the instruction stream as words of code.          *)


PROCEDURE REG           (    Register   : INTEGER)
                        : LONGINT;

(* Answer the content of the indexed processor Register as a LONGINT
   value. Index 0 corresponds to D0, 1 to D1, 15 to A7. *)


PROCEDURE SETREG        (    Register   : INTEGER;
                             Value      : LONGINT);

(* Set the indexed Register to Value. *)


PROCEDURE SIZE          (                AnyType)
                        : INTEGER;

(* Answer the size in Bytes of the argument, which may be any type or
   a variable of any type. *)


PROCEDURE LONG          (    Value      : ShortType)
                        : LongType;
```

```
(* Answer the long form of Value, which may be of type CARDINAL,
   INTEGER or REAL. *)

PROCEDURE SHORT         (    Value      : LongType)
                        : ShortType;

(* Answer the short form of Value, which may be of type LONGCARD,
   LONGINT or LONGREAL. *)

PROCEDURE SHIFT         (    Value      : AnyType;
                             Count      : INTEGER)
                        : AnyType;

(* Answer the shifted Value. Value is shifted by Count bits, if count
   is positive, then Value is shifted to the left. *)

PROCEDURE VAL           (                 AnyType;
                             Argument   : AnyOtherType)
                        : AnyType;

(* Transfer Argument, which is of type AnyOtherType to AnyType. No
   extra code is generated for type transfers. *)

END SYSTEM.
```

Die STDLIB

Anhang F

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) ASCII;
(* .      Function      : Declare Non-Printing ASCII Characters         *)
(* .      Version/Date  : 1.00  27.1.88                                 *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract -------------------------------------**

  Declaration of non-printing ASCII characters as symbolic constants.*)


(*----- Category : Types and Data ---------------------------------------*)

CONST   NUL   = 00C;    SOH   = 01C;      STX   = 02C;
        ETX   = 03C;    EOT   = 04C;      ENQ   = 05C;
        ACK   = 06C;    BEL   = 07C;      BS    = 10C;
        HT    = 11C;    LF    = 12C;      VT    = 13C;
        FF    = 14C;    CR    = 15C;      SO    = 16C;
        SI    = 17C;    DLE   = 20C;      DC1   = 21C;
        DC2   = 22C;    DC3   = 23C;      DC4   = 24C;
        NAK   = 25C;    SYN   = 26C;      ETB   = 27C;
        CAN   = 30C;    EM    = 31C;      SUB   = 32C;
        ESC   = 33C;    FS    = 34C;      GS    = 35C;
        RS    = 36C;    US    = 37C;      DEL   = 177C;

CONST   EOL   = 36C; (* MODULA-2 standard End-Of-Line marker *)


END ASCII.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name           : *) ByteStreams;
(* .      Function       : Standard Input/Output Services              *)
(* .      Version/Date   : 1.1  22.1.88                                *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe     *)


(*----- Category : Module Abstract -----------------------------------**

  A ByteStream is a stream of bytes or words, with no interpretation
  put onto the data. Streams can be of several types: terminal, printer
  and file (communication in later versions). A stream is unidirectional.
  Once open, it can be either only read or written. Several error +
  conditions must be observed, when using streams. These can be io
  errors or end-of-stream conditions. *)


(*----- Category : Types and Data ------------------------------------*)

FROM    SYSTEM  IMPORT  BYTE;

TYPE    Results         = (Done         , (* no problems occurred      *)
                           NotDone      , (* formatting problems       *)
                           IOError      , (* device, os problems       *)
                           EndOfStream  , (* no more characters        *)
                           NotSupported); (* function not supported    *)

        Types           = (TerminalIn   , (* interactive device        *)
                           TerminalOut  , (* interactive device        *)
                           PrinterOut   , (* buffered device           *)
                           FileIn       , (* buffered device           *)
                           FileOut      );(* buffered device           *)

        Descriptor      ;

TYPE    Streams         = RECORD
          Result        :   Results;
          Descr         :   Descriptor;
                        END;


(*----- Category : Control -------------------------------------------*)

PROCEDURE Open          (VAR Stream     : Streams;
                             Name       : ARRAY OF CHAR;
                             Type       : Types);

(* Open the named stream. *)


PROCEDURE Close         (VAR Stream     : Streams);

(* Close Stream. *)


(*----- Category : Input ---------------------------------------------*)

PROCEDURE Read          (VAR Stream     : Streams;
                         VAR Block      : ARRAY OF BYTE;
                             Bytes      : INTEGER;
                         VAR BytesRead  : INTEGER);
```

```
(* Read Bytes (which must be <= SIZE(Block)) from Stream into Block and
   answer the number of BytesRead. *)

PROCEDURE ReadByte      (VAR Stream     : Streams;
                         VAR Byte       : CHAR);

(* Read a Byte from Stream. *)

PROCEDURE ReadWord      (VAR Stream     : Streams;
                         VAR Word       : CARDINAL);

(* Read a Word from Stream. *)

(*----- Category : Output -----------------------------------------------*)

PROCEDURE Write         (VAR Stream     : Streams;
                         VAR Block      : ARRAY OF BYTE;
                             Bytes      : INTEGER;
                         VAR BytesWritt : INTEGER);

(* Write Bytes (which must be <= SIZE(Block)) from Block to the Stream. *)

PROCEDURE WriteByte     (VAR Stream     : Streams;
                             Byte       : CHAR);

(* Write a Byte to Stream. *)

PROCEDURE WriteWord     (VAR Stream     : Streams;
                             Word       : CARDINAL);

(* Write a Word to Stream. *)

END ByteStreams.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Clock;
(* .      Function      : Standard Clock Module                          *)
(* .      Version/Date  : 1.0  19.9.87                                   *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe             *)


(*----- Category : Module Abstract -------------------------------------**

Clock provides a Modula-2 standard for time of day and time interval
measuring. The clocks resolution is implementation dependent and is
given below as the number of milliseconds, that cannot be resolved.
Absolute time is measured relative to 1st January 1980 00:00. The
time delivered is the system time (i.e. MEZ or MEZ summer depending on
the operators input at system startup). Future versions will provide
MEZ and the socalled dialog-time, which is either MEZ or MEZ summer.
*)

(*----- Category : Types and Data --------------------------------------*)

CONST   Resolution      = 5; (* [milliseconds] *)

TYPE    Time            = RECORD
          Day           :   LONGINT;
          Millisec      :   LONGINT;
                        END;

        Weekdays        = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);

        DecodedTime     = RECORD
          Year          :   CARDINAL;
          Month         :   [1..12];
          Day           :   [1..31];
          Hour          :   [0..23];
          Minute        :   [0..59];
          Millisec      :   CARDINAL;
          Weekday       :   Weekdays;
          DayInYear     :   [1..366];
                        END;

(*----- Category : Accessing System Timer ------------------------------*)

PROCEDURE Get           (VAR Arg        : Time);

(* Get the time of day in its encoded representation. *)


PROCEDURE Set           (    Arg        : Time);

(* Set the machines time of day clock. *)


(*----- Category : Conversions -----------------------------------------*)

PROCEDURE Decode        (    Enc        : Time;
                         VAR Dec        : DecodedTime);

(* Convert the encoded time value Enc to its decoded pendent Dec. *)


PROCEDURE Encode        (    Dec        : DecodedTime;
                         VAR Enc        : Time);
```

```
(* Convert the decoded time value Dec to its encoded pendent Enc. *)

(*----- Category : Calculations ----------------------------------------*)

PROCEDURE Sub            (VAR Minuend    : Time;
                              Subtrahend : Time);

(* Calulate the expression Minuend:= Minued-Subtrahend. *)

END Clock.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Coroutines;
(* .      Function      : *)
(* .      Version/Date  : 1.00 / 13.11.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)

(*----- Category : Change/Version Remarks ------------------------------**

  1.00  MC68000 ATARI coroutine handler, implemented according to
        Wirth, Niklaus: Programmieren in Modula-2, 1985
*)

(*----- Category : Types and Data ---------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;
PROCEDURE NEWPROCESS    (    P          : PROC;
                             A          : ADDRESS;
                             n          : CARDINAL;
                         VAR new        : ADDRESS);

(* create a new coroutine variable new
   consisting of:
    - the parameterless procedure P
    - the coroutine stack with address A and length n
   The system needs a minimum stacklength of 128 bytes,
   but it is usefull to give sufficient stack.
   ATTENTION: stackoverflow is not detected.  *)


PROCEDURE TRANSFER      (VAR source,
                             destination: ADDRESS);

(* Switch to coroutine destination.
   The context of the actual coroutine is saved to source. *)


PROCEDURE IOTRANSFER    (VAR source,
                             destination: ADDRESS;
                             vector     : CARDINAL);

(* Switch to coroutine <destination>.
   The context of the actual coroutine is saved to <source>
   and the actual coroutine is intialized as interruptservice-
   routine for interruptvector <vector>.

   There is no possibility in the concept of N. Wirth to stop
   the interrupt by an explicit procedure. Therefore its neccessary
   to restore the old interruptservice-routine after an interrupt.
   User's interruptservice-routine had to install the interrupt
   again by a new call to IOTRANSFER! *)


END Coroutines.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) FileSystem;
(* .      Function      : Standard File Services                        *)
(* .      Version/Date  : 1.0  27.8.87                                  *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

FileSystem provides basic sequential and random file access as defined
in Wirth's "Programming in MODULA-2". Files are streams of bytes or
words. The application programmer is advised to use this module for
file io to make its programs portable between different MODULA-2
implementations. *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT  WORD, BYTE, ADDRESS;
IMPORT  Clock;

TYPE    Response        = (done          , (* successful completion      *)
                           notdone       , (* error, not specified else  *)
                           notsupported  , (* internal use               *)
                           callerror     , (* improper filestate         *)
                           unknownmedium, (* drive does not exist        *)
                           unknownfile   , (* file not found             *)
                           paramerror    , (* invalid parameter          *)
                           toomanyfiles  , (* more files than system sup *)
                           eom           , (* end of medium reached      *)
                           userdeverror); (* internal use                *)

        (* File is an implementation dependent file descriptor. The      *)
        (* only fields, that are generally visible to the application    *)
        (* level, are eof, which indicates, that the end of the file     *)
        (* has been reached, and res, which is used to indicate the      *)
        (* completion status of each operation.                          *)

TYPE    Descriptor      ;

TYPE    File            = RECORD
          res           :   Response;
          eof           :   BOOLEAN;
          (* following items are not available *)
          (* to the application program level. *)
          Descr         :   Descriptor;
                        END;


(*----- Category : Opening, Closing, Renaming, Deleting ----------------*)

PROCEDURE Lookup        (VAR F          : File;
                             Filename   : ARRAY OF CHAR;
                             New        : BOOLEAN);

(* Looks for a file with the given name. If the file exists, it is
connected to F (opened). If the requested file is not found or new is
TRUE, a permanent file is created with the given name. After the call
F.res = done, if the file f is connected,
F.res = notdone, if the file does not exist or some error occurred. *)
```

```
PROCEDURE Close         (VAR F          : File);

(* Terminate any actual input or output operation on file F and
disconnect F from the actual file. *)


PROCEDURE Delete        (VAR F          : File);

(* Terminate any actual input or output operation on file F and
disconnect F from the actual file. The file is deleted. *)


PROCEDURE Rename        (VAR F          : File;
                         Filename       : ARRAY OF CHAR);

(* Change the name of file F to Filename. If F.res returnes not done,
then a file with the given name already exists, or some other error
occured. *)

(*----- Category : Position and Size ---------------------------------*)

PROCEDURE SetPos        (VAR F          : File;
                         Pos            : LONGCARD);

(* Sets the current read/write position of file F to Pos. If Pos is
greater than the actual file length, then the file is positioned to
its end. *)


PROCEDURE GetPos        (VAR F          : File;
                         VAR Pos        : LONGCARD);

(* Get the actual read/write position of file F. *)


PROCEDURE Length        (VAR F          : File;
                         VAR Len        : LONGCARD);

(* Get the number of bytes in file F. *)


(*----- Category : Reading -------------------------------------------*)

PROCEDURE ReadChar      (VAR F          : File;
                         VAR Ch         : CHAR);

(* Read the next byte form file F and assign its value to Ch. If the
operation was not successfull, then Ch will return 0C and F.res
indicates the problem. F.eof implies Ch = 0C. The opposite, however,
is not true: Ch = 0C does not imply F.eof. After the call
F.eof = FALSE, Ch has been read,
F.eof = TRUE , operation was not successfull.
IF F.eof = TRUE, then
F.res = done, end of file has been reached,
F.res # done, some error occured. *)


PROCEDURE ReadWord      (VAR F          : File;
                         VAR W          : WORD);

(* Same as ReadChar, except that a word quantity is read from the
file. *)


(*----- Category : Writing -------------------------------------------*)
```

```
PROCEDURE WriteChar     (VAR F          : File;
                             Ch         : CHAR);

(* Write the byte Ch to file F at its current read/write position. *)

PROCEDURE WriteWord     (VAR F          : File;
                             W          : WORD);

(* Same as WRiteChar for word quantities. *)

END FileSystem.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) HFS;
(* .      Function      : Hierarchical File System                      *)
(* .      Version/Date  : 1.0  6.11.87                                  *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -----------------------------------**

The module supports the naming conventions of a hierarchical file
system, while freeing the application modules from the syntax of
the filesystem at hand. By carefully using the exports of HFS,
programs may become filesystem independent. Actual access to files
is done via the services of FileSystem. Important terms :

  1. Selection      - is a full filename given as a string of
                      arbitrary length in an implementation
                      dependent syntax.
  2. Volume         - is the name of the medium containing a file.
  3. Folder         - is a catalog of files. Folders may be nested.
  4. Document       - (synonym File) is a collection of data.
  5. Type           - is a sequence of characters, appended to the
                      document's name to indicate its type.
  6. Current Folder - is the folder, which's documents are accessed
                      when no extra folder name is specified.
*)


(*----- Category : Type and Data -------------------------------------*)

VAR     FolderSep       ,
        VolumeSep       ,
        TypeSep         : ARRAY [0..0] OF CHAR;
        NameLength      : INTEGER;
        TypeLength      : INTEGER;

TYPE    FileProc        = PROCEDURE ( (*Filename : *) ARRAY OF CHAR);

        (* Do something with the file named Filename. *)


(*----- Category : Primitives ----------------------------------------*)

PROCEDURE ForAllFilesDo (    Selection  : ARRAY OF CHAR;
                             What       : FileProc;
                             rOption    : BOOLEAN);

(* To all files, matching Selection, apply the procedure What. If the
rOption is on, then traverse all subdirectories of the directory
containing Selection. *)


PROCEDURE CurrentFolder (VAR Selection  : ARRAY OF CHAR);

(* Answer the name of the current folder in Selection. *)


PROCEDURE AskName       (VAR Selection  : ARRAY OF CHAR;
                         VAR Done       : BOOLEAN);

(* Ask the user for a filename. Answer the filename in Selection and
set Done TRUE if successfull. The default selction is passed in
```

```
Selection to AskName. *)

PROCEDURE Decode        (    Selection  : ARRAY OF CHAR;
                         VAR Volume     : ARRAY OF CHAR;
                         VAR Folder     : ARRAY OF CHAR;
                         VAR Document   : ARRAY OF CHAR;
                         VAR Type       : ARRAY OF CHAR);

(* Decode a full filename given in Selection into its components as
explained above. *)

PROCEDURE Encode        (    Volume     : ARRAY OF CHAR;
                             Folder     : ARRAY OF CHAR;
                             Document   : ARRAY OF CHAR;
                             Type       : ARRAY OF CHAR;
                         VAR Selection  : ARRAY OF CHAR);

(* Construct a filename from the components given. *)

END HFS.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) InOut;
(* .      Function      : Standard Input/Output Services                *)
(* .      Version  Date : 23: 5  15.10.1988                             *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

Provides the standard input/output service. Input/output is directed to
the interactive console, unless it is redirected by calling OpenInput
or OpenOutput to a disk file. *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT ADDRESS;

VAR     Done    : BOOLEAN;  (* Signal the success of certain functions. *)
        TermCh  : CHAR;


(*----- Category : Control -----------------------------------------------*)

PROCEDURE OpenInput     (    Extension  : ARRAY OF CHAR);

(* Request a file name with the given extension. Done is TRUE if the
file was successfully opened. If open, subsequent input is read from
this file. *)


PROCEDURE OpenOutput    (    Extension  : ARRAY OF CHAR);

(* Request a file name with the given extension. Done is TRUE if the
file was successfully opened. If open, subsequent output is written to
this file. *)


PROCEDURE RedirectInput (    Name       : ARRAY OF CHAR);

(* Redirect input to the named file.  Done is TRUE if the file was
successfully opened. If open, subsequent input is read from this file. *)


PROCEDURE RedirectOutput(    Name       : ARRAY OF CHAR);

(* Redirect output to the named file.  Done is TRUE if the file was
successfully opened. If open, subsequent output is Written to this file. *)


PROCEDURE CloseInput;

(* Closes input file, returns input to terminal. *)


PROCEDURE CloseOutput;

(* Closes output file, returns output to terminal. *)


(*----- Category : Input -------------------------------------------------*)
```

```
PROCEDURE Read          (VAR Ch         : CHAR);

(* Read a character from standard input. Done is TRUE, if input has not
reached eof. *)


PROCEDURE ReadCard      (VAR Number     : CARDINAL);

(* Read a string, convert it to CARDINAL and assign it to Number.
Syntax: cardinal = digit {digit}.
Leading blanks are ignored. Done is TRUE if Number was read. *)


PROCEDURE ReadInt       (VAR Number     : INTEGER);

(* Read string and convert to INTEGER.
Syntax: integer = ["+"|"-"] digit {digit}.
Leading blanks are ignored. Done is TRUE if Number was read. *)


PROCEDURE ReadReal      (VAR Number     : REAL);

(* Read a string, convert it to REAL and assign it to Number. Syntax:

   realnumber       = fixedpointnumber [exponent].
   fixedpointnumber = [sign] {digit} [ '.' {digit} ].
   exponent         =  ('e' | 'E') [sign] digit {digit}.
   sign             = '+' | '-'.
   digit            = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'.

Leading blanks are ignored. Done is TRUE if Number was read. *)


PROCEDURE ReadLongcard  (VAR Number     : LONGCARD);

PROCEDURE ReadLongint   (VAR Number     : LONGINT);

PROCEDURE ReadLongreal  (VAR Number     : LONGREAL);


PROCEDURE ReadString    (VAR String     : ARRAY OF CHAR);

(* Read string, i.e. sequence of characters not containing blanks nor
control characters. Leading blanks are ignored. Input is terminated by
any character <= " ". This character is assigned to TermCh. Backspace
is used for backspacing when input from terminal. *)


PROCEDURE ReadLn        (VAR String     : ARRAY OF CHAR);

(* Read a line, i.e. a sequnce of characters not containing control
characters. Input is terminated by any characters < ' '. This character
is assigned to TermCh. Backspace is used for backspacing when input
from terminal. *)


(*----- Category : Output -----------------------------------------------*)

PROCEDURE Write         (    Ch         : CHAR);

(* Write character Ch to standard output. BS is interpreted. *)


PROCEDURE WriteString   (    String     : ARRAY OF CHAR);
```

```
(* Write the String to standard output. *)

PROCEDURE WriteCard     (    Number     : CARDINAL;
                             Length     : CARDINAL);

PROCEDURE WriteHex      (    Number     : CARDINAL;
                             Length     : CARDINAL);

PROCEDURE WriteOct      (    Number     : CARDINAL;
                             Length     : CARDINAL);

PROCEDURE WriteInt      (    Number     : INTEGER;
                             Length     : CARDINAL);

PROCEDURE WriteReal     (    Number     : REAL;
                             Length     : CARDINAL;
                             FracLength : INTEGER);

PROCEDURE WriteLongcard (    Number     : LONGCARD;
                             Length     : CARDINAL);

PROCEDURE WriteLongint  (    Number     : LONGINT;
                             Length     : CARDINAL);

PROCEDURE WriteLongreal (    Number     : LONGREAL;
                             Length     : CARDINAL;
                             FracLength : INTEGER);

PROCEDURE WriteAddress  (    Number     : ADDRESS;
                             Length     : CARDINAL);

(* Write integer/cardinal/real Number with (at least) n characters to
standard output. If n is greater than the number of digits needed,
blanks are added preceding the number. *)


PROCEDURE WriteLn;

(* terminate line *)


END InOut.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) LMathLib;
(* .      Function      : Standard Math Functions                        *)
(* .      Version/Date  : 1.0  24.9.87                                   *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe             *)


(*----- Category : Module Abstract ---------------------------------------**

*)

(*----- Category : Types and Data ----------------------------------------*)

CONST   e       = 2.71828183D;
        pi      = 3.14159265D;

VAR     Epsilon : LONGREAL;


(*----- Category : Double Precision Arithmetics -------------------------*)

PROCEDURE exp           ( x : LONGREAL) : LONGREAL;

PROCEDURE ln            ( x : LONGREAL) : LONGREAL;

PROCEDURE lg            ( x : LONGREAL) : LONGREAL;

PROCEDURE sqrt          ( x : LONGREAL) : LONGREAL;


PROCEDURE sin           ( x : LONGREAL) : LONGREAL;

PROCEDURE cos           ( x : LONGREAL) : LONGREAL;

PROCEDURE tan           ( x : LONGREAL) : LONGREAL;

PROCEDURE cot           ( x : LONGREAL) : LONGREAL;

PROCEDURE arcsin        ( x : LONGREAL) : LONGREAL;

PROCEDURE arccos        ( x : LONGREAL) : LONGREAL;

PROCEDURE arctan        ( x : LONGREAL) : LONGREAL;


PROCEDURE sinh          ( x : LONGREAL) : LONGREAL;

PROCEDURE cosh          ( x : LONGREAL) : LONGREAL;

PROCEDURE tanh          ( x : LONGREAL) : LONGREAL;


(*----- Category : Conversions -------------------------------------------*)

PROCEDURE real          ( x : LONGINT) : LONGREAL;

PROCEDURE entier        ( x : LONGREAL) : LONGINT;


(*----- Category : Initialisation ----------------------------------------*)

PROCEDURE Init;
```

```
END LMathLib.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) MathLib;
(* .      Function      : Standard Math Functions                        *)
(* .      Version/Date  : 1.0  24.9.87                                   *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe             *)


(*----- Category : Module Abstract -------------------------------------**

*)

(*----- Category : Types and Data ---------------------------------------*)

CONST   e               = 2.71828183;
        pi              = 3.14159265;

VAR     Epsilon         : REAL;


(*----- Category : Double Precision Arithmetics ------------------------*)

PROCEDURE exp           ( x : REAL) : REAL;

PROCEDURE ln            ( x : REAL) : REAL;

PROCEDURE lg            ( x : REAL) : REAL;

PROCEDURE sqrt          ( x : REAL) : REAL;


PROCEDURE sin           ( x : REAL) : REAL;

PROCEDURE cos           ( x : REAL) : REAL;

PROCEDURE tan           ( x : REAL) : REAL;

PROCEDURE cot           ( x : REAL) : REAL;

PROCEDURE arcsin        ( x : REAL) : REAL;

PROCEDURE arccos        ( x : REAL) : REAL;

PROCEDURE arctan        ( x : REAL) : REAL;


PROCEDURE sinh          ( x : REAL) : REAL;

PROCEDURE cosh          ( x : REAL) : REAL;

PROCEDURE tanh          ( x : REAL) : REAL;


(*----- Category : Conversions -----------------------------------------*)

PROCEDURE real          ( x : INTEGER) : REAL;

PROCEDURE entier        ( x : REAL) : INTEGER;


END MathLib.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) NumberConversions;
(* .      Function      : Standard Number Conversions                  *)
(* .      Version/Date  : 1.00  15.1.88                                *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract -----------------------------------**

Contains routines to convert numbers to strings and vice versa.
*)


(*----- Category : Numbers to Strings --------------------------------*)

PROCEDURE CardToString  (    Number     : CARDINAL;
                         VAR String     : ARRAY OF CHAR;
                             Width      : CARDINAL);

(* Convert Number into a String of length Width. *)


PROCEDURE IntToString   (    Number     : INTEGER;
                         VAR String     : ARRAY OF CHAR;
                             Width      : CARDINAL);

(* See above. *)


PROCEDURE LongCardToString
                        (    Number     : LONGCARD;
                         VAR String     : ARRAY OF CHAR;
                             Width      : CARDINAL);

(* See above. *)


PROCEDURE LongIntToString
                        (    Number     : LONGINT;
                         VAR String     : ARRAY OF CHAR;
                             Width      : CARDINAL);

(* See above. *)


(*----- Category : Strings To Numbers --------------------------------*)

PROCEDURE StringToCard  (    String     : ARRAY OF CHAR;
                         VAR Number     : CARDINAL;
                         VAR Ok         : BOOLEAN);

(* Convert String into a Number, andanswer, if Ok. *)


PROCEDURE StringToInt   (    String     : ARRAY OF CHAR;
                         VAR Number     : INTEGER;
                         VAR Ok         : BOOLEAN);

(* See above. *)


PROCEDURE StringToLongCard
```

```
                        (    String      : ARRAY OF CHAR;
                         VAR Number      : LONGCARD;
                         VAR Ok          : BOOLEAN);

(* See above. *)


PROCEDURE StringToLongInt
                        (    String      : ARRAY OF CHAR;
                         VAR Number      : LONGINT;
                         VAR Ok          : BOOLEAN);

(* See above. *)


END NumberConversions.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type         : *) DEFINITION MODULE
(* .      Name         : *) Printer;
(* .      Function     : Printer Driver Module                            *)
(* .      Version Date : 14:48  12.11.1988                                *)
(* Product Name        : SPC                                              *)
(* Copyright           : (c) 1987,1988, MODsoft, D7500 Karlsruhe          *)


(*----- Category : Module Abstract ---------------------------------------**

  Printer provides the interface to the system's printer. The printer
  may be configured by supplying a configuration file, which gives the
  control codes for a particular printer. The configuration file has the
  same format as the WordPlus printer configuration (PRINTER.CFG).
  Printer configuration files can be found in the public domain for
  nearly all printers.

  Configuration parameters, including the name of the configuration file
  are stored within the environment variable PRINTFLAGS. *)


(*----- Category : Types and Data ----------------------------------------*)

TYPE    HeadProc        = PROCEDURE ((* pageno : *) CARDINAL);

        Fonts           = (Pica, Elite, Small, Large);
        FontSet         = SET OF Fonts;

        Attributes      = (Highlight, Underline, Superscript, Subscript);
        AttributeSet    = SET OF Attributes;


(*----- Category : Printer state -----------------------------------------*)

PROCEDURE Online        () :BOOLEAN;

(* Answer TRUE if printer is ready. *)


(*----- Category : Print primitives --------------------------------------*)

PROCEDURE Write         (     c         : CHAR );

(* Print a single character. If a special printcode is found send it
   to the printer. If on the last position in line then WriteLn. *)


PROCEDURE WriteString   (     String    : ARRAY OF CHAR);

(* Print a specified string using printer code. *)


PROCEDURE WriteLn;

(* Advance paper to the beginning of the next line, if the actual line
   is the last line on the form then Page. *)


PROCEDURE Page;

(* Advance paper to the top of a new form. *)
```

```
(*----- Category : Print layout params --------------------------------*)

PROCEDURE Reset;

(* Set default values for column, line number and page number. *)

PROCEDURE CurrentPosition() : CARDINAL;

(* Answer current print position (1..LastPos) in line. *)

PROCEDURE CurrentLine   () : CARDINAL;

(* Answer current line (1..LastLine) on page. *)

PROCEDURE CurrentPage   () : CARDINAL;

(* Answer numbers of page since last call of Reset. *)

PROCEDURE CharsPerLine  () : CARDINAL;

(* Answer maximum number of chars per line. *)

PROCEDURE LinesPerPage  () : CARDINAL;

(* Answer maximum number of lines per page. *)

PROCEDURE LeftBorderSize() : CARDINAL;

(* Answer indent on the left side. *)

PROCEDURE SetCharsPerLine(   newvalue   : CARDINAL);

(* Set maximum number of chars per line. *)

PROCEDURE SetLinesPerPage(   newvalue   : CARDINAL);

(* Set maximum number of lines per page. *)

PROCEDURE SetLeftBorderSize
                         (   newvalue   : CARDINAL);

(* Set left indentation. *)

PROCEDURE InstallHeader (    Header     : HeadProc);

(* Tell printer, how to print the page header. *)

PROCEDURE Head          (    OnNotOff   : BOOLEAN );

(* Activate or deactive the installed header procedure *)

(*----- Category : Print attribute params -----------------------------*)
```

```
PROCEDURE SetAttribute  (    Attr       : Attributes;
                             OnNotOff   : BOOLEAN);

(* Set an printing attribute on or off, if possible.*)

PROCEDURE SetFont       (    Font       : Fonts);

(* Set a print font for subsequent printing, if possible.*)

PROCEDURE SupportedFonts() : FontSet;

(* Returns the fonts supported by the current driver.*)

PROCEDURE SupportedAttributes
                        () : AttributeSet;

(* Returns the attributes supported by the current driver.*)

(*----- Category : Initialisation ---------------------------------------*)

PROCEDURE Init;

PROCEDURE GetName       (VAR PrinterName: ARRAY OF CHAR);

(* Answer the name of the configured printer. *)

PROCEDURE Load          (VAR CFGFileName: ARRAY OF CHAR);

(* Load a new configuration file. *)

PROCEDURE Term;

END Printer.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) RealConversions;
(* .      Function      : Standard Realnumber Conversions              *)
(* .      Version  Date :  0:39   2. 6.1988                            *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract ---------------------------------------**

Contains routines to convert real numbers to strings and vice versa.
*)


(*----- Category : Numbers to Strings -----------------------------------*)

PROCEDURE RealToString  (    Number     : REAL;
                         VAR String     : ARRAY OF CHAR;
                             Width      : CARDINAL;
                             FracWidth  : INTEGER);

(* Convert real Number into a String of length Width with FracWidth
   digits to the right of the decimal point. *)


PROCEDURE LongRealToString
                        (    Number     : LONGREAL;
                         VAR String     : ARRAY OF CHAR;
                             Width      : CARDINAL;
                             FracWidth  : INTEGER);

(* See above. *)


(*----- Category : Strings to Numbers -----------------------------------*)

PROCEDURE StringToReal  (    String     : ARRAY OF CHAR;
                         VAR Number     : REAL;
                         VAR Ok         : BOOLEAN);

(* Convert a String into a real Number and answer Ok if successfull. *)


PROCEDURE StringToLongReal
                        (    String     : ARRAY OF CHAR;
                         VAR Number     : LONGREAL;
                         VAR Ok         : BOOLEAN);

(* See above. *)

END RealConversions.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Storage;
(* .      Function      : Standard Memory Management                    *)
(* .      Version  Date : 10:51  16.10.1988                             *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)

(*----- Category : Module Abstract -----------------------------------**

The module provides dynamic memory allocation and deallocation. However,
modules must carefully check, if the memory, they requested has been
successfully allocated. If not, the failed program MUST deallocate all
previously allocated memory and free all resources, before terminating.
If programs terminate, leaving memory allocated, the system has no
chance to garbage-collect this memory.
*)

(*----- Category : Types and Data ------------------------------------*)

IMPORT  SYSTEM;

TYPE    Quantity        = LONGCARD;

(*----- Category : Primitives ----------------------------------------*)

PROCEDURE ALLOCATE      (VAR MemoryAddr : SYSTEM.ADDRESS;
                         Size       : Quantity);
PROCEDURE Allocate      (VAR MemoryAddr : SYSTEM.ADDRESS;
                         Size       : Quantity);

(* Allocate an area of Size bytes in length and return its address in
MemoryAddr. If no space is available, an error condition is raised. *)


PROCEDURE DEALLOCATE    (VAR MemoryAddr : SYSTEM.ADDRESS);
PROCEDURE Deallocate    (VAR MemoryAddr : SYSTEM.ADDRESS);

(* Deallocate the area of memory pointed to by MemoryAddr. This area
must have been allocated previously by Allocate. *)


PROCEDURE Available     (    Size       : Quantity)
                        : BOOLEAN;

(* Answer TRUE, if Size bytes could be allocated. *)


END Storage.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Strings;
(* .      Function      : String Primitives                              *)
(* .      Version/Date  : 1.01 10.1.88                                   *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe        *)


(*----- Category : Module Abstract -----------------------------------**

The Strings module realises the Modula-2 standard string processing
facilities. Strings are ARRAYs OF CHAR of arbitrary length. A String
is terminated either by a NUL character (0C) or by the length of the
character array (HIGH(String)+1). Positions within Strings are denoted
by indices ranging from 0 to stringlength-1. If an index greater than
the stringlength is passed to one of the procedures, then stringlength
 1 is assumed. If the result of any string primitive is longer than
the destination character array, then the result is truncated.
*)

(*----- Category : Types and Data ------------------------------------*)

CONST   End             = 0C; (* Termination of Modula-2 Strings *)
        Equal           = 0;  (* String Compare Results          *)
        Less            = -1;
        Greater         = 1;


(*----- Category : Inquiries -----------------------------------------*)

PROCEDURE Length        (VAR Source     : ARRAY OF CHAR)
                        : CARDINAL;

(* Answer the number of characters in Source. *)


(*----- Category : String Manipulation -------------------------------*)

PROCEDURE Clear         (VAR Destin     : ARRAY OF CHAR);

(* Make Destin a string of length 0. Extension to standard lib. *)


PROCEDURE Assign        (VAR Destin     : ARRAY OF CHAR;
                             Source     : ARRAY OF CHAR);

(* Assign the Source string to the Destin array. *)


PROCEDURE Concat        (    Source1    ,
                             Source2    : ARRAY OF CHAR;
                         VAR Destin     : ARRAY OF CHAR);

(* Concatenate the Source1 and Source2 strings and return the result
in the Destin character array. *)


PROCEDURE Insert        (    Source     : ARRAY OF CHAR;
                         VAR Destin     : ARRAY OF CHAR;
                             At         : CARDINAL);

(* Insert the Source string immediatly before the character denoted by
At into the Destin string. *)
```

```
PROCEDURE Delete        (VAR Destin     : ARRAY OF CHAR;
                             At, For    : CARDINAL);

(* Delete For characters starting with the character at position At
from the Destin string. *)


PROCEDURE Copy          (    Source     : ARRAY OF CHAR;
                             From, For  : CARDINAL;
                         VAR Destin     : ARRAY OF CHAR);

(* Copy For characters starting with the character at position From
from the Source string into the Destin character array, starting at 0. *)


PROCEDURE Pad           (VAR Destin     : ARRAY OF CHAR;
                             UpTo       : CARDINAL;
                             With       : ARRAY OF CHAR);

(* Pad the Destin character array up to the character position UpTo
by repeating the string With. Extension to standard lib. *)


(*----- Category : Comparisons ----------------------------------------*)

PROCEDURE Pos           (    Substr     : ARRAY OF CHAR;
                             String     : ARRAY OF CHAR)
                        : CARDINAL;

(* Search the string Substr within the String and return the starting
character index if found, otherwise MAX(CARDINAL). *)


PROCEDURE Compare       (    String1    : ARRAY OF CHAR;
                             String2    : ARRAY OF CHAR)
                        : INTEGER;

(* Compare String1 with String2 and return 0 if they are equal, 1 if
String1 greater String2 or -1 if String1 less than String2. Two strings
are equal, if the have the same length and contain the same characters.
A string A is greater than string B, if they contain the same characters
up to a certain index and then either string B has no more characters
or the next character of string A is greater than that of string B. *)

END Strings.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Terminal;
(* .      Function      : Window Based Standard Terminal                *)
(* .      Version/Date  : 1.0  19.9.87                                  *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -----------------------------------**

Standard terminal module used for basic interactive io. *)


(*----- Category : Input and Output ----------------------------------*)

PROCEDURE Read          (VAR ch         : CHAR);

(* Read a character from interactive input. No interpretation of control
   characters is performed (transparent input). *)


PROCEDURE BusyRead      (VAR ch         : CHAR);

(* Look for a character in the input buffer. If none is present, then
   answer 0C. *)


PROCEDURE Write         (    ch         : CHAR);

(* Write a character to interactive output. BS,CR and FF are
   interpreted. *)


PROCEDURE WriteString   (    Text       : ARRAY OF CHAR);

(* Write a string to interactive output. No control characters are
interpreted. *)


PROCEDURE WriteLn;

(* Terminate the current line and skip to the next line on interactive
output. *)


PROCEDURE WriteLong     (    Arg        : LONGINT;
                             Length     : CARDINAL);

(* Write a number to interactive output. Used for test purposes. *)


(*----- Category : Window Driven Terminals ---------------------------*)

PROCEDURE Expose;

(* If Terminal has a window driven implementation, Terminal's window is
   placed on top of other windows. Will be activated upon read. *)


PROCEDURE Hide;

(* If Terminal has a window driven implementation, Terminal's window is
   iconised. *)
```

```
END Terminal.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) TextStreams;
(* .      Function      : Standard Textual Input/Output                  *)
(* .      Version  Date :  9:37  11.12.1988                              *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract ---------------------------------------**

  A TextStream is a stream of characters, not containing any control
  characters with the exception of the end-of-line marker EOL. EOL is
  a single character and is mapped to the implementation or hardware
  dependent end-of-line action (and vice versa). Streams can be of
  several types: terminal, printer and file (communication in later
  versions). A stream is unidirectional. Once open, it can be either
  only read or written. Several error conditions must be observed,
  when using streams. These can be io errors, end-of-stream conditions
  and formatting problems. After every input action the TermCh holds
  the last character read from the stream, which is usually the item
  terminating character. *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT ADDRESS;

TYPE    Results         = (Done         , (* no problems occurred       *)
                           NotDone      , (* formatting problems        *)
                           IOError      , (* device, os problems        *)
                           EndOfStream  , (* no more characters         *)
                           NotSupported); (* function not supported     *)

        Types           = (TerminalIn   , (* interactive device         *)
                           TerminalOut  , (* interactive device         *)
                           PrinterOut   , (* buffered device            *)
                           FileIn       , (* buffered device            *)
                           FileOut      );(* buffered device            *)

        Descriptor      ;

TYPE    Streams         = RECORD
          Result        :   Results;      (* see above                  *)
          TermCh        :   CHAR;         (* terminator on last input   *)
          Descr         :   Descriptor;   (* not visible to caller      *)
                        END;

CONST   EOL             = 36C;            (* MODULA-2 end-of-line marker*)


(*----- Category : Control -----------------------------------------------*)

PROCEDURE Open          (VAR Stream     : Streams;
                             Name       : ARRAY OF CHAR;
                             Type       : Types);

(* Open Stream with Name and Type. If Type is FileIn or FileOut, Name
   is the filename, not used otherwise. *)


PROCEDURE Close         (VAR Stream     : Streams);

(* Close Stream. Stream cannot be used after call to Class. *)
```

```
(*----- Category : Input ------------------------------------------------*)

PROCEDURE Read          (VAR Stream     : Streams;
                         VAR Ch         : CHAR);

(* Read the next character, which is either EOL or a non-control
   character. Note, that TermCh contains the same character. *)


PROCEDURE ReadCard      (VAR Stream     : Streams;
                         VAR Number     : CARDINAL);

(* Read a cardinal number from Stream and assign it to number if
   Done. Leading blanks and control characters are ignored. If
   NotDone, then Stream is not advanced, i.e. the same characters,
   with the exception of the skipped blanks and control characters
   can be read again. The character following the number is assigned
   to TermCh. *)


PROCEDURE ReadInt       (VAR Stream     : Streams;
                         VAR Number     : INTEGER);

(* Same as above for integer number. *)


PROCEDURE ReadReal      (VAR Stream     : Streams;
                         VAR Number     : REAL);

(* Same as above for real number. *)


PROCEDURE ReadLongcard  (VAR Stream     : Streams;
                         VAR Number     : LONGCARD);

(* Same as above for long cardinal number. *)


PROCEDURE ReadLongint   (VAR Stream     : Streams;
                         VAR Number     : LONGINT);

(* Same as above for long int number. *)


PROCEDURE ReadLongreal  (VAR Stream     : Streams;
                         VAR Number     : LONGREAL);

(* Same as above for long real number. *)


PROCEDURE ReadString    (VAR Stream     : Streams;
                         VAR String     : ARRAY OF CHAR);

(* Same as above, but a string is returned not containing blanks
   nor control characters. Leading blanks and control characters
   are skipped. Terminating character is assigned to TermCh. *)


PROCEDURE ReadLn        (VAR Stream     : Streams;
                         VAR String     : ARRAY OF CHAR);

(* Skip any leading control characters with the exception of EOL.
   Then transfer all characters up to the next control character
```

```
   to String. Assign the terminating character to TermCh. Note,
   that String may be empty. *)

(*----- Category : Output ------------------------------------------------*)

PROCEDURE Write          (VAR Stream     : Streams;
                          Ch         : CHAR);

(* If Ch is a control character, then it is ignored. Otherwise it is
   written to Stream. If Stream is interactive (see declaraction of
   StreamTypes) then the buffer is flushed immediatly. *)


PROCEDURE WriteString    (VAR Stream     : Streams;
                          String     : ARRAY OF CHAR);

(* Write String to Stream, skipping all control characters. If
   Stream is interactive, then the buffer is flushed immediatly. *)


PROCEDURE WriteCard      (VAR Stream     : Streams;
                          Number     : CARDINAL;
                          Length     : CARDINAL);

(* Write Number in decimal representation to Stream, using at least
   Length digits (more if needed). Flush if interactive. *)


PROCEDURE WriteHex       (VAR Stream     : Streams;
                          Number     : CARDINAL;
                          Length     : CARDINAL);

(* Same as above in sedecimal representation. Suffix 'H' is added. *)


PROCEDURE WriteOct       (VAR Stream     : Streams;
                          Number     : CARDINAL;
                          Length     : CARDINAL);

(* Same as above in octal representation. Suffix 'O' is added. *)


PROCEDURE WriteInt       (VAR Stream     : Streams;
                          Number     : INTEGER;
                          Length     : CARDINAL);

(* Same as above for integer numbers in decimal representation. *)


PROCEDURE WriteReal      (VAR Stream     : Streams;
                          Number     : REAL;
                          Length     : CARDINAL;
                          FracLength : INTEGER);

(* Same as above for real numbers in decimal representation. *)


PROCEDURE WriteLongcard (VAR Stream     : Streams;
                          Number     : LONGCARD;
                          Length     : CARDINAL);

(* Same as above for long cardinal numbers in decimal representation. *)
```

```
PROCEDURE WriteLongint  (VAR Stream     : Streams;
                         Number     : LONGINT;
                         Length     : CARDINAL);

(* Same as above for long integer numbers in decimal representation. *)


PROCEDURE WriteLongreal (VAR Stream     : Streams;
                         Number     : LONGREAL;
                         Length     : CARDINAL;
                         FracLength : INTEGER);

(* Same as above for long real numbers in decimal representation. *)


PROCEDURE WriteAddress  (VAR Stream     : Streams;
                         Address    : ADDRESS;
                         Length     : CARDINAL);

(* Same as above for pointer values in sedecimal representation. *)


PROCEDURE WriteLn       (VAR Stream     : Streams);

(* Add an EOL to the Stream. *)


END TextStreams.
```

Die SPCLIB

Anhang G

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Bytes;
(* .      Function      : Fast Copy and Scan Routines                    *)
(* .      Version  Date :  7:26  10.11.1988                              *)
(* Authors              : R.Huetter                                      *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1988, MODsoft, D7500 Karlsruhe             *)


(*----- Category : Module Abstract ---------------------------------------**

  Bytes provides a set of basic routines to efficiently scan and copy
  sequences of bytes. The routines may be used to process 0C terminated
  strings, if the terminator  is contained in one of the termination
  conditions. Compared to the routines in module Strings, these routines
  are in general faster, since they are all realised using asembler
  loops. Another advantage of Bytes is, that all source and destination
  operands are given as VAR parameters, thus preventing the system from
  pushing strings onto the stack.

  A set of bytes may be used to specify the termination condition for
  scans and copies. The set type may be seen as a set of characters,
  which is normally not a feature of the SPC-MODULA-2 implementation.
  Procedures to manipulate large sets are part of this module as well.
  *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT  BYTE;    (* compatible to all 8 bit quantities. *)

TYPE    SetOfBytes      = ARRAY [0..16] OF BITSET;


(*----- Category : Scanning Sequences of Bytes --------------------------*)

PROCEDURE ScanWhileNot  (VAR Src        : ARRAY OF BYTE;
                             For        : INTEGER;
                             Equal      : BYTE)
                        : INTEGER;

(* Scan the Src buffer while its component is not equal the value Equal,
   or until For bytes are scanned or until all bytes in the buffer are
   scanned whichever occurs first. Return the number of bytes scanned.*)


PROCEDURE ScanWhileIn   (VAR Src        : ARRAY OF BYTE;
                             For        : INTEGER;
                         VAR In         : SetOfBytes)
                        : INTEGER;

(* Scan the Src buffer while its component is in the byteset In,
   or until For bytes are scanned or until all bytes in the buffer are
   scanned whichever occurs first. Return the number of bytes scanned.*)


PROCEDURE ScanWhileNotIn(VAR Src        : ARRAY OF BYTE;
                             For        : INTEGER;
                         VAR NotIn      : SetOfBytes)
                        : INTEGER;

(* Scan the Src buffer while its component is not in the byteset In,
```

```
   or until For bytes are scanned or until all bytes in the buffer are
   scanned whichever occurs first. Return the number of bytes scanned.*)


(*----- Category : Copying Sequences of Bytes -------------------------*)

PROCEDURE CopyFor       (VAR Dst, Src   : ARRAY OF BYTE;
                             For        : INTEGER)
                        : INTEGER;

(* Copy the Src buffer into the Dst buffer, until For bytes were copied,
   or until the Src buffer is exhausted or until the Dst buffer is full,
   whichever occurs first. Return the numer of bytes copied. *)


PROCEDURE CopyWhileNot  (VAR Dst, Src   : ARRAY OF BYTE;
                             For        : INTEGER;
                             Equal      : BYTE)
                        : INTEGER;

(* Copy the Src buffer into the Dst buffer, while the component of the Src
   buffer does not equal the byte Equal, or until For bytes were copied,
   or until the Src buffer is exhausted or until the Dst buffer is full,
   whichever occurs first. Return the numer of bytes copied. *)


PROCEDURE CopyWhileIn   (VAR Dst, Src   : ARRAY OF BYTE;
                             For        : INTEGER;
                         VAR In         : SetOfBytes)
                        : INTEGER;

(* Copy the Src buffer into the Dst buffer, while the component of the Src
   buffer is in the byteset In, or until For bytes were copied,
   or until the Src buffer is exhausted or until the Dst buffer is full,
   whichever occurs first. Return the numer of bytes copied. *)


PROCEDURE CopyWhileNotIn(VAR Dst, Src   : ARRAY OF BYTE;
                             For        : INTEGER;
                         VAR NotIn      : SetOfBytes)
                        : INTEGER;

(* Copy the Src buffer into the Dst buffer, while the component of the Src
   buffer is not in the byteset In, or until For bytes were copied,
   or until the Src buffer is exhausted or until the Dst buffer is full,
   whichever occurs first. Return the numer of bytes copied. *)


(*----- Category : Large Sets (including SetOfBytes above) ------------*)

PROCEDURE Clear         (VAR Dst        : ARRAY OF BYTE);

(* Clear all bits in Dst, i.e. make Dst the empty set. *)


PROCEDURE Or            (VAR Dst, Src   : ARRAY OF BYTE);

(* Calculate the logical or between Dst and Src buffer and assign the
   result to Dst. If Dst and Src are sets, this is equivalent to the
   set operation Dst:= Dst + Src; If the Src buffer is shorter than the
   Dst buffer, then Src is reused as long as necessary. *)


PROCEDURE AndNot        (VAR Dst, Src   : ARRAY OF BYTE);
```

```
(* Calculate the logical difference between Dst and Src buffer and assign
   the result to Dst. If Dst and Src are sets, this is equivalent to the
   set operation Dst:= Dst - Src; If the Src buffer is shorter than the
   Dst buffer, then Src is reused as long as necessary. *)


PROCEDURE And           (VAR Dst, Src   : ARRAY OF BYTE);

(* Calculate the logical and between Dst and Src buffer and assign the
   result to Dst. If Dst and Src are sets, this is equivalent to the
   set operation Dst:= Dst * Src; If the Src buffer is shorter than the
   Dst buffer, then Src is reused as long as necessary. *)


PROCEDURE Xor           (VAR Dst, Src   : ARRAY OF BYTE);

(* Calculate the exclusive or between Dst and Src buffer and assign the
   result to Dst. If Dst and Src are sets, this is equivalent to the
   set operation Dst:= Dst / Src; which is also called symmetric set
   difference. If the Src buffer is shorter than the Dst buffer, then
   Src is reused as long as necessary. *)


PROCEDURE Tst           (VAR Src        : ARRAY OF BYTE;
                             Element    : INTEGER)
                        : BOOLEAN;

(* Return TRUE, if the given Element is contained in Src, FALSE if it is
   not or if Src does not contain enogh elements at all. *)


PROCEDURE Incl          (VAR Dst        : ARRAY OF BYTE;
                             Element    : INTEGER);

(* Include the given Element into Dst, provided, Dst contains enough
   elements. *)


PROCEDURE Excl          (VAR Dst        : ARRAY OF BYTE;
                             Element    : INTEGER);

(* Exclude the given Element from Dst. *)


END Bytes.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) CmdLine;
(* .      Function      : Standard Commandline Support Functions        *)
(* .      Version  Date : 19:39  16.10.1988                             *)
(* Authors              : R.Huetter                                     *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1988, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

  Tools and utilities of the SPC-MODULA-2 language system are called via
  a command line, which's format is common to all standard utilities. We
  recommend to use this format with further utilities too, to perfectly
  fit into the calling convention of the overall system.

  The commandline has the following structure, where [](n..m) denotes a
  component, which must occur n to m times. (n..m), if not explicitly
  given defaults to (0..1).

  <cmdline>   ::= <utility> [" " <fileargs>] [" " <options>]
  <fileargs>  ::= <filename> [" " <filename>](0..3)
  <options>   ::= ["-" <optionchar> [["'"] <string> ["'"]]](0..16)
  <optionchar>::= "A"|"B"| ... |"y"|"z"
  <utility>   ::= <filename>

  A utility may return a result string, which details the success or
  failure of the utility. The result string is empty by default.

  To properly correlate the command line with a utility, a utility must
  parse its command line immidiatly after its start and  must store its
  result just before its termination.
  *)


(*----- Category : Primitives -------------------------------------------*)

PROCEDURE Set           (    Command    : ARRAY OF CHAR);

(* Redefine the current command line and reset commandline parsing. *)


PROCEDURE UtilityName   (VAR Name       : ARRAY OF CHAR);

(* Reset command line parsing and answer the utility name, which is
   always present. *)


PROCEDURE FileArg       (VAR Arg        : ARRAY OF CHAR
                        : BOOLEAN;

(* Answer, whether there is at least one more file argument and return
   it's name. *)


PROCEDURE Option        (VAR OptionChar : CHAR;
                         VAR OptionStr  : ARRAY OF CHAR)
                        : BOOLEAN;

(* Answer, whether there is at least one more option and return the
   option character and the optional option string. *)
```

```
PROCEDURE Get            (VAR Text       : ARRAY OF CHAR);

(* Get the part of the command line not yet parsed. May be used to get
   a non-standard command line, or to check, that command line is empty. *)


PROCEDURE ResultIs       (    Done       : BOOLEAN;
                              Result     : ARRAY OF CHAR);

(* Store the result of a command. *)


PROCEDURE Result         (VAR Done       : BOOLEAN;
                          VAR Result     : ARRAY OF CHAR);

(* Get the result of a command. *)


END CmdLine.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Environment;
(* .      Function      : Managing Environment Variables                *)
(* .      Version/Date  : 1.00  20.1.88                                 *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract ---------------------------------------**

  Environment maintains a list of socalled enviroment variables. These
  are labeled strings, which are read from the profile at system startup
  and saved upon termination. Storing attributes in environment variables
  is thus a simple way to save information for the next invocation of an
  utility.

  Another part of environment decouples filenames from the operating
  systems conventions by transforming commonly used file types to system
  dependent extensions. Language tools search paths are stored within
  environment variables and are therefore managed by environment, too.
*)


(*----- Category : Types and Data ----------------------------------------*)

TYPE    FileTypes       = (Mod, Def, Obm, Sbm, Rfm, Cmd, Lst, Cnf, Bak,
                           Tmp, Lib, Prg, Other, None);


(*----- Category : Environment Variable ----------------------------------*)

PROCEDURE Get           (    VarName    : ARRAY OF CHAR;
                         VAR String     : ARRAY OF CHAR)
                        : BOOLEAN;

(* If environment variable VarName exists, then get its content into
   String and asnwer TRUE else answer FALSE. *)


PROCEDURE GetIndexed    (    Index      : INTEGER;
                         VAR VarName    : ARRAY OF CHAR;
                         VAR String     : ARRAY OF CHAR)
                        : BOOLEAN;

(* Environment variables are indexed from 1 to n, where n is the number
   of variables. If Index is not greater than n, then return the indexed
   variables name in VarName and its content in String and answer TRUE
   else answer FALSE. *)


PROCEDURE Set           (    VarName    : ARRAY OF CHAR;
                             String     : ARRAY OF CHAR);

(* Set or modify the environment variable VarName using String. *)


(*----- Category : Search Pathnames --------------------------------------*)

PROCEDURE GetFilename   (VAR Name       : ARRAY OF CHAR;
                             Preference : INTEGER;
                             Type       : FileTypes)
                        : BOOLEAN;
```

```
(* Construct a filename, using the given Name and Type. If Preference
   is 0, then only change the filename extension, else use the search
   path indexed by preference. Search pathes are determined by
   environment variables Path<i> and ObjPath<i>, depending on whether
   Type is an object or a source type. If Preference is not greater
   than the number of paths configured answer TRUE else answer FALSE. *)


PROCEDURE GetFileType   (    Name       : ARRAY OF CHAR)
                        : FileTypes;

(* Answer the type of the given Name by inspection the filename
   extension field. *)


PROCEDURE Save;

(* Save the environment variables into the profile. *)


END Environment.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Frags;
(* .      Function      : Text Fragments Management                      *)
(* .      Version  Date : 12:21   6. 9.1988                              *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

  A fragment is a packet containing an arbitrary number of characters,
  called the fragment's content, and having a certain type. The type
  normally determines the interpretation of the content. Both, type
  and content are irrelevant to the module and are maintained by the
  calling module. The type is any character between <blank> and 'z'.
  The content may be any sequence of characters not containing 0C
  up to a length of 250 characters.

  Fragments are stored in a fast and memory efficient way, which in
  detail is transparent to the caller, as long, as memory is available.

  The caller may, for example, assign fragment types like "plain text",
  "linefeed", "font-control", "graphics", etc. The content is then
  reinterpreted to detail the fragment type at hand, e.g. to specify
  the font to be used from now on.

  A cursor is maintained to indicate the hot spot within the sequence
  of fragments. The cursor specifies a certain fragment called hot
  fragment. The cursor may be moved to the hot fragment's predecessor
  or its successor.
  *)


(*----- Category : Types and Data --------------------------------------*)

TYPE    File;


TYPE    Results         = (Done         ,
                           NotDone      ,
                           NoMemory     ,
                           FileNotFound ,
                           IoError      ,
                           IllFiletype  );

VAR     Result          : Results;


TYPE    Types           = [' '..'z'];

CONST   ReservedType    = ' ';
        BeginOfFile     = 'A';
        EndOfFile       = 'B';
        PlainText       = 'C';
        LineFeed        = 'D';

TYPE    Text            = ARRAY [0..252] OF CHAR;
        TextPtr         = POINTER TO Text;


(*----- Category : Initialisations -------------------------------------*)

PROCEDURE Open          (VAR aFile      : File;
```

```
                              Name       : ARRAY OF CHAR;
                              TextOnly   : BOOLEAN);

(* Open the named diskfile and read its content into the newly created
   fragments file F. If TextOnly is specified, then a standard textfile
   is read, and fragments are of type PlainText or Linefeed. Otherwise
   a true fragments file is expected, and it's data is read
   transparently. *)


PROCEDURE Create        (VAR aFile      : File);

(* Create a new fragments file. *)


PROCEDURE SaveAs        (    aFile      : File;
                             Name       : ARRAY OF CHAR;
                             TextOnly   : BOOLEAN);

 * Save the fragments file F to disk under Name. If TextOnly is set,
   then only fragments of type PlainText and LineFeed are saved to a
   standard text file. *)


PROCEDURE Abandon       (    aFile      : File);

(* Delete fragments file F. Do not save its content. *)


(*----- Category : Cursor Positioning ---------------------------------*)

PROCEDURE Next          (    aFile      : File)
                        : Types;

(* Make the next fragment the hot fragment and answer its type. If the
   hot fragment is of type EndOfFile, then Result is NotDone. *)


PROCEDURE Prev          (    aFile      : File)
                        : Types;

(* Make the previous fragment the hot fragment and answer its type. If
   the hot fragment is of type BeginOfFile, Then Result is NotDone. *)


PROCEDURE Current       (    aFile      : File)
                        : Types;

(* Answer the hot fragment's type. *)


PROCEDURE OccurrencesOf (    aFile      : File;
                             Type       : CHAR)
                        : INTEGER;

(* Answer the number of fragments of Type before the hot fragment. *)


(*----- Category : Fragments ------------------------------------------*)

PROCEDURE Insert        (    aFile      : File;
                             Type       : Types;
                         VAR Content    : ARRAY OF CHAR);

(* After the hot fragment insert a new fragment of Type with an
```

```
   initial Content. If hot fragment is BeginOfFile, the Result is
   NotDone. *)

PROCEDURE Delete        (    aFile      : File);

(* Delete the hot fragment. If hot fragment is BeginOfFile or
   EndOfFile then Result is NotDone. *)


PROCEDURE Retype        (    aFile      : File;
                             Type       : CHAR);

(* Change the hot fragment's type to Type, provided it is neither
   BeginOfFile nor EndOfFile. *)


PROCEDURE ContentOf     (    aFile      : File;
                         VAR Content    : ARRAY OF CHAR);

(* Answer the content of the hot fragment. *)


PROCEDURE LengthOf      (    aFile      : File)
                        : INTEGER;

(* Answer the length of the hot fragment. *)


PROCEDURE PointerOf     (    aFile      : File)
                        : TextPtr;

(* Answer a pointer to the current fragments content. The content is 0C
   terminated. The pointer may only be used for READ ONLY access to the
   fragment's content. *)


PROCEDURE Test          (    aFile      : File);

END Frags.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) JCL;
(* .      Function      : Job Control Level Primitives                  *)
(* .      Version  Date :  8:13  19. 1.1989                             *)
(* Product Name/Ident   : SPC                                           *)
(* Copyright            : (c) 1988, R. Huetter, D7500 Karlsruhe         *)


(*----- Category : Module Abstract ---------------------------------------**

  Within the SPC MODULA-2 language system MODULA-2 is used as the job
  control language. This module provides some of the primitives commonly
  use on the job control level. Of course, all the other libraries may be
  used, as well. Describing jobs as MODULA-2 programs has thus several
  advantages. The most important is, that the full power of MODULA-2 and
  its libraries is available on the job control level. *)


(*----- Category : Types and Data ----------------------------------------*)

IMPORT  Clock;


TYPE    (* The results of JCL primitives are members of a simplified    *)
        (* error type. If an error occurs, the error handler is invoked *)
        (* to to some error processing. The default error handler just  *)
        (* prints a message to inout. It can be called by an application*)
        (* dependent errr handler.                                      *)

        Results         = (Ok, Syntax, Hard, Soft);

        ErrorHandler    = PROCEDURE ( (* Result  : *) Results,
                                      (* Message : *) ARRAY OF CHAR,
                                      (* Detail  : *) INTEGER);


(*----- Category : Errors ------------------------------------------------*)

PROCEDURE OnErrorDo     (    WhatToDo   : ErrorHandler);

(* Redefine the error handler, that will be invoked if an error occurs. *)


PROCEDURE DefaultHandler(    aResult    : Results;
                             aMessage   : ARRAY OF CHAR;
                             aDetail    : INTEGER);

(* Print an error message to InOut. *)


(*----- Category : Messages and Inquiries --------------------------------*)

PROCEDURE Echo          (    aText      : ARRAY OF CHAR);

(* Print a message to InOut. *)


PROCEDURE Query         (    aText      : ARRAY OF CHAR)
                        : CHAR;

(* Print a message and read a character. Return that character. *)
```

```
PROCEDURE Inquire       (    aText      : ARRAY OF CHAR;
                         VAR aResponse  : ARRAY OF CHAR);

(* Print a message and ask for some textual response. *)


(*----- Category : Run Programs ----------------------------------------*)

PROCEDURE Call          (    aCommand   : ARRAY OF CHAR);

(* Set the command line to aCommand an call a program via the linking
   loader. *)


(*----- Category : File Primitives -------------------------------------*)

PROCEDURE Cp            (    Source     : ARRAY OF CHAR;
                             Destin     : ARRAY OF CHAR);

(* Copy Source files to Destin. If Source contains wildcards, then
   Destin is expected to be a directory name, else Destin may be a file
   name, as well. *)


PROCEDURE Mv            (    Source     : ARRAY OF CHAR;
                             Destin     : ARRAY OF CHAR);

(* Move Source files to Destin. If Source contains wildcards, then
   Destin is expected to be a directory name, else Destin may be a file
   name, as well. *)


PROCEDURE Rm            (    aFile      : ARRAY OF CHAR);

(* Delete aFile, while aFile may contain wildcards. *)


PROCEDURE Timestamp     (    aFile      : ARRAY OF CHAR;
                         VAR aStamp     : Clock.Time);

(* Return the timestamp of aFile. *)


PROCEDURE Exists        (    aFile      : ARRAY OF CHAR)
                        : BOOLEAN;

(* Answer, whether aFile exists at all. *)


PROCEDURE Cd            (    aFolder    : ARRAY OF CHAR);

(* Change the current directory to aFolder. *)


PROCEDURE Wd            (VAR aFolder    : ARRAY OF CHAR);

(* Return the current directory (also called working directory. *)


PROCEDURE MkDir         (    aFolder    : ARRAY OF CHAR);

(* Create a new directory (also called folder). *)


PROCEDURE RmDir         (    aFolder    : ARRAY OF CHAR);
```

```
(* Remove a directory. *)

PROCEDURE Space          (    aVolume    : ARRAY OF CHAR)
                         : LONGCARD;

(* Return the number of bytes left on aVolume. *)

END JCL.
```

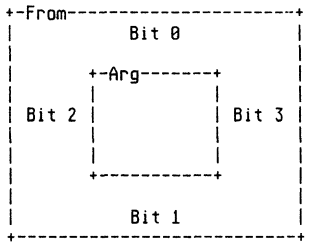
```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .       Name          : *) Loader;
(* .       Function      : SPC Dynamic Loader Utility                    *)
(* .       Version  Date : 15: 4  24. 9.1988                             *)
(* Product Name          : SPC                                           *)
(* Copyright             : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract ---------------------------------------**

  Loader is the dynamic linking loader of the SPC-MODULA-2 system. Loader
  allows threads (i.e. group of modules) to be brought into memory and
  added to the list of resident modules. The main module of a thread is
  then be started and runs as a subprogram of the calling thread.

  All required linkage is done dynamically by the loader module while the
  modules are read from mass storage (so called loadtime linking). *)


(*----- Category : Load and Start a Thread ------------------------------*)

PROCEDURE Call          (    Name       : ARRAY OF CHAR;
                             Hold       : BOOLEAN;
                         VAR ErrorMsg   : ARRAY OF CHAR)
                        : BOOLEAN;

(* Load the thread given by Name into memory, do linkage as required and
   start the thread as a subprogram or answer FALSE and a error message.
   If hold is TRUE, then leave the thread in memory and linked so that
   it may be started again within reading it from mass storage. *)


END Loader.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Process;
(* .      Function      : Multitasking Kernel                          *)
(* .      Version  Date : 21:58  20. 6.1988                            *)
(* Product Name/Ident   : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)

(*----- Category : Module Abstract -----------------------------------**

  Process implements a simple coroutine scheduler to simplify programming
  of concurrent programs. A Process is a coroutine as provided by module
  Coroutines. Processes can be created, which implicitly enters it into
  the list of ready Processes. From that time on, the Process runs,
  whenever it is the first Process in the ready list.

  A Process runs, until it explicitly Relinquishes (calls the Process
  module to perform a dispatch cycle). Another way to enter run a
  dispatch cycle is to consume units from a Resource.

  A Resource is a pool of units, which are produced by some Process and
  are consumed by another Process. Whenever a Process tries to consume
  more units, than are available, the Process module enters a dispatch
  cycle and resumes another process. The waiting Process is put into the
  wait list for the requested Resource. If a process produces units,
  Process checks, if there is a Process waiting for that Resource, and
  whether its request for units can be met by the amount of currently
  available units. If so, all waiting Processes are put into the ready
  list, until there are no longer enough units.

  Resources are therefore a means of process synchronisation.

  It is essential, that the producing module keeps on running, until it
  Relinquishes or calls for more units, than are available. At that time
  the first module on the ready list continues processing.

  NOTE: This is NOT a true multitasking, since dispatch cycles are
        triggered by the processes explicitly (rather than by a timer or
        external interrupt). An endless loop without calls to Relinquish
        or Consume prevents other processes from running.

  A Resource is processes do produce and consume.
  Whenever a process wants to consume a Resource, and the Resource is
  empty, the process is suspendend, and some other process is activated. *)


(*----- Category : Types and Data -------------------------------------*)

TYPE    Resource;  (* something processes do produce and consume *)


(*----- Category : Initialization -------------------------------------*)

PROCEDURE Init;

(* Initialize the scheduler and enter the caller as a process into the
   ready list. *)


PROCEDURE Term;

(* Terminate the scheduler. Continue processing with the caller of Init.
   All processes are stopped. *)
```

```
(*----- Category : Process Creation ----------------------------------*)

PROCEDURE Create        (    StartAt    : PROC;
                             StackSize  : CARDINAL;
                         VAR Done       : BOOLEAN);

(* Create a new process and start its execution at StartAt, giving an
   initial StackSize. If the memory request for the stacksegemnt fails,
   Done will be FALSE. *)


PROCEDURE Delete;

(* Delete the calling process. *)


(*----- Category : Process Synchronisation ---------------------------*)

PROCEDURE Relinquish;

(* Allow a dispatch cycle to be performed. *)


PROCEDURE Produce       (    Arg        : Resource);

(* Produce one unit of a Resource and ready a waiting process, if any. *)


PROCEDURE Consume       (    Arg        : Resource);

(* Consume one unit of a Resource, wait if no units available. *)


PROCEDURE Available     (    Arg        : Resource)
                        : BOOLEAN;

(* Answer, if units of a Resource are available. *)


PROCEDURE InitResource  (VAR Arg        : Resource;
                             Initial    : CARDINAL);

(* Initialise a Resource, and give Initial units. *)


END Process.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Rectangle;
(* .      Function      : Operations on Rectangles                      *)
(* .      Version       : 1.0                                           *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

Provide operations on rectangles. *)


(*----- Category : Types and Data --------------------------------------*)

TYPE    Instance        = RECORD
          X, Y, W, H    :   INTEGER;
                          END;


(*----- Category : Rectangle Primitives --------------------------------*)

PROCEDURE Preset        (VAR Arg        : Instance;
                             x, y, w, h : INTEGER);

(* Preset Arg to the value x/y-w/h. *)


PROCEDURE MoveRel       (VAR Arg        : Instance;
                             x, y       : INTEGER);

(* Move Arg by the increment x/y. *)


PROCEDURE MoveAbs       (VAR Arg        : Instance;
                             x, y       : INTEGER);

(* Move Arg to x/y. *)


PROCEDURE Resize        (VAR Arg        : Instance;
                             w, h       : INTEGER);

(* Set Arg's new size to w/h. *)


PROCEDURE Combine       (VAR Arg        : Instance;
                             With       : Instance);

(* Answer the union of Arg and With in Arg. *)


PROCEDURE Intersect     (VAR Arg        : Instance;
                             With       : Instance)
                        : BOOLEAN;

(* Intersect Arg with With and return the result as Arg. If the result
is not empty return TRUE else FALSE. *)


PROCEDURE Subtract      (    Arg        : Instance;
                             From       : Instance)
                        : BITSET;
```

```
(* Subtract Arg from From and return the result as a bitset with the
following implication : If bit 0 is on, then there is a resulting
Rectangle above Arg. If bit 1 is on , then there is are result below
Arg. If bit 2 is on, then we have a result to the left and if bit 3 is
on, then there is a result ro the right of Arg. The resulting Rectangles
can be obtained by calling SubResult with the bit number (0..3).
```




```
*)


(*----- Category : Inquiries ------------------------------------------*)

PROCEDURE SubResult     (    Arg        : Instance;
                             From       : Instance;
                        VAR  Result     : Instance;
                             N          : INTEGER);

(* Return the Nth (0<=N<=3) resulting Rectangle from the operation
From-Arg. *)


PROCEDURE Empty         (    Arg        : Instance)
                        : BOOLEAN;

(* Answer, whether Arg  is empty. *)


PROCEDURE Includes      (    Arg        : Instance;
                             x, y       : INTEGER)
                        : BOOLEAN;

(* Answer, if Arg includes the point x/y. *)


END Rectangle.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) SplittedPieces;
(* .      Function      : Splitted Piece List Management                *)
(* .      Version  Date : 10:54  16.10.1988                             *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract -------------------------------------**
*)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

TYPE    AllocateProc    = PROCEDURE (VAR LONGINT) : ADDRESS;
        DeallocateProc  = PROCEDURE (VAR ADDRESS);

        List            ;


(*----- Category : List Primitives -------------------------------------*)

PROCEDURE Create        (VAR Self       : List;
                             Alloc      : AllocateProc;
                             Dealloc    : DeallocateProc;
                             ChunkSize  : LONGINT)
                        : BOOLEAN;

PROCEDURE Delete        (VAR Self       : List);

PROCEDURE Get           (    Self       : List;
                             Amount     : LONGINT;
                         VAR BlockPtr   : ADDRESS);

PROCEDURE Put           (    Self       : List;
                         VAR BlockPtr   : ADDRESS);


END SplittedPieces.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) SSWiS;
(* .      Function      : Small Systems Windowing Standard            *)
(* .      Version/Date  : 0.01  06.03.88                              *)
(* Product Name         : SSWiS                                       *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe     *)


(*----- Category : Module Abstract ---------------------------------------**
```

The Small Systems Windowing Standard is an interface library to an operating system independent windowing system. The implementation module of SSWiS will map the interface functions to the resident windowing system. By only using SSWiS functions for window-, menue-, event-, etc. processing, applications remain portable between different SSWiS implementations. Important concepts are described below.

- Client
  Modules, that want to use SSWis, register as SSWiS clients. Every client has its own pool of resources. Many clients may coexist at the same time. Different clients may belong to the same or to different programs. Clients are known to SSWiS via ModuleHandles.

- Current Module
  The Current Module is the module, which receives subsequent keyboard input. The current module may be selected under program control as well as interactively.

- Window
  Windows allow SSWiS to give parts of screen space to different clients. The window arrangement can be influenced by the clients and the user, as well. Each client may use several windows. Windows are known to SSWiS via WindowHandles.

- World
  While screen space is shared on the basis of windows, each window opens a view into a usually larger World (e.g. a textfile, a CAD drawing, etc.). The world is discribed in world Coordinates, while points on the screen are referenced via ScreenCoordinates. World Coordinates in SSWiS are 32 bits wide.

- Menu
  Menus are provided to interactivly select functions via menu items. Items are group to titles. Each client has one set of menu titles, which can be labelled, mask, etc. Titles and items are known to SSWiS via TitleHandles and ItemHandles. As the current module changes, SSWiS presents the titles of the current module.

- Form
  A Form is a means of interactive data entry. SSWiS implements a method to notify the user, to ask the user about something and to identify the module.

- Callback Procedure
  Clients register at SSWiS, to share input devices and screen space with other clients. While controlling the screen and the input devices, SSWiS calls procedures, that have been presented by the clients at initialisation time. In particular two types of procedures are use, one to Accept events and one to Restore the window content. Since the procedures are called back at a later time, they are commonly referred to as Callback Procedures.

- Cursor

The Cursor represents the mouse position with respect to the displ screen. The cursor's shape can be configured, to meet differe echoing requirements.

- Caret
  The Caret represents another position, which is explicitly controlled by the application. In text applications the caret may be used to mark the point of text insertion. Since the caret is controlled by SSWiS directly, it may be moved very rapidly and does not require a screen restore operation.

- Events
  Since many clients share the same screen and the same input devices, SSWiS must distribute the input device activity to its clients. The input is communicated as events to the clients. An event is described in detail by an EventReport. To process an event, the clients Accept procedure is called back.

*)

```
(*----- Category : Handles for Various Objects -------------------------*)

TYPE    ModuleHandles   = [0..31]; (* limited number of client Modules  *)
        WindowHandles   = [0.. 7]; (* Windows per Module                *)
        TitleHandles    = [0.. 6]; (* Menu Titels per Module            *)
        ItemHandles     = [0.. 7]; (* Items per Menu Title              *)


(*----- Category : World and Screen Coordinate space -------------------*)

        (* The applications drawing are is determined by the world-     *)
        (* space. The application may actually use a smaller part       *)
        (* thereof which will be called the world below.                *)

        Coordinates     = LONGINT;        (* wc-space is 32 bit wide.   *)

        Points          = RECORD          (* an absolute point within   *)
          X, Y          :   Coordinates;  (* the world.                 *)
                          END;            (*                            *)

        Lines           = RECORD          (* an absolute line given by  *)
          A, B          :   Points;       (* start and end points wihin *)
                          END;            (* the world.                 *)


        (* The screen space is the area, where output must be generated.*)
        (* Mapping of the applications world to the screen is done by   *)
        (* adding an offset to the world coordinates and restricting    *)
        (* output to the windows extent. This mapping must be done by   *)
        (* the application during restore operations.                   *)

TYPE    ScreenCoordinates = INTEGER;      (* screen space is 16 bit wide.*)

        ScreenPoints    = RECORD                (* a point in screen    *)
          X, Y          :   ScreenCoordinates;  (* space.               *)
                          END;                  (*                      *)

        ScreenLines     = RECORD                (* same as Lines in     *)
          A, B          :   ScreenPoints;       (* screen space.        *)
                          END;                  (*                      *)


        (* The application is notified about several events occuring    *)
        (* during window operation. Notification is based on callback   *)
```

```
        (* procedures provided by the application at window creation    *)
        (* time.                                                        *)

TYPE    RestoreProc     = PROCEDURE ((* Owner      : *) ModuleHandles,
                                     (* Window     : *) WindowHandles,
                                     (* WorldArea  : *) Lines,
                                     (* Offset     : *) Points);

        (* The application is requested to restore part of their world  *)
        (* on the screen. WorldArea specifies position and size of the  *)
        (* area in wc-space, while Offset defines the position of the   *)
        (* window content in wc-space. Screen coordinates are obtained  *)
        (* by adding Offset to the worldcoordinates at hand. As long as *)
        (* world-coordinates are within WorldArea, it is guaranteed,    *)
        (* that mapped screen-coordinates are within screen-space.      *)


(*----- Category : Standard Keyboard ------------------------------------*)

        (* Keystrokes are entered via the standard keyboard, which's    *)
        (* layout is considered to be system independent. It is mapped  *)
        (* by SWIS to the keyboard actually attached.                   *)

CONST   NILKey          = -1;

        UpArrow         = 256;
        DownArrow       = 257;
        LeftArrow       = 258;
        RightArrow      = 259;
        Help            = 260;
        Undo            = 261;
        Insert          = 262;
        Clear           = 263;
        NumLeftBracket  = 264;
        NumRightBracket = 265;
        NumSlash        = 266;
        NumAsterisk     = 267;
        NumMinus        = 269;
        NumPlus         = 270;
        NumEnter        = 271;
        NumDot          = 272;
        Num0            = 273;
        Num1            = 274;
        Num2            = 275;
        Num3            = 276;
        Num4            = 277;
        Num5            = 278;
        Num6            = 279;
        Num7            = 280;
        Num8            = 281;
        Num9            = 282;
        F1              = 283;
        F2              = 284;
        F3              = 285;
        F4              = 286;
        F5              = 287;
        F6              = 288;
        F7              = 289;
        F8              = 290;
        F9              = 291;
        F10             = 292;

TYPE    AllKeys         = [ -1..511];  (* all keys known by SSWiS.      *)
        ControlKeys     = [  0.. 31];  (* ASCII control keys, exc DEL   *)
        AlphaKeys       = [ 32..126];  (* ASCII printable keys          *)
```

```
        NationalKeys    = [128..255];  (* national keys, system dep.    *)
        EditKeys        = [256..511];  (* editing keys commonly used    *)

        MetaKeys        = (Shift     ,
                           Control   ,
                           Alternate );

        SetOfMetaKeys   = SET OF MetaKeys;


(*----- Category : Mouse Styles ----------------------------------------*)

        (* The mousecursor can be configured to have different styles,  *)
        (* and within each style (except off) to have different sprites.*)

TYPE    MouseStyles     = (Off          ,
                           SpriteOnly   ,
                           RubberLine   ,
                           RubberBox    ,
                           DragBox      ,
                           ApplSpecific );

        MouseSprites    = (StdArrow     ,
                           Hourglass    ,
                           Disk         ,
                           StdCross     ,
                           StdText      ,
                           FlatHand     ,
                           Finger       );

(*----- Category : Input Events ----------------------------------------*)


TYPE    KeyEvent        = RECORD
          MetaKey       :   SetOfMetaKeys;
          Position      :   Points;
          Keys          :   ARRAY [0..20] OF AllKeys;
                        END;


TYPE    ButtonActivities= (Clicked      ,   (* pressed and released      *)
                           DoubleClicked,  (* two very fast clicks      *)
                           Pressed      ,  (* pressed and held          *)
                           Released     ,  (* released after pressed    *)
                           WindowExit   ,  (* window contents exited    *)
                           Moved        ); (* snap area exited          *)

TYPE    MouseEvent      = RECORD
          MetaKey       :   SetOfMetaKeys;
          ButtonDown    :   BOOLEAN;
          Position      :   Points;
          Activity      :   ButtonActivities;
                        END;


TYPE    MenuEvent       = RECORD
            Title       :   TitleHandles;
            Item        :   ItemHandles;
                        END;


TYPE    EventTypes      = (Notification,  (* window parameters changed  *)
                           Keyboard    ,  (* keystrike occurred         *)
                           Mouse       ,  (* moved or button clicked    *)
                           Menu        ,  (* menu item selected         *)
```

```
                          Message      , (* piping system              *)
                          Timer        , (* system grants idle time    *)
                          Identification); (* ... requested            *)

TYPE    EventReports    = RECORD
          CASE Type     :   EventTypes OF
          | Keyboard        : Strokes   : KeyEvent;
          | Mouse           : Clicks    : MouseEvent;
          | Menu            : Selection : MenuEvent;
          | Message         : Pipe      : ARRAY [0..40] OF CHAR;
          | Timer           :
          | Identification  :
          END;
                        END;

        (* Events are communicated to clients by calling the Accept     *)
        (* callback procedure, below. Accept is given when registering. *)

TYPE    AcceptProc      = PROCEDURE (     (* Owner  : *) ModuleHandles,
                                          (* Window : *) WindowHandles,
                                    VAR (* Report : *) EventReports);

        (* Owner is requested to process an event. The event is further *)
        (* detailed by Report. The event has been received via Window.  *)


(*----- Category : Window Layout ---------------------------------------*)

        (* A windows has WindowElements to allow for interactive        *)
        (* manipulation of window parameters. These elements may or may *)
        (* not be present in a particular window.                       *)

TYPE    WindowElements  = (MessageLine  ,
                           Iconiser     ,
                           Fuller       ,
                           Sizer        ,
                           XScroller    ,
                           YScroller    );

        SetOfWindowElements = SET OF WindowElements;


(*----- Category : Useful Constants ------------------------------------*)

VAR     NullPoint       : Points;
        NullLine        : Lines;
        NullScreenPoint : ScreenPoints;
        NullScreenLine  : ScreenLines;

        NeverClip       : Lines;

        ScreenSize      : ScreenPoints;
        ScreenColours   : INTEGER;

        (* Switch to control umlauts on german keyboard. May be set     *)
        (* under program control or toggled by pressing both Shift keys.*)

        Umlauts         : BOOLEAN;


(*----- Category : Error Reporting -------------------------------------*)

TYPE    Results         = (Done,
                           NotDone,
                           NoMemory,
```

```
                        TooManyModules,
                        TooManyWindows,
                        TooManyTitles);

VAR     Result          : Results;          (* result of the last call. *)

(*----- Category : Operating SWIS ---------------------------------------*)

PROCEDURE Reinit;

(* Cause SSWiS to restore its desktop and window configuration. *)


PROCEDURE PollEvents;

(* Cause SSWiS to enter its event processing loop. *)


PROCEDURE Resync;

(* Cause SSWiS to process window related events *)


(*----- Category : Registering the Application -------------------------*)

PROCEDURE Register      (VAR Handle     : ModuleHandles;
                         ModuleName : ARRAY OF CHAR;
                         Accept     : AcceptProc);

(* Register a module as an application known by SSWiS. Events will
   be communicated by calling Accept. *)


PROCEDURE Deregister    (    Handle     : ModuleHandles);

(* Deregister the application and free all its resources. *)


(*----- Category : Window Creation and Deletion ------------------------*)

PROCEDURE CreateWindow  (    Owner      : ModuleHandles;
                         Window     : WindowHandles;
                         Restore    : RestoreProc);

(* Owner creates a window with will be refreshed using Restore. *)


PROCEDURE DeleteWindow  (    Owner      : ModuleHandles;
                         Window     : WindowHandles);

(* Delete a window and it's resources. *)


PROCEDURE PlaceWindowOnTop
                        (    Owner      : ModuleHandles;
                         Window     : WindowHandles);

(* Place the window on top of all other windows. *)


PROCEDURE IconiseWindow (    Owner      : ModuleHandles;
                         Window     : WindowHandles);

(* Place a window behind all other windows. *)
```

```
PROCEDURE SetWindowTitle(    Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Text       : ARRAY OF CHAR);

(* Set the window's title. *)


PROCEDURE SetWindowMessage
                         (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Text       : ARRAY OF CHAR);

(* Set the window's message. *)


PROCEDURE SetWindowElements
                         (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Elements   : SetOfWindowElements);

(* Configure the window to have Elements. *)


PROCEDURE ExplicitRestore
                         (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Clip       : Lines);

(* Do an explicit restore of a window. Assume, that changes have taken
   place only within the Clip area, which may be used to optimize the
   restore operation. *)


(*----- Category : Window's Position and Size --------------------------*)

(* The window's position and size on screen is partly controlled by the
   application and by the user via interactive tools. However, SSWiS is
   responsible to maintain certain conditions concerning screen layout,
   and it may be impossible to meet the requirements exactly. *)


PROCEDURE PositionWindow(    Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Corner     : ScreenPoints);

(* Position the upper left corner of the window's border. This is only
   a hint, and it is not guaranteed, that the window is positioned. *)


PROCEDURE SizeWindowContent
                         (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             MinSize    ,
                             IdealSize  ,
                             MaxSize    : ScreenPoints);

(* Resize the windows content. IdealSize is a hint to SSWiS. The
   window must not become smaller than MinSize and it must not become
   larger than MaxSize. SSWiS will choose a size between these
   limits and as near as possible to IdealSize. *)


PROCEDURE PositionOfWindow
```

```
                          (    Owner      : ModuleHandles;
                               Window     : WindowHandles;
                           VAR Corner     : ScreenPoints);

(* Answer the position of the window. *)


PROCEDURE SizeOfWindowContent
                          (    Owner      : ModuleHandles;
                               Window     : WindowHandles;
                           VAR Size       : ScreenPoints);

(* Answer the size of the window's content. *)


(*----- Category : World's Position and Size --------------------------*)

(* The world is the application's drawing area. The world is partially
   visible within the window's content. Moving the world to expose
   different parts in the window's content is controlled by either
   the application or by the user via interactive tools of the window
   itself. *)


PROCEDURE PositionWorld (    Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Corner     : Points);

(* Position the world, so that Corner (a point within the world)
   corresponds to the upper left corner of the window's content.
   Change of world's position causes a notification event. *)


PROCEDURE SizeWorld      (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Size       : Points);

(* Resize the world. The worlds size is used by SSWiS to prevent
   undefined holes within the window's content. Change of world's
   size causes a notification event. *)


PROCEDURE RasterWorld    (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             GridSpacing: Points);

(* Tell SSWiS, the the invisible grid of world is not 1 pixel wide, but
   GridSpacing pixels wide (in x and y). Gridspacing will be applied, when
   the user scrolls by steps through the world. *)


PROCEDURE PositionOfWorld
                          (    Owner      : ModuleHandles;
                               Window     : WindowHandles;
                           VAR Corner     : Points);

(* Answer the world's position. *)


PROCEDURE SizeOfWorld    (   Owner      : ModuleHandles;
                               Window     : WindowHandles;
                           VAR Size       : Points);

(* Answer the world's size. *)
```

```
(*----- Category : Mouse Configuration ---------------------------------*)

PROCEDURE ConfigureMouse(    Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             Style      : MouseStyles;
                             Sprite     : MouseSprites;
                             CoolSpot   : Points);

(* Owner configures the mouse to have Style and optionally show Sprite
   as long, as it is in Window. *)


PROCEDURE SetCaret       (   Owner      : ModuleHandles;
                             Window     : WindowHandles;
                             HotSpot    : Points);

(* Set the caret to the given position. *)


(*----- Category : Menu Primitives -------------------------------------*)

(* A Menu is a means of command selection. A Menu consists of several
   titles, where each title contains several items. Every window can
   have it's own menu. A menu item is selected under control of SSWiS.
   The Selection is communicated as an event to the owning module.

   The syntax of menu elements is :

   <MenuElement>  ::= <Attributes><String>
   <Attributes>   ::= ("M"|"C"|" ")
   <String>       ::= {CHAR}0..n

   where :

   the M attribute masks the entry, and no data can be entered,
   the C attribute checks the entry
*)


PROCEDURE SetMenuTitle  (    Owner      : ModuleHandles;
                             Title      : TitleHandles;
                             Definition : ARRAY OF CHAR);

(* Owner gives the Definition for the menu Title of Window using the
   above syntax. *)


PROCEDURE SetMenuItem   (    Owner      : ModuleHandles;
                             Title      : TitleHandles;
                             Item       : ItemHandles;
                             Definition : ARRAY OF CHAR);

(* Owner gives the Definition for the menu Item of Title of Window
   using the above syntax. *)


(*----- Category : Forms Primitives ------------------------------------*)

(* A Form is a means of interactive data entry. SSWiS has three standard
   forms, which may be used to notify the user, to ask the user about
   some parameters and last not least, to identify the current modules
   by product name, version author and copyrights.

   Forms may use buttons to label upto four push buttons, which can
```

```
   be interactively selected. Buttons are given in the syntax:

   <Button>  ::= [<Label>{"|"<Label>}0..3]

*)


PROCEDURE NotifyForm    (    Message    : ARRAY OF CHAR;
                             Buttons    : ARRAY OF CHAR;
                        VAR Result      : INTEGER);

(* Display a notification using Message and prompting Buttons. Result
   is the default Button on entry and the selected button upon return. *)


PROCEDURE AskForm       (    Message    : ARRAY OF CHAR;
                             Buttons    : ARRAY OF CHAR;
                             Options    : ARRAY OF CHAR;
                        VAR Answer      : ARRAY OF CHAR;
                        VAR OptionsRes  : BITSET;
                        VAR Result      : INTEGER);

(* Display an editable form. Message is some constant text, while Answer
   is preset by the caller and is edited by the user. Options are
   provided by up to 4 buttons. The selected buttons are given by
   OptionsRes and are returned via OptionsRes. Buttons are provided to
   exit the form, while Result specifies the default button and holds
   the selected button upon return. *)


PROCEDURE Identify      (    Program    : ARRAY OF CHAR;
                             Version    : ARRAY OF CHAR;
                             Author     : ARRAY OF CHAR;
                             Copyright  : ARRAY OF CHAR);

(* Use a standard method to identify the calling module by the given
   string arguments. *)


END SSWiS.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) System;
(* .      Function      : Modula-2 Runtime System for MC68000/ATARI ST  *)
(* .      Version  Date :  0:22  16.10.1988                             *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract --------------------------------------**
```

System's functionality is devided into the following categories :

1. Processor dependent instruction supplement

Module System provides the instructions, that the processor does not support. Since the code generator uses fixed indices to access these instructions, most of the operators must be declared in the order given below.

2. Dynamic linkable module chain

SPC-MODULA-2 contains a dynamic load feature, which allows modules to be brought into memory and integrated into the running system at runtime. This is normally done by a module called Loader. Necessary fixup and linkage must however be done by that module. System declares the structures used to maintain the modules in memory. Furthermore, funcions are provided to add groups of new modules and to start a main module within such a group as a socalled thread. After the thread has run, modules, which are no longer needed, are released from memory. Calls are available, to abort a thread due to some error conditions. HALT is another means to abort a thread.

The module chain can be inquired, to get some information about the loaded modules. Furthermore can a module be made resident, which causes it to remain in memory, even if it is not imported by any active module.

3. Module Termination

Each module may declare a procedure, whch is called prior to releasing the module. This may be used to do some cleanup and to orderly release resources on module termination. All imported modules remain intact, provided, that there are no import loops (module A imports B and B imports A).

4. Active Procedure Chain

A procedure invocation is described by a so called procedure frame. The chain of nested procedure calls may be traversed to get the procedure frame of the caller.

5. Exceptions

Exceptions are abnormal conditions that occur during program execution and normally indicate an error within the program. Exceptions may be detected by hardware or by software, In any case an exception handler is called to take some corrective action or to kill the running thread.

A thread (see above) may announce its own exception handler to do some application dependent processing. Otherwise it uses the default handler. The default handler again may be overriden (e.g. by the Loader to load the debugger, etc.).

```
  *)

(*----- Category : Types and Data -------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

VAR     BasePagePtr     : ADDRESS;

(*----- Category : Fixed Order of Operators ---------------------------*)

PROCEDURE HALTX       (* (    Error      : RegisterD0)  *);

PROCEDURE MULU32      (* (    Op1        : RegisterD0;
                               Op2        : RegisterD1) : RegistersD0andD1 *);

PROCEDURE DIVU32      (* (    Op1        : RegisterD0;
                               Op2        : RegisterD1) : RegistersD0andD1 *);

PROCEDURE MULS32      (* (    Op1        : RegisterD0;
                               Op2        : RegisterD1) : RegistersD0andD1 *);

PROCEDURE DIVS32      (* (    Op1        : RegisterD0;
                               Op2        : RegisterD1) : RegistersD0andD1 *);


PROCEDURE FADDs           (    adder      ,
                               addend     : REAL)       : REAL;

PROCEDURE FSUBs           (    minuend    ,
                               subtrahend : REAL)       : REAL;

PROCEDURE FMULs           (    multiplicand,
                               multiplier : REAL)       : REAL;

PROCEDURE FDIVs           (    dividend   ,
                               divisor    : REAL)       : REAL;

PROCEDURE FREMs           (    dividend   ,
                               divisor    : REAL)       : REAL;

PROCEDURE FCMPs           (    first      ,
                               second     : REAL)    (* : CCR *);

PROCEDURE FNEGs           (    toNeg      : REAL)       : REAL;

PROCEDURE FABSs           (    toAbs      : REAL)       : REAL;

PROCEDURE FLOATs          (    toFloat    : LONGINT)    : REAL;

PROCEDURE TRUNCs          (    toTrunc    : REAL)       : LONGINT;


PROCEDURE FADDd           (    adder      ,
                               addend     : LONGREAL)   : LONGREAL;

PROCEDURE FSUBd           (    minuend    ,
                               subtrahend : LONGREAL)   : LONGREAL;

PROCEDURE FMULd           (    multiplicand,
                               multiplier : LONGREAL)   : LONGREAL;

PROCEDURE FDIVd           (    dividend   ,
                               divisor    : LONGREAL)   : LONGREAL;
```

```
PROCEDURE FREMd         (    dividend   ,
                             divisor    : LONGREAL)   : LONGREAL;

PROCEDURE FCMPd         (    first      ,
                             second     : LONGREAL) (*: CCR *);

PROCEDURE FNEGd         (    toNeg      : LONGREAL)   : LONGREAL;

PROCEDURE FABSd         (    toAbs      : LONGREAL)   : LONGREAL;

PROCEDURE FLOATd        (    toFloat    : LONGINT)    : LONGREAL;

PROCEDURE TRUNCd        (    toTrunc    : LONGREAL)   : LONGINT;

PROCEDURE FLONG         (    toConvert  : REAL)       : LONGREAL;

PROCEDURE FSHORT        (    toConvert  : LONGREAL)   : REAL;


(*----- Category : Installing the Floatingpoint Emulator ---------------*)

PROCEDURE InstFPEmulator(    Emulator   : ADDRESS);

(* Install the system and hardware dependent floating point emulator *)


(*----- Category : Dynamic Linkable Module Chain -----------------------*)

TYPE    (* A module name is recorded in the object file with 16         *)
        (* significant characters. The module key is 6 bytes in length. *)

        ModuleNames     = ARRAY [0..15] OF CHAR;
        ModuleKeys      = ARRAY [0.. 2] OF INTEGER;

TYPE    ThreadIds       = [0..15];
        SetOfThreadIds  = SET OF ThreadIds;

        (* A module, which is linked to the module chain, is descriped  *)
        (* by the module descriptor and by the auxiliary descriptor. No *)
        (* program may change those parts during runtime. All pointers  *)
        (* to those structures are read only.                           *)

        ModuleDescrPtr  = POINTER TO ModuleDescr;
        AuxDescrPtr     = POINTER TO AuxDescr;

        ModuleDescr     = RECORD
          Next          :   ModuleDescrPtr;  (* next descr in chain     *)
          Frame         :   ADDRESS;         (* start of code segment   *)
          StaticBase    :   ADDRESS;         (* module's global data    *)
          ImportedMods  :   INTEGER;         (* length of imports list  *)
          ExportedProcs :   INTEGER;         (* length of exports list  *)
          CodeLength    :   INTEGER;         (* length of code area     *)
          DataLength    :   INTEGER;         (* length of variable area *)
          ConstLength   :   INTEGER;         (* length of constant area *)
          Thread        :   ThreadIds;       (* owning thread           *)
          References    :   SetOfThreadIds;  (*                         *)
          Reserved      :   ADDRESS          (*                         *)
                        END;

        AuxDescr        = RECORD
          Descr         :   ModuleDescrPtr;  (* back link to ModuleDescr*)
          Name          :   ModuleNames;     (*                         *)
```

```
        Key             :   ModuleKeys;         (*                          *)
        Flags           :   INTEGER;            (* initialisation status    *)
                        END;

TYPE    SearchResults   = (Done         ,       (* module was found         *)
                           NotFound     ,       (* module was not found     *)
                           IllegalKey   ,       (* found, but key mismatch *)
                           NotDone      );      (* not found, none of above*)


(*----- Category : Dynamic Linkable Module Chain (Primitives) ----------*)

PROCEDURE SearchModuleByName
                        (    aName      : ModuleNames;
                             aKey       : ModuleKeys;
                         VAR aDescr     : ModuleDescrPtr;
                         VAR aAuxDescr  : AuxDescrPtr)
                        : SearchResults;

(* Search the module chain for a module given by aName and aKey. Return
   a pointer to its ModuleDescr and to its AuxDescr if found. In that
   case answer Done, else answer the reason of failure, e.g. NotFound,
   IllegalKey or NotDone. *)


PROCEDURE SearchModuleByStaticBase
                        (    aStaticBase: ADDRESS;
                         VAR aDescr     : ModuleDescrPtr;
                         VAR aAuxDescr  : AuxDescrPtr)
                        : SearchResults;

(* Same as above, but the StaticBase, which is unique to each module is
   given. *)


PROCEDURE NextDescriptor
                        (VAR aDescr     : ModuleDescrPtr;
                         VAR aAuxDescr  : AuxDescrPtr);

(* Answer the descriptor behind aDescr within the module chain and its
   corresponding AuxDescr. If aDescr is NIL on entry, then the first
   module descriptor is answered. If aDescr is NIL upon return, the end
   of the modulelist has been reached. *)


(*----- Category : Threads ---------------------------------------------*)

        (* A thread is a list of modules with the main module beeing    *)
        (* the first module of the list. Within the list, all references*)
        (* to imported modules must be resolved and all necessary fixup *)
        (* must have been done. This is normally the job of the Loader  *)
        (* module.                                                      *)
        (* A thread is first linked to the currently known modules. It  *)
        (* may then be started to run until it completes or until it is *)
        (* killed due to some exceptional condition. It remains in      *)
        (* memory, until a module calls unlink and obtains the original *)
        (* module list.                                                 *)

PROCEDURE LinkThread    (    aDescr     : ModuleDescrPtr)
                        : BOOLEAN;

(* Link a thread, given by aDescr to the list of threads. All imports
   of the modules must have been resolved and all necessary fixup must
   have been done. The module given by aDescr is the main module of the
   Thread. *)
```

```
PROCEDURE RunThread     (    aDescr     : ModuleDescrPtr);

(* Run the thread, to which the module, given by aDescr, belongs. *)


PROCEDURE KillThread    (    aDescr     : ModuleDescrPtr);

(* Kill the thread, to which the module, given by aDescr, belongs.
   Furthermore, all threads that import modules of the terminating
   thread are killed as well. *)


PROCEDURE UnlinkThread  (VAR aDescr     : ModuleDescrPtr)
                        : BOOLEAN;

(* Unlink the thread, to which the module, given by aDescr, belongs,
   provided, the thread is not running and no other thread imports
   modules of that thread. *)


(*----- Category : Module Termination (Primitives) ---------------------*)

PROCEDURE OnModuleTerminationDo
                        (    aHandler   : PROC)
                        : BOOLEAN;

(* Request, that aHandler is called upon termination of the owning
   Thread. The termination handler announced last is called first upon
   thread termination. It is good practice, to  install termination
   handlers during module initialisation. Only 16 termination handlers
   can be installed for one single thread. Answer TRUE if successfull. *)


(*----- Category : Dynamic Procedure Chain -----------------------------*)

TYPE    (* A procedure invocation is described by a procedure frame,    *)
        (* which contains the current PC within the module, the module's*)
        (* static base value (see above) and a link to the calling      *)
        (* procedure's frame.                                           *)

        ProcedureFrames  = RECORD
          StaticBase     :   ADDRESS;
          DynamicBase    :   ADDRESS;
          RelativePC     :   LONGINT;
                           END;


(*----- Category : Dynamic Procedure Chain (Primitives) ----------------*)

PROCEDURE CallerOf      (VAR aFrame     : ProcedureFrames);

(* Using aFrame search the calling procedure frame and return it in
   aFrame. If aFrame is the main program upon entry, then return NIL in
   aFrame's StaticBase. *)


(*----- Category : Exceptions ------------------------------------------*)

TYPE    ExcTypes        = (ProgramHalt
                           InvalidCaseIndex
```

```
                              MissingResult             ,
                              CorruptedPointer          ,
                              DivideByZero              ,
                              RangeViolation            ,
                              ArithmOverflow            ,
                              StackOverflow             ,
                              Breakpoint                ,
                              ControlC                  ,
                              ProcessorDependent        ,
                              FPUnderflow               ,
                              FPOverflow                ,
                              FPDivideByZero            ,
                              FPNotANumber              ,
                              FPNotComparable           ,
                              FPInvalidDomain           ,
                              FPSingularity             ,
                              FPTotalLossOfSignificance ,
                              FPPartLossOfSignificance  ,
                              FPUnimplementedFunction   );


VAR     CurrExcType     : ExcTypes;
        CurrExcFrame    : ProcedureFrames;
        CurrExcRoot     : ProcedureFrames;


(*----- Category : Exceptions (Primitives) ----------------------------*)

PROCEDURE TextOfExc     (    anExc      : ExcTypes;
                         VAR aString    : ARRAY OF CHAR);

(* Answer a textual representation of anExc in aString (ca. 40 chars
   are required). *)


PROCEDURE OnExceptionDo (    aHandler   : PROC);

(* Request, that subsequent exceptions of the calling thread be handled
   by aHandler. *)


PROCEDURE SetDefaultExcHandler
                        (    aHandler   : PROC);

(* Request, that subsequently loadad threads use aHandler as their
   exception handler, provided, they do not announce their own handler. *)


PROCEDURE DefaultExcHandler;

(* Use some implementation dependent technique to display information
   about the current exception and the active procedure chain. *)


END System.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) TextFiles;
(* .      Function      : Virtual Text File Management                  *)
(* .      Version  Date : 19:40   5. 6.1988                             *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

A Textfile is a sequence of lines. The implementation of Textfile is
hidden from the calling modules. Lines are referenced via line numbers
or labels. The advantage of labels is, that they reference the same
line, even if lines are inserted before the labeled line (line numbers
change implicitly). A Textfile maintains a pointer to the current line.
the pointer can be moved to line numbers or labels. The content of the
current line can be read, referenced via a pointer and changed. New lines
are inserted behind the current line. *)


(*----- Category : Types and Data ----------------------------------------*)

TYPE    Results         = (Done, NotDone,
                           NoMemory, FileNotFound, IoError);

TYPE    rgText          = [0..255];
        Text            = ARRAY rgText OF CHAR;
        TextPtr         = POINTER TO Text;
        Labels          = [1..20];

        File;


(*----- Category : Initialisations ---------------------------------------*)

PROCEDURE Init;         (* no comment. *)


PROCEDURE Open          (VAR F          : File;
                             Name       : ARRAY OF CHAR;
                         VAR Result     : Results);

(* Open the named diskfile and read its content into the newly created
Textfile F. Answer Done=TRUE if successfull. *)


PROCEDURE Create        (VAR F          : File;
                         VAR Result     : Results);

(* Create a new Textfile. Return Done=TRUE if successfull. *)


PROCEDURE SaveAs        (    F          : File;
                             Name       : ARRAY OF CHAR;
                         VAR Result     : Results);

(* Save the Textfile F to disk under Name. Return Done=TRUE if
successfull. *)


PROCEDURE Abandon       (    F          : File);

(* Delete Textfile F. Do not save its content. *)
```

```
(*----- Category : Text Primitives -------------------------------------*)

PROCEDURE Position      (    F          : File;
                             Line       : CARDINAL);

(* Position the current line pointer of Textfile F to linenumber Line.
IF Line is < 1 then the first line becomes the current line, if Line
ist > total number of line, the last line becomes the current line. *)


PROCEDURE Replace       (    F          : File;
                          VAR NewLine    : ARRAY OF CHAR;
                          VAR Result     : Results);

(* Replace the content of the current line by NewLine. *)


PROCEDURE Insert        (    F          : File;
                          VAR NewLine    : ARRAY OF CHAR;
                          VAR Result     : Results);

(* Insert NewLine behind the current line. Please note, that linenumbers
of all subsequent lines are incremented. *)


PROCEDURE Delete        (    F          : File);

(* Delete the current line. Note, that linenumbers of all subsequent
lines are decremented. *)


(*----- Category : Labels ----------------------------------------------*)

PROCEDURE Label         (    F          : File;
                             Lbl        : Labels);

(* Set the label Lbl to the current position of file F. *)


PROCEDURE LineNumberOf  (    F          : File;
                             Lbl        : Labels)
                        : CARDINAL;

(* Position file F TO label Lbl. *)


(*----- Category : Inquiries -------------------------------------------*)

PROCEDURE ContentOf     (    F          : File;
                          VAR Content    : ARRAY OF CHAR);

(* Transfer the content of the current line into the buffer Content. *)


PROCEDURE LengthOf      (    F          : File)
                        : CARDINAL;

(* Answer the length of the current line. *)


PROCEDURE PositionOf    (    F          : File)
                        : CARDINAL;
```

```
(* Answer the linenumber of the current line. *)

PROCEDURE TotalLinesOf  (    F          : File)
                        : CARDINAL;

(* Answer the total number of lines of file F. *)

PROCEDURE PointerOf     (    F          : File)
                        : TextPtr;

(* Answer a TextPtr to the content of the current line. The TextPtr must
be used with extreme care. In particular is it not allowed to use the
term HIGH(tp). *)

END TextFiles.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name           : *) TextWindows;
(* .      Function       : Modula-2 Standard Window System              *)
(* .      Version/Date   : 1.1  1.8.87                                  *)
(* Product Name         : SSWiS                                        *)
(* Copyright            : (c) 1987,1988 MODsoft, D7500 Karlsruhe       *)


(*----- Category : Module Abstract ---------------------------------------**

TextWindows provides the windowing environment of SSWiS, but
redefines the term "World Coordinates". In the sense of a text window
the world consists of "Lines" and "Columns", where lines and columns
are numbered 0..N. Linenumbers grow from top to bottom (of the world),
columns grow from left to right. The position of the window, etc. can
be specified using lines and columns, as long as the window at hand
has been opened via TextWindows.Open (instead of SSWiS.CreateWindow).

The size of the world in terms of pixels depends on the character
font used. It is good programming practice, to isolate the application
modules from the details of character fonts by strictly relying on
the functionality provided by module TextWindows.

However it is possible, to manipulate the windows parameters by calling
the module SSWiS directly (passing the same handles). This is the
normal case for all screen coordinate-related functions, as well as
for the input event interface. Since the EventReport contains
world coordinates of module SSWiS, these coordinates must be
transformed to TextWindows.WorldCoordinates using the procedure
Identify.

Each text window maintains a socalled "Caret" at some position within
the world of lines and columns. The caret is used to specify the
position of textual output generated via the WriteXXXX procedures.
Whenever output has been done, the caret position moves to the end
of the current write operation. It is essential, that the WriteXXXX
procedures be only used by the restore callback procedures. The
caret position can be inquired and explicitly set.

The output procedures below are to be used by the restore procedure,
which, in turn, is triggered by textwindows. Thus, these procedures do
not need a window handle, etc.
*)


(*----- Category : Types and Data ----------------------------------------*)

IMPORT  SSWiS;

        (* The world of a textwindows consists of lines and columns.    *)
        (* The supported range is currently determined by 16bit value.  *)

TYPE    Coordinates     = INTEGER;
        FontSizes       = (Small, Normal, Large);

        Points          = RECORD
          X, Y          :   Coordinates;
                        END;


        RestoreProc     = PROCEDURE ((* Owner  *) SSWiS.ModuleHandles,
                                     (* Window *) SSWiS.WindowHandles,
                                     (* XY     *) Points,
```

```
                                    (* WH     *) Points);

        (* Callback procedure to restore the content of Window. The *)
        (* affected area of the world is given in WorldCoords.      *)


(*----- Category : Opening, Closing, Setup  ----------------------------*)

PROCEDURE Create        (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                            Restore     : RestoreProc);

(* Open a new text window Window. Note, that you MUST use this procedure
to open a TextWindow (you cannot use SSWiS.CreateWindow). *)


PROCEDURE Delete        (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles);

(* Close the TextWindow. *)


PROCEDURE ExplicitRestore
                        (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                            XY, WH      : Points);

(* Issue an explicit repaint operation of window u, but restrict
repaint to the given rectangle within the world coordinate space. *)


PROCEDURE AssignFont    (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                            FontSize    : FontSizes);

(* Assign non-proportional font for all output in window. *)


(*----- Category : Control scrolling (World Coordinates) ---------------*)

PROCEDURE PositionWorld (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                            XY          : Points);

(* Set the positon of the windows upper left corner in terms of
columns and lines. *)


PROCEDURE SizeWorld     (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                            WH          : Points);

(* Set the size of the application workspace in terms of
columns and lines. *)


PROCEDURE WorldOf       (   Owner       : SSWiS.ModuleHandles;
                            Window      : SSWiS.WindowHandles;
                        VAR XY, WH      : Points);

(* Answer the position and outline of the "world" in terms of columns
and lines. *)
```

```
PROCEDURE InteriorOf     (    Owner       : SSWiS.ModuleHandles;
                              Window      : SSWiS.WindowHandles;
                         VAR WH           : Points);

(* Answer the actual size of the window measured in lines and columns. *)


PROCEDURE Identify       (    Owner       : SSWiS.ModuleHandles;
                              Window      : SSWiS.WindowHandles;
                              RawXY       : SSWiS.Points;
                         VAR XY           : Points);

(* Answer the position in terms of column/line behind absolute position
x/y. *)


PROCEDURE SetCaret       (    Owner       : SSWiS.ModuleHandles;
                              Window      : SSWiS.WindowHandles;
                              XY          : Points);

(* Turn the caret OnNotOff and set it to position Col/Line. *)


(*----- Category : Streamlike Output (used by restore) -----------------*)

PROCEDURE Position       (    XY          : Points);

(* Set caret to Col/Line. *)


PROCEDURE PositionOf     (VAR XY          : Points);

(* Answer caret position. *)


PROCEDURE Invert         (    OnNotOff    : BOOLEAN);

(* Turn inverse writing mode OnNotOff. *)


PROCEDURE Write          (    ch          : CHAR);

(* Write character ch to window u at current position. *)


PROCEDURE WriteString    (    s           : ARRAY OF CHAR);

(* Write a string at caret position to window u. *)


PROCEDURE WriteLn;

(* End current line (line containing the caret) and skip to next
line on window u. *)


PROCEDURE Clear;

(* Clear until end of window. *)


PROCEDURE WriteLine      (VAR s           : ARRAY OF CHAR);

(* Write a string at caret position to window u. Same as WriteString
but improved speed due to VAR parameter. *)
```

```
END TextWindows.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) XStr;
(* .      Function      : Extended String Primitives                    *)
(* .      Version/Date  : 1.0  29.5.1987                                *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

XStr provides operations on strings beyound those of the standard module
Strings. In particular are conversions between non-string datatypes and
strings in either direction supported. XStr interfaces reference the
portion of interest by an index. The index is normally proceeded by the
XStr function itself. *)


(*----- Category : Types and Data ----------------------------------------*)

TYPE    TermProc        = PROCEDURE (CHAR) : BOOLEAN;

                            (* Answer FALSE, if the character meets some  *)
                            (* termination condition.                     *)

VAR     NumberBase      : CARDINAL;


(*----- Category : Copying and Scanning Strings -------------------------*)

PROCEDURE CopyForWhile  (    While      : TermProc;
                         VAR For        : CARDINAL;
                         VAR From       : ARRAY OF CHAR;
                         VAR FromInd    : CARDINAL;
                         VAR To         : ARRAY OF CHAR;
                         VAR ToInd      : CARDINAL);

(* Copy the String From starting at character FromInd to the string To
starting at character ToInd until For characters are copied or the
termination procedure While answers FALSE to the current character. *)


PROCEDURE ScanForWhile  (    While      : TermProc;
                         VAR For        : CARDINAL;
                         VAR From       : ARRAY OF CHAR;
                         VAR FromInd    : CARDINAL);

(* Scan the String From starting at character FromInd until For
characters are copied or the termination procedure While answers
FALSE to the current character. *)


(*----- Category : Default Termination Procs ----------------------------*)

PROCEDURE Forever                       (c : CHAR) : BOOLEAN;
PROCEDURE WhileInDigits                 (c : CHAR) : BOOLEAN;
PROCEDURE WhileInHexDigits              (c : CHAR) : BOOLEAN;
PROCEDURE WhileEqualBlank               (c : CHAR) : BOOLEAN;
PROCEDURE WhileInAlphas                 (c : CHAR) : BOOLEAN;
PROCEDURE WhileInAlphaNums              (c : CHAR) : BOOLEAN;
PROCEDURE WhileInPathChars              (c : CHAR) : BOOLEAN;


(*----- Category : Texts -------------------------------------------------*)
```

```
PROCEDURE Char          (    Ch          : CHAR;
                         VAR String      : ARRAY OF CHAR;
                         VAR Pos         : CARDINAL);

(* Put the character Ch into String at position Pos. Advance Pos. *)


(*----- Category : Numbers To Strings ---------------------------------*)

PROCEDURE Cardinal      (    Number      : CARDINAL;
                             Length      : CARDINAL;
                             Base        : CARDINAL;
                         VAR String      : ARRAY OF CHAR;
                         VAR Pos         : CARDINAL);

(* Convert Number to a textual representation of Length characters
(leading blanks are inserted if required) using Base as the number base
and put it into String at Pos. Advance Pos is sucessfull. *)


PROCEDURE Longcard      (    Number      : LONGCARD;
                             Length      : CARDINAL;
                             Base        : CARDINAL;
                         VAR String      : ARRAY OF CHAR;
                         VAR Pos         : CARDINAL);

(* See above. *)


PROCEDURE Integer       (    Number      : INTEGER;
                             Length      : CARDINAL;
                             Base        : CARDINAL;
                         VAR String      : ARRAY OF CHAR;
                         VAR Pos         : CARDINAL);

(* See above. *)


PROCEDURE Longint       (    Number      : LONGINT;
                             Length      : CARDINAL;
                             Base        : CARDINAL;
                         VAR String      : ARRAY OF CHAR;
                         VAR Pos         : CARDINAL);

(* See above. *)


PROCEDURE Real          (    Number      : REAL;
                             Length      : INTEGER;
                             FracLength  : INTEGER;
                         VAR String      : ARRAY OF CHAR;
                         VAR Pos         : CARDINAL);

(* Convert Number to a textual representation of Length characters
(leading blanks are inserted if required) in total and FracLength
characters behind the decimal point. Put it into String at Pos and
advance Pos is sucessfull. If FracLength is less than 0, then
scientific notation is used. If Number is not a valid real number,
then the string NAN is inserted into String, if the number does not
fit into Length, then Length asterisks are inserted. *)


PROCEDURE Longreal      (    Number      : LONGREAL;
                             Length      : INTEGER;
```

```
                            FracLength : INTEGER;
                           VAR String  : ARRAY OF CHAR;
                           VAR Pos     : CARDINAL);

(* see above. *)


PROCEDURE InvCardinal    (VAR Number   : CARDINAL;
                          VAR String   : ARRAY OF CHAR;
                          VAR Pos      : CARDINAL)
                         : BOOLEAN;

(* Parse String, starting for a position Pos, for a CARDINAL number. If
successfull then answer TRUE and advance Pos.
The number syntax is : digit{digit}.    *)


PROCEDURE InvLongcard    (VAR Number   : LONGCARD;
                          VAR String   : ARRAY OF CHAR;
                          VAR Pos      : CARDINAL)
                         : BOOLEAN;

(* See above. *)


PROCEDURE InvInteger     (VAR Number   : INTEGER;
                          VAR String   : ARRAY OF CHAR;
                          VAR Pos      : CARDINAL)
                         : BOOLEAN;

(* Parse String, starting for a position Pos, for an INTEGER number. If
successfull then answer TRUE and advance Pos.
The number syntax is : [+|-]digit{digit}.    *)


PROCEDURE InvLongint     (VAR Number   : LONGINT;
                          VAR String   : ARRAY OF CHAR;
                          VAR Pos      : CARDINAL)
                         : BOOLEAN;

(* See above. *)


PROCEDURE InvReal        (VAR Number   : REAL;
                          VAR String   : ARRAY OF CHAR;
                          VAR Pos      : CARDINAL)
                         : BOOLEAN;

(* Parse String, starting for a position Pos, for a REAL number. If
successfull then answer TRUE and advance Pos. Real syntax is :

   realnumber       = fixedpointnumber [exponent].
   fixedpointnumber = [sign] {digit} [ '.' {digit} ].
   exponent         =  ('e' | 'E') [sign] digit {digit}.
   sign             = '+' | '-'.
   digit            = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'.

The following numbers are legal representations of one
hundred: 100, 10E1, 100E0, 1000E-1, E2, +E2, 1E2, +1E2,
+1E+2, 1E+2 .*)


PROCEDURE InvLongreal    (VAR Number   : LONGREAL;
                          VAR String   : ARRAY OF CHAR;
```

```
                        VAR Pos         : CARDINAL)
                       : BOOLEAN;

(* See above. *)

END XStr.
```

Die SYSLIB

Anhang H

Diese Seite wurde aus
satztechnischen Gründen frei
gelassen

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESApplications;
(* .      Function      : Interface to GEM AES                          *)
(* .      Version/Date  : 1.0  1.7.1987                                 *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM SYSTEM IMPORT ADDRESS;


(*----- Category : Primitives ------------------------------------------*)

PROCEDURE Initialise    () : INTEGER;

(* Initialise application. Returns ApId *)


PROCEDURE Read          (   Id, Length  : INTEGER;
                            PBuff       : ADDRESS);

(* Read from a message pipe. Returns coded return message, 0 = error *)


PROCEDURE Write         (   Id, Length  : INTEGER;
                            PBuff       : ADDRESS);

(* Write to a message pipe. Returns coded return message, 0 = error *)


PROCEDURE Find          (VAR FPname     : ARRAY OF CHAR)
                        : INTEGER;

(* Find the Id of another application in the system. *)


PROCEDURE TPlayback     (   TpMem       : ADDRESS;
                            TpNum       ,
                            TpScale     : INTEGER);

(* Play back a piece of GEM AES recording of user's actions. *)


PROCEDURE TRecord       (   TrMem       : ADDRESS;
                            TrNum       : INTEGER);

(* Record the next ApTrNum user actions. Returns number recorded. *)


PROCEDURE Exit;
```

```
(* Exit application. *)

END AESApplications.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESBase;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                              *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe          *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

CONST   White           =  0;
        Black           =  1 ;
        Red             =  2;
        Green           =  3;
        Blue            =  4;
        Cyan            =  5;
        Yellow          =  6;
        Magenta         =  7;
        LightWhite      =  8;
        LightBlack      =  9;
        LightRed        = 10;
        LightGreen      = 11;
        LightBlue       = 12;
        LightCyan       = 13;
        LightYellow     = 14;
        LightMagenta    = 15;

TYPE    AESGlobalType   = RECORD
          apVersion     :   CARDINAL;
          apCount       :   CARDINAL;
          apID          :   CARDINAL;
          apPrivate     :   LONGCARD;
          apPTree       :   ADDRESS;
          ap1Resv       :   LONGCARD;
          ap2Resv       :   LONGCARD;
          ap3Resv       :   LONGCARD;
          ap4Resv       :   LONGCARD;
                        END;

        AESControlType  = RECORD
          opcode        :   CARDINAL;
          sizeIntIn     :   CARDINAL;
          sizeIntOut    :   CARDINAL;
          sizeAddrIn    :   CARDINAL;
          sizeAddrOut   :   CARDINAL;
                        END;

        AESIntInType    = ARRAY [0..16] OF INTEGER;
        AESIntOutType   = ARRAY [0..7] OF INTEGER;
        AESAddrInType   = ARRAY [0..2] OF ADDRESS;
        AESAddrOutType  = ARRAY [0..1] OF ADDRESS;
```

```
        AESParameterType= RECORD
          control       :   POINTER TO AESControlType;
          global        :   POINTER TO AESGlobalType;
          intIn         :   POINTER TO AESIntInType;
          intOut        :   POINTER TO AESIntOutType;
          addrIn        :   POINTER TO AESAddrInType;
          addrOut       :   POINTER TO AESAddrOutType;
                        END ;

VAR     AESParameters   : AESParameterType ;
        AESGlobal       : AESGlobalType ;
        AESControl      : AESControlType ;
        AESIntIn        : AESIntInType ;
        AESIntOut       : AESIntOutType ;
        AESAddrIn       : AESAddrInType ;
        AESAddrOut      : AESAddrOutType ;

        CallResult      : INTEGER;


PROCEDURE GEMCall       (   Opcode      : INTEGER;
                            Cntrl1      ,
                            Cntrl2      ,
                            Cntrl3      ,
                            Cntrl4      : INTEGER);

END AESBase.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name           : *) AESEvents;
(* .      Function       : Interface to GEM AES                          *)
(* .      Version/Date   : 1.0  1.7.1987                                 *)
(* Product Name          : SPC                                           *)
(* Copyright             : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)

(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  WORD;

TYPE    Events          = (KeyboardEvent,
                           ButtonEvent,
                           Mouse1Event,
                           Mouse2Event,
                           MessageEvent,
                           TimerEvent);

        SetOfEvents     = SET OF Events;

        KbdStates       = (RightShift,
                           LeftShift,
                           CTRL,
                           ALT);

        SetOfKbdStates  = SET OF KbdStates;

CONST   (* Message values *)
        MenuSelected    = 10;    (* Menu item was selected *)
        WindowRedraw    = 20;    (* Window needs redrawing *)
        WindowTopped    = 21;    (* A window was moved to the top *)
        WindowClosed    = 22;    (* Window was closed *)
        WindowFulled    = 23;    (* Wndow was fulled *)
        WindowArrowed   = 24;    (* Window was arrowed *)
        WindowHorizSlided = 25;  (* Horizontal slider was moved *)
        WindowVertSlided  = 26;  (* Vertical slider was moved *)
        WindowSized     = 27;    (* Window was sized *)
        WindowMoved     = 28;    (* Window was moved *)
        WindowNewTop    = 29;    (* Window was moved to top (activated) *)
        AccessoryOpen   = 40;    (* Accessory requested to open *)
        AccessoryClose  = 41;    (* Accessory requested to close *)


(*----- Category : Primitives ------------------------------------------*)

PROCEDURE Keyboard      () : INTEGER;

(* Wait for keyboard input. *)


PROCEDURE Button        (    Clicks     : INTEGER;
                             RequMask   ,
```

```
                            RequState  : BITSET;
                        VAR x, y       : INTEGER;
                        VAR ButState   : BITSET;
                        VAR KbdState   : SetOfKbdStates)
                       : INTEGER;

(* Wait for a mouse action. *)


PROCEDURE Mouse         (   RetOnExit  : BOOLEAN;
                            x, y, w, h : INTEGER;
                        VAR mX, mY     : INTEGER;
                        VAR ButState   : BITSET;
                        VAR KbdState   : SetOfKbdStates);

(* Wait for mouse to enter or leave a specified rectangle. *)


PROCEDURE Message       (VAR MsgBuffer  : ARRAY OF WORD);

(* Wait for 16 bit message from the message pipe. *)


PROCEDURE Timer         (   Millisecs  : LONGCARD);

(* Wait for time to pass *)


PROCEDURE Multiple      (   EventMask  : SetOfEvents;
                            RequClicks : INTEGER;
                            RequButtons: BITSET;
                            RequState  : BITSET;
                            RetOnExit1 : BOOLEAN;
                            x1, y1     ,
                            w1, h1     : INTEGER;
                            RetOnExit2 : BOOLEAN;
                            x2, y2     ,
                            w2, h2     : INTEGER;
                        VAR MsgBuffer  : ARRAY OF INTEGER;
                            Millisecs  : LONGCARD;
                        VAR mX, mY     : INTEGER;
                        VAR ButState   : BITSET;
                        VAR KbdState   : SetOfKbdStates;
                        VAR KeyCode    : INTEGER;
                        VAR Clicks     : INTEGER)
                       : SetOfEvents;

(* Wait for multiple events. *)


PROCEDURE DoubleClick   (   NewSpeed   : INTEGER;
                            SetNotGet  : BOOLEAN)
                       : INTEGER ;

(* Set or Get double click speed. *)


END AESEvents.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESForms;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                                *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987            *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;
IMPORT  AESObjects;

TYPE    Phases          = (Start, Grow, Shrink, Finish);


(*----- Category : Primitives --------------------------------------------*)

PROCEDURE Do            (    Tree       : AESObjects.TreePtr;
                             StartObj   : INTEGER)
                        : INTEGER ;

(* Causes the form library to monitor a users interaction with a form *)


PROCEDURE Dialogue      (    Flag       : Phases;
                             LitX, LitY ,
                             LitW, LitH ,
                             BigX, BigY ,
                             BigW, BigH : INTEGER);

(* Multi forms action according to flag *)


PROCEDURE Alert         (    DefButton  : INTEGER;
                         VAR String : ARRAY OF CHAR)
                        : INTEGER ;

(* Displays an alert *)


PROCEDURE Error         (    ErrorNum   : INTEGER)
                        : INTEGER ;

(* Displays an error *)


PROCEDURE Center        (    Tree       : AESObjects.TreePtr;
                         VAR X, Y, W, H : INTEGER);

(* Centers a dialog box on the screen *)
```

```
PROCEDURE FileSelectorInput
                        (VAR Path         ,
                             Selection    : ARRAY OF CHAR;
                         VAR OkNotCancel: BOOLEAN);


END AESForms.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESGraphics;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                                *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987            *)


(*----- Category : Types and Data ----------------------------------------*)

IMPORT  AESEvents, AESObjects;

CONST   Arrow           = 0;
        TextCursor      = 1;
        HourGlass       = 2;
        PointHand       = 3;
        FlatHand        = 4;
        ThinCross       = 5;
        ThickCross      = 6;
        OutlineCross    = 7;
        UserDef         = 255;
        MouseOff        = 256;
        MouseOn         = 257;

TYPE    Cursors         = [Arrow..MouseOn];


(*----- Category : Primitives --------------------------------------------*)

PROCEDURE RubberBox     (    X, Y, W, H : INTEGER;
                         VAR LastW      ,
                             LastH      : INTEGER);

(* Draws a "rubber box" *)


PROCEDURE DragBox       (    X, Y, W, H : INTEGER;
                             BoundX     ,
                             BoundY     ,
                             BoundW     ,
                             BoundH     : INTEGER;
                         VAR LastX      ,
                             LastY      : INTEGER);

(* Allow user to drag a box *)


PROCEDURE MoveBox       (    X, Y, W, H : INTEGER;
                             DestX      ,
                             DestY      : INTEGER);

(* Draws a moving box *)
```

```
PROCEDURE GrowBox       (    StX, StY   ,
                             StW, StH   : INTEGER;
                             FinX, FinY ,
                             FinW, FinH : INTEGER);

(* Draws an expanding box outline *)


PROCEDURE ShrinkBox     (    FinX, FinY ,
                             FinW, FinH : INTEGER;
                             StX, StY   ,
                             StW, StH   : INTEGER);

(* Draws an shrinking box outline *)


PROCEDURE WatchBox      (    Tree       : AESObjects.TreePtr;
                             Object     : INTEGER;
                             InState    ,
                             OutState   : AESObjects.SetOfStates)
                        : BOOLEAN;

(* Tracks mouse in and out of box *)


PROCEDURE SlideBox      (    Tree       : AESObjects.TreePtr;
                             Parent     ,
                             Obj        : INTEGER;
                             VertNotHor : BOOLEAN)
                        : INTEGER ;

(* Tracks sliding box in a parent box *)


PROCEDURE Handle        (VAR WChr, HChr ,
                             WBox, HBox : INTEGER)
                        : INTEGER ;

(* Gets the GEM VDI handle *)


PROCEDURE Mouse         (    Form       : Cursors;
                             UserDef    : AESObjects.AnyBitmapPtr);


PROCEDURE MouseKeyboardState
                        (VAR X, Y       : INTEGER;
                         VAR ButtonState: BITSET;
                         VAR KbdState   : AESEvents.SetOfKbdStates);

(* Return mouse loc and state *)


END AESGraphics.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESMenus;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                              *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe          *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

IMPORT  AESObjects;


(*----- Category : Primitives ------------------------------------------*)

PROCEDURE Bar           (    Tree       : AESObjects.TreePtr;
                             ShowNotHide: BOOLEAN);

(* Display or erase current menu bar *)


PROCEDURE ItemCheck     (    Tree       : AESObjects.TreePtr;
                             Item       : INTEGER;
                             SetNotRemv : BOOLEAN);

(* Display or erase a check mark next to a menu item *)


PROCEDURE ItemEnable    (    Tree       : AESObjects.TreePtr;
                             Item       : INTEGER;
                             Enable     : BOOLEAN);

(* Enables or disables a menu item *)


PROCEDURE TitleNormal   (    Tree       : AESObjects.TreePtr;
                             Title      : INTEGER;
                             Normal     : BOOLEAN);

(* Displays a menu title in normal or reverse video *)


PROCEDURE Text          (    Tree       : AESObjects.TreePtr;
                             Item       : INTEGER;
                             String     : ARRAY OF CHAR);

(* Changes the text of a menu item *)


PROCEDURE Register      (    Apid       : INTEGER;
                             String     : ARRAY OF CHAR)
                        : INTEGER;
```

```
(* Placé desk accessorie's text in a menu *)

END AESMenus.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESObjects;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                              *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe          *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

TYPE    Types           = (t0, t1, t2, t3, t4, t5, t6, t7, t8, t9,
                           u0, u1, u2, u3, u4, u5, u6, u7, u8, u9,
                           Box,
                           Text,
                           BoxText,
                           Image,
                           ProgDef,
                           InvisibleBox,
                           Button,
                           BoxChar,
                           String,
                           FormattedText,
                           FormattedBoxText,
                           Icon,
                           Title);

        Flags           = (Selectable,
                           Default,
                           Exit,
                           Editable,
                           RadioButton,
                           LastObject,
                           TouchExit,
                           HideTree,
                           Indirect);

        SetOfFlags      = SET OF Flags;

        States          = (Selected,
                           Crossed,
                           Checked,
                           Disabled,
                           Outlined,
                           Shadowed);

        SetOfStates     = SET OF States;

        Edits           = (InitText, AcceptChar, Terminate);
        Justifications  = (Left, Right, Centered);
        AnyText         = ARRAY [0..999] OF CHAR;
        AnyBitmap       = ARRAY [0..999] OF BITSET;
```

```
        AnyTextPtr      = POINTER TO AnyText;
        AnyBitmapPtr    = POINTER TO AnyBitmap;
        TedInfoPtr      = POINTER TO TedInfo;
        IconBlkPtr      = POINTER TO IconBlk;
        BitBlkPtr       = POINTER TO BitBlk;
        ApplBlkPtr      = POINTER TO ApplBlk;
        ParamBlkPtr     = POINTER TO ParamBlk;

        Object          = RECORD
          Next          :   INTEGER;
          Head, Tail    :   INTEGER;
          Reserved      :   CHAR;
          Type          :   Types;
          Flag          :   SetOfFlags;
          State         :   SetOfStates;
          CASE          :   Types OF
            Box, BoxChar,
            InvisibleBox    : Colour   : INTEGER;
(*                            Border   : [-127..127]; *)
                              Char     : CHAR;
          | Text, BoxText   ,
            String, Title   : TxtP     : AnyTextPtr;
          | Image           : BitBlkP  : BitBlkPtr;
          | ProgDef         : ApplBlkP : ApplBlkPtr;
          | FormattedText,
            FormattedBoxText,
            Button          : TedInfoP : TedInfoPtr;
          | Icon            : IconBlkP : IconBlkPtr;
          END;
          x, y, w, h    :   INTEGER;
                        END;

        ObjectTree      = ARRAY [0..999] OF Object;
        TreePtr         = POINTER TO ObjectTree;

        TedInfo         = RECORD
          Text          ,
          Template      ,
          Valid         :   AnyTextPtr;
          Font          :   INTEGER;
          Reserved1     :   INTEGER;
          Justif        :   Justifications;
          Colour        :   INTEGER;
          Reserved2     :   INTEGER;
          Thickness     :   INTEGER;
          TextLen       :   INTEGER;
          TemplateLen   :   INTEGER;
                        END;

        IconBlk         = RECORD
          Mask          ,
          Data          :   AnyBitmapPtr;
          Text          :   AnyTextPtr;
          Reserved      :   CHAR;
          Char          :   CHAR;
          xChar, yChar  :   INTEGER;
          x, y, w, h    :   INTEGER;
          xText, yText  ,
          wText, hText  :   INTEGER;
                        END;

        BitBlk          = RECORD
          Data          :   AnyBitmapPtr;
          w, h, x, y    :   INTEGER;
```

```
    Colour          :   INTEGER;
                    END;

  ParamBlk          = RECORD
    Tree            :   TreePtr;
    Obj             :   INTEGER;
    PreviousState   :   SetOfStates;
    CurrentState    :   SetOfStates;
    x, y, w, h      :   INTEGER;
    xc, yc, wc, hc  :   INTEGER;
                    END;

  DrawProc          = PROCEDURE (VAR ParamBlk);

  ApplBlk           = RECORD
    Code            :   DrawProc;
    Param           :   ParamBlkPtr;
                    END;


(*----- Category : Primitives ----------------------------------------*)

PROCEDURE Add           (   Tree        : TreePtr;
                            Parent      ,
                            Child       : INTEGER);

(* Adds and object to the object tree *)


PROCEDURE Delete        (   Tree        : TreePtr;
                            Obj         : INTEGER);

(* Delete an object from an object tree *)


PROCEDURE Draw          (   Tree        : TreePtr;
                            StartObj    ,
                            Depth       ,
                            x, y, w, h  : INTEGER);

(* Draws any object(s) in the object tree *)


PROCEDURE Find          (   Tree        : TreePtr;
                            StartObj    ,
                            Depth       : INTEGER;
                            x, y        : INTEGER)
                        : INTEGER ;

(* Finds an object under the mouse form *)


PROCEDURE Offset        (   Tree        : TreePtr;
                            Obj         : INTEGER;
                         VAR x, y       : INTEGER);

(* Computes an objects X and Y coords relative to the screen *)


PROCEDURE Order         (   Tree        : TreePtr;
                            Obj         ,
                            NewPos      : INTEGER);

(* Moves an object within its parents list *)
```

```
PROCEDURE Edit          (    Tree       : TreePtr;
                             Obj        : INTEGER;
                             Char       : CHAR;
                             IdX        : INTEGER;
                             Kind       : Edits;
                         VAR NewIdX     : INTEGER);

(* Allow user to edit text in an object *)


PROCEDURE Change        (    Tree       : TreePtr;
                             Obj        ,
                             x, y, w, h : INTEGER;
                             NewState   : SetOfStates;
                             Redraw     : BOOLEAN);

(* Changes an objects State value *)


END AESObjects.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESResources;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                              *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe          *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;
IMPORT  AESObjects;

TYPE    ItemTypes       = (Tree, Object, TedInfo, IconBlock,
                           BitBlock, String, ImageData, ObSpec,
                           Text, Template, Valid, IconBlkMask,
                           IconBlkData, IconBlkText, BitBlkData,
                           FreeString, FreeImage);


(*----- Category : Primitives ------------------------------------------*)

PROCEDURE Load          (    FName      : ARRAY OF CHAR);

(* Load a resource file *)


PROCEDURE Free;

(* Free loaded space *)


PROCEDURE GetAddr       (    Type       : ItemTypes;
                             Index      : INTEGER;
                         VAR Addr       : ADDRESS);

(* Get address of resource *)


PROCEDURE SetAddr       (    Type       : ItemTypes;
                             Index      : INTEGER;
                             Addr       : ADDRESS);

(* Set address of resource *)


PROCEDURE ObjectFix     (    Tree       : AESObjects.TreePtr;
                             Obj        : INTEGER);

(* Convert object's character x,y to pixel x,y *)


END AESResources.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESScraps;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                                *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;


(*----- Category : Primitives ------------------------------------------*)

PROCEDURE Read          (    Scrap      : ADDRESS);

(* Reads the current scrap directory *)


PROCEDURE Write         (    Scrap      : ADDRESS);

(* Writes the current scrap directory *)


END AESScraps.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESShells;
(* .      Function      : *)
(* .      Version/Date  : 1.0 / 02.07.1987                                *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract ---------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;


(*----- Category : Primitives --------------------------------------------*)

PROCEDURE Read          (VAR Command    ,
                             Tail       : ARRAY OF CHAR);


PROCEDURE Write         (    ChainNotXit: BOOLEAN;
                             GrafAppl   : BOOLEAN;
                             GEMAppl    : BOOLEAN;
                             Command    ,
                             Tail       : ARRAY OF CHAR);


PROCEDURE Find          (VAR Buffer     : ARRAY OF CHAR);


PROCEDURE Envrn         (VAR Value      ,
                             Param      : ARRAY OF CHAR);


END AESShells.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AESWindows;
(* .      Function      : Interface to GEM AES                          *)
(* .      Version/Date  : 1.0  1.7.1987                                 *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM AES. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

TYPE    Elements        = (NameLine    , Closer      , Fuller      ,
                           Mover       , InfoLine    , Sizer       ,
                           UpArrow     , DownArrow   , VertSlider  ,
                           LeftArrow   , RightArrow  , HorizSlider );

        SetOfElements   = SET OF Elements;

TYPE    Items           = (Illegal,
                           Kind,
                           Name,                Info,
                           WorkXYWH,            CurrXYWH,
                           PrevXYWH,            FullXYWH,
                           HorizSliderPos,      VertSliderPos,
                           Top,
                           FirstXYWH,           NextXYWH,
                           Reserved,            NewDesk,
                           HorizSliderSize,     VertSliderSize,
                           WindowScreen);

        ItemStruc       = RECORD
          CASE Field    : Items OF
            Kind:                   Elems  : SetOfElements;
          | Name, Info:             Str    : ADDRESS;
          | NewDesk:                Tree   : ADDRESS;
                                    Obj    : INTEGER;
          | WorkXYWH , CurrXYWH,
            PrevXYWH , FullXYWH,
            FirstXYWH, NextXYWH:    x,y,w,h: INTEGER;
          | HorizSliderPos,
            VertSliderPos:          Pos    : INTEGER;
          | HorizSliderSize,
            VertSliderSize:         Size   : INTEGER;
          | Top:                    Wind   : INTEGER;
          | ELSE                    Nothing: CARDINAL;
          END;
                        END;

TYPE    WindowAreas     = (WindowOutline, WindowInterior);

        UpdateFlags     = (EndUpdate,
                           BeginUpdate,
```

```
                        EndMouseControl,
                        BeginMouseControl);


(*----- Category : Primitives ----------------------------------------*)

PROCEDURE Create        (    Kind       : SetOfElements;
                             X, Y, W, H : INTEGER)
                        : INTEGER ;

(* Create a new window with specified elements and maximum size of
X/Y+W/H. Return the AES window handle if successfull. *)


PROCEDURE Open          (    Handle     : INTEGER;
                             X, Y, W, H : INTEGER);

(* Open the window with an outline of X/Y+W/H. *)


PROCEDURE Close         (    Handle     : INTEGER);

(* Close the window. *)


PROCEDURE Delete        (    Handle     : INTEGER);

(* Delete the window and free space. *)


PROCEDURE Get           (    Handle     : INTEGER;
                         VAR Item       : ItemStruc);

(* Answers info of a window further specified by Item.Field in Item. *)


PROCEDURE Set           (    Handle     : INTEGER;
                             Item       : ItemStruc);

(* Sets info of a window further specified by Item*)


PROCEDURE Find          (    mX, mY     : INTEGER)
                        : INTEGER ;

(* Find window under position mX/mY. *)


PROCEDURE Update        (    BegEnd     : UpdateFlags);

(* Update window *)


PROCEDURE Calc          (    Type       : WindowAreas;
                             Kind       : SetOfElements;
                             InX, InY   ,
                             InW, InH   : INTEGER ;
                         VAR OutX, OutY ,
                             OutW, OutH : INTEGER );

(* Calc window outline (Type = WindowOutline) from interior or
interior (Type = WindowInterior) from outline. *)
```

```
END AESWindows.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) AltResource;
(* .      Function      : Load and Fixup AES Resource Files            *)
(* .      Version/Date  : 1.0 / 02.07.1987                             *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe           *)


(*----- Category : Module Abstract ---------------------------------------**

  Since MODULA-2 applications are highly modular, it is necessary to load
  several RSC-Files, depending on the modules loaded. AltResource loads
  RSC-Files into dynamic memory and does the necessary fixup. *)

(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;
IMPORT  AESResources;
TYPE    DataPtr;


(*----- Category : Loading and Deleting Resource ------------------------*)

PROCEDURE Load          (VAR Data       : DataPtr;
                             Name       : ARRAY OF CHAR;
                         VAR Done       : BOOLEAN);

(* Load a named RSC-File into a piece of dynamic memory and return its
   pointer if successfull. This corresponds to AESResources.Load with
   the exception, that AltResource may load more than one file. *)


PROCEDURE Free          (VAR Data       : DataPtr);

(* Free a previously loaded RSC-File. *)


(*----- Category : Obtaining Addresses ----------------------------------*)

PROCEDURE GetAddr       (    Data       : DataPtr;
                             Type       : AESResources.ItemTypes;
                             Index      : CARDINAL;
                         VAR Addr       : ADDRESS);

(* Get the address of an indexed tree within the RSC-File. *)


END AltResource.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Bios;
(* .      Function      : BIOS Interface to Modula-2                    *)
(* .      Version/Date  : 1.0 / 8.9.88                                  *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1988, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to BIOS. All functions
are explained in various documents about GEM on the ATARI ST computer.  *)


(*----- Category : Types and DATA --------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

TYPE    ScanCodes       = [0..255];

        Devices         = ( Printer,
                            Aux,
                            Console,
                            Midi,
                            Keyboard );

        BlockModes      = ( Read,
                            Write,
                            ReadNoMediaChange,
                            WriteNoMediaChange );

        BiosParmBlock   = RECORD
           BytesPerSec  :    INTEGER;
           SecPerClust  :    INTEGER;
           BytesPerClust:    INTEGER;
           DirLength    :    INTEGER;
           FATLength    :    INTEGER;
           FAT2Start    :    INTEGER;
           FirstFreeSec :    INTEGER;
           NumOfClust   :    INTEGER;
           Flags        :    ARRAY [1..8] OF INTEGER;
                         END;
        BiosParmPtr     = POINTER TO BiosParmBlock;

        Drives          = [0..15];
        SetOfDrives     = SET OF Drives;

        MemDefPtr       = POINTER TO MemDefBlock;
        MemParmBlock    = RECORD
           FreeList     :    MemDefPtr;
           AllocList    :    MemDefPtr;
           RovingPtr    :    MemDefPtr;
                         END;
        MemDefBlock     = RECORD
           Next         :    MemDefPtr;
           Start        :    ADDRESS;
           Length       :    LONGINT;
           OwnerProc    :    MemDefPtr;
                         END;

        MediaStat       = ( MediaChanged,
                            MediaMightChanged,
                            MediaNotChanged );
```

```
VAR     Result          : LONGINT;

(*----- Category : Bios Functions -----------------------------------*)

PROCEDURE Getmpb        (VAR mpb          : MemParmBlock );

PROCEDURE Bconstat      (    Dev          : Devices )
                        : BOOLEAN;

PROCEDURE Bconin        (    Dev          : Devices;
                         VAR Ch           : CHAR;
                         VAR ScanCode     : ScanCodes );

PROCEDURE Bconout       (    Dev          : Devices;
                             C            : CHAR );

PROCEDURE Rwabs         (    Flag         : BlockModes;
                             Buffer       : ADDRESS;
                             Count        : CARDINAL;
                             Sector       : CARDINAL;
                             Dev          : Drives );

PROCEDURE SetException  (    Nr           . INTEGER;
                             Vec          : ADDRESS )
                        : ADDRESS;

PROCEDURE TickCal       () : LONGINT;

PROCEDURE Getbpb        (    Dev          : INTEGER )
                        : BiosParmPtr;

PROCEDURE Bcostat       (    Dev          : Devices )
                        : BOOLEAN;

PROCEDURE MediaChange   (    Dev          : INTEGER )
                        : MediaStat;

PROCEDURE DriveMap      () : SetOfDrives;

PROCEDURE KeyShifts     (    Mode         : INTEGER )
                        : LONGINT;

END Bios.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Files;
(* .      Function      : Basic File Services                          *)
(* .      Version/Date  : 1.1  21.1.88                                 *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)

(*----- Category : Module Abstract -------------------------------------**

  Files pprovides the basic interface to the operating system dependent
  file system (not to confuse with module FileSystem). The interface to
  Files is slightly system dependent. So applications should use disk
  files via FileSystem, ByteStreams or TextStreams. *)


(*----- Category : Types and Data --------------------------------------*)

FROM    SYSTEM  IMPORT  WORD, BYTE, ADDRESS;
IMPORT  Clock;

TYPE    Results         = (Done, NotDone);
        Types           = (Text, Data, Code, DontCare);

TYPE    File            = RECORD
          Type          :   Types;
          Handle        :   INTEGER;
                        END;


(*----- Category : Opening, Closing, Renaming, Deleting ----------------*)

PROCEDURE Lookup        (VAR F          : File;
                             FileName   : ARRAY OF CHAR;
                             FileType   : Types;
                         VAR Result     : Results);

(* Lookup the named file and open it if successfull. Type is set to
   the apprpriate value, if type is of signifcance for the caller. *)


PROCEDURE Create        (VAR F          : File;
                             FileName   : ARRAY OF CHAR;
                             FileType   : Types;
                         VAR Result     : Results);

(* Create a named file of a given type. If the file already exists,
   the delete the old one first. *)


PROCEDURE Close         (VAR F          : File;
                         VAR Result     : Results);

(* Close the file and disconnect the file variable from it. *)


PROCEDURE Delete        (    FileName   : ARRAY OF CHAR;
                         VAR Result     : Results);

(* Delete the named file. *)


PROCEDURE Rename        (    FileName   : ARRAY OF CHAR;
                             NewName    : ARRAY OF CHAR;
```

```
                          VAR Result     : Results);

(* Rename the named file to NewName. *)

(*----- Category : Position and Size ---------------------------------*)

PROCEDURE SetPos         (VAR F          : File;
                              Pos        : LONGINT;
                          VAR Result     : Results);

(* Set the position of the filepointer to Pos. The units of Pos are
   system dependent. *)


PROCEDURE GetPos         (VAR F          : File;
                          VAR Pos        : LONGINT;
                          VAR Result     : Results);

(* Get the poition of the filepointer. The units of Pos are system
   dependent. *)


PROCEDURE Length         (VAR F          : File;
                          VAR Len        : LONGINT;
                          VAR Result     : Results);

(* Get the Length of the file. The units are system dependent. *)


PROCEDURE Timestamp      (VAR F          : File;
                          VAR Stamp      : Clock.Time);

(* Answer the creation/modification timestamp of file F. *)


(*----- Category : Basic Input/Output --------------------------------*)

PROCEDURE ReadBlock      (VAR F          : File;
                              BlockPtr   : ADDRESS;
                              Bytes      : LONGINT;
                          VAR BytesRead  : LONGINT;
                          VAR Result     : Results);

(* Read a block of Bytes into BlockPtr. Answer the number of BytesRead. *)


PROCEDURE WriteBlock     (VAR F          : File;
                              BlockPtr   : ADDRESS;
                              Bytes      : LONGINT;
                          VAR BytesWritt : LONGINT;
                          VAR Result     : Results);

(* Write the block given via BlockPtr of Bytes length into file F and
   answer the Number of BytesWritten. *)


END Files.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) GemDos;
(* .      Function      : GEMDOS Interface to Modula-2                  *)
(* .      Version  Date :  8:10  16.11.1988                             *)
(* Product Name         : SPC                                           *)
(* Copyright            : (c) 1987, MODsoft, D7500 Karlsruhe            *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM DOS. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

IMPORT  SYSTEM;

CONST   FolderSep       = '\';
        VolumeSep       = ':';
        TypeSep         = '.';
        NameLength      = 8;
        TypeLength      = 3;

CONST   StdIn           = 0;
        StdOut          = 1;
        Serial          = 2;
        Parallel        = 3;

TYPE    ScanCodes       = [0..255];

        Drives          = [0..15];
        SetOfDrives     = SET OF Drives;

        Attributes      = (WriteProtect,
                           Hidden,
                           SystemFile,
                           Label,
                           Directory,
                           Saved);
        SetOfAttributes = SET OF Attributes;

        DosTime         = CARDINAL;
        DosDate         = CARDINAL;

        DTAPtr          = POINTER TO DTA;

        DTA             = RECORD
          Reserved      :   ARRAY [0..19] OF CHAR;
          Attributes    :   SetOfAttributes;
          Time          :   DosTime;
          Date          :   DosDate;
          Length        :   LONGINT;
          Name          :   ARRAY [0..13] OF CHAR;
                        END;

        BasePagePtr     = POINTER TO BasePage;

        BasePage        = RECORD
```

```
            LowTPA          :   SYSTEM.ADDRESS;
            HighTPA         :   SYSTEM.ADDRESS;
            TextBase        :   SYSTEM.ADDRESS;
            TextLength      :   LONGINT;
            DataBase        :   SYSTEM.ADDRESS;
            DataLength      :   LONGINT;
            BssBase         :   SYSTEM.ADDRESS;
            BssLength       :   LONGINT;
            DefDTA          :   DTAPtr;
            ParentBasePage  :   BasePagePtr;
            ra              :   SYSTEM.ADDRESS;
            Environment     :   SYSTEM.ADDRESS;
            rb              :   ARRAY [030H..07FH] OF CHAR;
            CommandLine     :   ARRAY [0..1FH] OF CHAR;
                              END;

          OpenModes         = (ReadOnly, WriteOnly, ReadWrite);

          SeekModes         = (Abs, Rel, AbsReverse);

          LoadModes         = (LoadStart, Res1, Res2,
                               LoadOnly, StartOnly, CrBasePage);

          GetModes          = (Set, Get);

          FileTimes         = RECORD
            Time            :   DosTime;
            Date            :   DosDate;
                              END;

          DiskInfo          = RECORD
            FreeClusters    :   LONGINT;
            TotalClusters   :   LONGINT;
            SectorSize      :   LONGINT;
            ClusterSize     :   LONGINT;
                              END;

          Handles           = INTEGER;

          Paths             = ARRAY [0..64] OF CHAR;

VAR       Result            : LONGINT;


(*----- Category : GemDos Functions ------------------------------------*)

PROCEDURE Term0;

PROCEDURE ConIn           (VAR Ch         : CHAR;
                           VAR ScanCode   : ScanCodes);

PROCEDURE ConOut          (    Ch         : CHAR);

PROCEDURE AuxIn           (VAR Ch         : CHAR);

PROCEDURE AuxOut          (    Ch         : CHAR);

PROCEDURE PrnOut          (    Ch         : CHAR);

PROCEDURE ConRawIO        (VAR Ch         : CHAR);

PROCEDURE ConRawIn        (VAR Ch         : CHAR);

PROCEDURE ConNegIn        (VAR Ch         : CHAR);
```

```
PROCEDURE ConWriteString(    Line       : ARRAY OF CHAR);

PROCEDURE ConReadString (VAR Line       : ARRAY OF CHAR);

PROCEDURE ConInStat      () : BOOLEAN;

PROCEDURE SetDrv         (    Drive      : Drives)
                         : SetOfDrives;

PROCEDURE ConOutStat     () : BOOLEAN;

PROCEDURE PrnOutStat     () : BOOLEAN;

PROCEDURE AuxInStat      () : BOOLEAN;

PROCEDURE AuxOutStat     () : BOOLEAN;

PROCEDURE GetDrive       () : Drives;

PROCEDURE SetDTA         (VAR Dta        : DTA);

PROCEDURE Super          (VAR Stck       : LONGINT);

PROCEDURE GetDate        (VAR Today      : DosDate);

PROCEDURE SetDate        (    Today      : DosDate);

PROCEDURE GetTime        (VAR Now        : DosTime);

PROCEDURE SetTime        (    Now        : DosTime);

PROCEDURE GetDTA         () : DTAPtr;

PROCEDURE Version        () : CARDINAL;

PROCEDURE TermResident   (    Memory     : LONGCARD;
                              Return     : INTEGER);

PROCEDURE DiskFree       (VAR Info       : DiskInfo;
                              Drive      : Drives);

PROCEDURE DirCreate      (    Name       : ARRAY OF CHAR);

PROCEDURE DirDelete      (    Name       : ARRAY OF CHAR);

PROCEDURE SetPath        (    Name       : ARRAY OF CHAR);

PROCEDURE Create         (    Name       : ARRAY OF CHAR;
                              Attribute  : SetOfAttributes)
                         : Handles;

PROCEDURE Open           (    Name       : ARRAY OF CHAR;
                              Mode       : OpenModes)
                         : Handles;

PROCEDURE Close          (    Handle     : Handles);

PROCEDURE Read           (    Handle     : Handles;
                              Buffer     : SYSTEM.ADDRESS;
                              Size       : LONGINT)
                         : LONGINT;

PROCEDURE Write          (    Handle     : Handles;
                              Buffer     : SYSTEM.ADDRESS;
                              Size       : LONGINT)
```

```
                        : LONGINT;

PROCEDURE Delete        (    Name        : ARRAY OF CHAR);

PROCEDURE Seek          (    Handle      : Handles;
                             Mode        : SeekModes;
                             Position    : LONGINT);


PROCEDURE Attribute     (    Name        : ARRAY OF CHAR;
                             Mode        : GetModes;
                         VAR Attrib      : SetOfAttributes);

PROCEDURE Dup           (    StdHandle   : Handles)
                        : Handles;

PROCEDURE Force         (    StdHandle   : Handles;
                             NonStdHndl  : Handles);

PROCEDURE GetPath       (VAR Path        : Paths;
                             Drive       : Drives);

PROCEDURE MemAlloc      (    Amount      : LONGINT)
                        : LONGINT;

PROCEDURE MemFree       (    Block       : SYSTEM.ADDRESS);

PROCEDURE Shrink        (    Block       : SYSTEM.ADDRESS;
                             Size        : LONGINT);

PROCEDURE Exec          (    Mode        : LoadModes;
                             Path        : ARRAY OF CHAR;
                             CmdLine     : ARRAY OF CHAR;
                             Environment: ARRAY OF CHAR)
                        : LONGINT;

PROCEDURE Term          (    Return      : INTEGER);

PROCEDURE SearchFirst   (    Spec        : ARRAY OF CHAR;
                             Attr        : SetOfAttributes);

PROCEDURE SearchNext;

PROCEDURE Rename        (    OldName     : ARRAY OF CHAR;
                             NewName     : ARRAY OF CHAR);

PROCEDURE Timestamp     (VAR DatTim      : FileTimes;
                             Handle      : Handles;
                             Mode        : GetModes);

END GemDos.
```

```
(*------------------------------------------------------------------------*)
(*---                      Module LineA  V 1.1                         ---*)
(*---                     ---------------------                        ---*)
(*---                                                                  ---*)
(*---  Die graphischen Grundroutinen des Atari ST                      ---*)
(*---                                                                  ---*)
(*--- Programmiersprache :  SPC-Modula-2 V1.3                          ---*)
(*--- Computersystem     :  ATARI 1040 ST                              ---*)
(*--- Autor              :  Uwe A. Ruttkamp                            ---*)
(*--- Datum              :  21.10.1988                                 ---*)
(*---                                                                  ---*)
(*------------------------------------------------------------------------*)

DEFINITION MODULE LineA;

FROM SYSTEM IMPORT ADDRESS;

TYPE

 FontPointer = POINTER TO FontTyp;
 FontTyp = RECORD
       FaceID        : CARDINAL;     (* Fontnummer                    *)
       FontSize      : CARDINAL;     (* Fontgröße in Satzpunkten      *)
       FaceName      : ARRAY [0..31] OF CHAR;
                                     (* Namen des Zeichensatzes       *)
       LowADE        : CARDINAL;     (* kleinster ASCII-Wert          *)
       HighADE       : CARDINAL;     (* größter ASCII-Wert            *)
       TopLine       : CARDINAL;     (* Abstand Top-Baseline          *)
       AscentLine    : CARDINAL;     (* Abstand Ascent-Baseline       *)
       HalfLine      : CARDINAL;     (* Abstand Half-Baseline         *)
       DescentLine   : CARDINAL;     (* Abstand Descent-Baseline      *)
       BottomLine    : CARDINAL;     (* Abstand Bottom-Baseline       *)
       MaxFontWidth  : CARDINAL;     (* maximale Zeichenbreite        *)
       MaxFaceWidth  : CARDINAL;     (* maximale Zeichenzellenbreite *)
       LeftOffset    : CARDINAL;     (* Offset links                  *)
       RightOffset   : CARDINAL;     (* Offset rechts                 *)
       Thickening    : CARDINAL;     (* Verbreiterungsfaktor          *)
       UnderlineSize : CARDINAL;     (* Unterstreichungsdicke         *)
       LightMask     : CARDINAL;     (* Maske für helle Schrift       *)
       SkewMask      : CARDINAL;     (* Maske für Kursivschrift       *)
       Flags         : BITSET;       (* Bits: 0 Sytstemfont
                                              1 Horizontal Offset Tabelle
                                              2 Formatflag
                                              3 aus: proportional
                                                ein: mono-spaced     *)
       HorizOffsetTab: POINTER TO ARRAY [0C..377C] OF CARDINAL;
                                     (* Horizontal Offset Tabelle     *)
       CharOffsetTab: POINTER TO ARRAY [0C..377C] OF CARDINAL;
                                     (* Zeichen Offset Tabelle        *)
       FontData      : ADDRESS;      (* Zeiger auf Zeichensatzdaten   *)
       FormWidth     : CARDINAL;     (* Breite des Zeichensatzimage   *)
       FormHeight    : CARDINAL;     (* Höhe des Zeichensatzes        *)
       NextFont      : POINTER TO FontTyp;
                                     (* Zeiger auf nächsten Font      *)
 END;
 FontArray = ARRAY [0..2] OF FontPointer;

 LineAVarPointer = POINTER TO LineAVarRecord;
 LineAVarRecord = RECORD
     VideoPlanes   : CARDINAL;  (* Anzahl der Bildschirmebenen      *)
     VBytesLine    : CARDINAL;  (* Bytes pro Bildschirmzeile        *)
     (* Die Control Arrays *)
     Contrl        : POINTER TO ARRAY [0..11 ] OF INTEGER;
     IntIn         : POINTER TO ARRAY [0..127] OF INTEGER;
     PtsIn         : POINTER TO ARRAY [0..127] OF INTEGER;
```

```
    IntOut        : POINTER TO ARRAY [0..127] OF INTEGER;
    PtsOut        : POINTER TO ARRAY [0..127] OF INTEGER;
    Bp1           : CARDINAL;  (* Farbwert für Plane 0          *)
    Bp2           : CARDINAL;  (* Farbwert für Plane 1          *)
    Bp3           : CARDINAL;  (* Farbwert für Plane 2          *)
    Bp4           : CARDINAL;  (* Farbwert für Plane 3          *)
    LstLin        : INTEGER;   (* auf -1 setzen                 *)
    LineStyle     : CARDINAL;  (* Linienmuster                  *)
    WriteMode     : CARDINAL;  (* VDI-Schreibmodus              *)
    X1,Y1,X2,Y2   : CARDINAL;  (* Koordinaten                   *)
    PatPointer    : ADDRESS;   (* Zeiger auf Füllmuster         *)
    PatMask       : CARDINAL;  (* Füllmuster Maske              *)
    MultiFill     : CARDINAL;  (* Füllmuster mono / farbig      *)
    Clip          : CARDINAL;  (* Clipping aus/an               *)
    XMinClip      : CARDINAL;  (* linke obere Ecke des Clip     *)
    YMinClip      : CARDINAL;  (* Rechtecks                     *)
    XMaxClip      : CARDINAL;  (* rechte untere Ecke des Clip   *)
    YMaxClip      : CARDINAL;  (* Rechtecks                     *)
    XAccData      : CARDINAL;  (* vor Textausgabe auf 8000H     *)
    DDAInc        : CARDINAL;  (* Vergrößerungsfaktor           *)
    ScaleDir      : CARDINAL;  (* Vergr. Richtung (1=vergrößern)*)
    MonoStatus    : CARDINAL;  (* Proportionalschrift ja/nein   *)
    SourceX       : CARDINAL;  (* X Koordinate im Zeichensatz   *)
    SourceY       : CARDINAL;  (* Y Koordinate im Zeichensatz   *)
    DestX         : CARDINAL;  (* X Koordinate auf dem Bildschirm*)
    DestY         : CARDINAL;  (* Y Koordinate auf dem Bildschirm*)
    DeltaX        : CARDINAL;  (* Breite des Zeichens           *)
    DeltaY        : CARDINAL;  (* Höhe des Zeichens             *)
    FontBase      : FontPointer;(* Zeiger auf Zeichensatzimage  *)
    FontWidth     : CARDINAL;  (* Breite des Zeichensatzimage   *)
    Style         : CARDINAL;  (* Schreibstil                   *)
    LightMask     : CARDINAL;  (* Maske für Schattierung        *)
    SkewMask      : CARDINAL;  (* Maske für Italics             *)
    Weight        : CARDINAL;  (* zusätzliche Breite bei Bold   *)
    ROff          : CARDINAL;  (* Kursiv-Offset rechts          *)
    LOff          : CARDINAL;  (* Kursiv-Offset links           *)
    Scale         : CARDINAL;  (* Vergrößerung ja/nein          *)
    CharUp        : CARDINAL;  (* Rotationswinkel * 10          *)
    TextFG        : CARDINAL;  (* Textfarbe                     *)
    Scrtchp       : ADDRESS;   (* Zeiger auf Texteffektbuffer   *)
    Scrpt2        : CARDINAL;  (* Offset für Texteffektbuffer   *)
    TextBG        : CARDINAL;  (* Text Hintergrundfarbe         *)
    CopyTran      : CARDINAL;  (* Copy Raster Form Flag         *)
END;

LineAVDIPointer = POINTER TO LineAVDIRecord;
LineAVDIRecord = RECORD
     InquireTab    : ARRAY [0..44]  OF INTEGER; (* VDI Inquire Werte     *)
     DeviceTab     : ARRAY [0..56]  OF INTEGER; (* VDI Workst. Werte     *)
     Reserved      : ARRAY [0..265] OF INTEGER; (* Platzhalter           *)
     CelHeight     : CARDINAL;  (* Zeichenhöhe          *)
     CelMaxX       : CARDINAL;  (* Maximale Cursor X-Position           *)
     CelMaxY       : CARDINAL;  (* Mixiamale Cursor Y-Position          *)
     CelLineWidth  : CARDINAL;  (* Breite einer Characterzeile in Bytes *)
     BGColor       : CARDINAL;  (* Hintergrundfarbe                     *)
     FGColor       : CARDINAL;  (* Vordergrundfarbe                     *)
     CursorAdr     : ADDRESS;   (* Adresse der aktuellen Cursorposition *)
     CursorOffset  : INTEGER;   (* Vert. Offset zum phys. Bildsch.anfang*)
     CursorX       : CARDINAL;  (* X-Position des Cursors               *)
     CursorY       : CARDINAL;  (* Y-Position der Cursors               *)
     CurBlinkCnt   : CHAR;      (* Cursor Blinkgeschwindigkeit          *)
     CurBlinkTim   : CHAR;      (* Zähler für Cursorblinken             *)
     FontAdr       : FontPointer;(* Zeiger auf Systemzeichensatzdaten   *)
     LastFontChar  : CARDINAL;  (* Letztes Zeichen im Zeichensatz       *)
     FirstFontChar : CARDINAL;  (* Erstes  Zeichen im Zeichensatz       *)
```

```
    FontWidth     : CARDINAL;  (* Breite des Fontdaten in Bytes         *)
    PixelWidth    : CARDINAL;  (* Bildschirmbreite in Pixeln            *)
    FontOffsetAdr : ADDRESS;   (* Zeiger auf Zeichensatz-Offset-Tabelle *)
    CursorFlag    : BITSET;    (* Bestimmt des Verhalten des Cursor     *)
    PixelHeight   : CARDINAL;  (* Bildschirmhöhe in Pixeln              *)
    PixLineWidth  : CARDINAL;  (* Bytes pro Pixelzeile                  *)
END;

BitBltPointer = POINTER TO BitBltRecord;
BitBltRecord = RECORD
    BlockWidth  : CARDINAL;              (* Breite des Blocks in Pixeln  *)
    BlockHeight : CARDINAL;              (* Höhe des Blocks in Pixeln    *)
    PlaneCount  : CARDINAL;              (* Anzahl der Farbplanes        *)
    FGColor     : CARDINAL;              (* Vordergrundfarbe             *)
    BGColor     : CARDINAL;              (* Hintergrundfarbe             *)
    OpTable     : ARRAY [0..3] OF CHAR; (* Logische Verknüpfungstabelle *)
    SourceX     : CARDINAL;              (* X-Koord. des Quellrasters    *)
    SourceY     : CARDINAL;              (* Y-Koord. des Quellrasters    *)
    SourcePtr   : ADDRESS;               (* Anfangsadresse des Q-Rasters *)
    SNextWord   : CARDINAL;              (* Highres 2,Midres 4,Lowres 8  *)
    SNextLine   : CARDINAL;              (* Breite des Q-Rasters in Bytes*)
    SNextPlane  : CARDINAL;              (* gleich 2                     *)
    DestX       : CARDINAL;              (* X-Koord. des Zielrasters     *)
    DestY       : CARDINAL;              (* Y-Koord. des Zielrasters     *)
    DestPtr     : ADDRESS;               (* Anfangsadresse des Z-Rasters *)
    DNextWord   : CARDINAL;              (* Highres 2,Midres 4,Lowres 8  *)
    DNextLine   : CARDINAL;              (* Breite des Z-Rasters in Bytes*)
    DNextPlane  : CARDINAL;              (* gleich 2                     *)
    PatternPtr  : ADDRESS;               (* Undierung mit dieser Maske   *)
    PNextLine   : CARDINAL;              (* Breite der Maske in Bytes    *)
    PNextPlane  : CARDINAL;              (* Offset zur folgenden Plane   *)
    PMaskHeight : CARDINAL;              (* Höhe der Maske in Zeilen     *)
    reserved    : ARRAY [0..23] OF CHAR;
END;

MFormPointer = POINTER TO MFormRecord;
MFormRecord = RECORD
    XHot, YHot       : CARDINAL;
    NoPlanes         : CARDINAL;
    MaskCol, CursCol : CARDINAL;
    Mask             : ARRAY [0..15] OF CARDINAL;
    Cursor           : ARRAY [0..15] OF CARDINAL;
END;

SDBPointer = POINTER TO SDBRecord;
SDBRecord = RECORD
    XHot, YHot       : CARDINAL;
    Form             : INTEGER;
    BgColor, FgColor : CARDINAL;
    Image            : ARRAY [0..31] OF CARDINAL;
END;


WriteModes = ( Replace, Transparent, Invert, InverseTransparent );

PROCEDURE Initialize () : LineAVarPointer;

(*--- Mit dieser Prozedur werden die LineA-Funktionen für jede An-  ---*)
(*--- wendung zugänglich. Initialiesiert wird mit Initialize()      ---*)
(*--- nichts, aber man kann mit der Kenntnis des LineAVarRecord's   ---*)
(*--- weitere Operationen implementieren. Siehe Profibuch. Für die  ---*)
(*--- Benutzung der weiteren LineA-Funktionen ist Initialize() ohne ---*)
(*--- Bedeutung.                                                    ---*)
```

```
PROCEDURE VDIDescription () : LineAVDIPointer;

(*--- Liefert einen Pointer auf eine Strukur, die sehr wichtige      ---*)
(*--- Informationen bezüglich Bildschirmauflösung und Zeichensatz   ---*)
(*--- enthält. Man spart sich den Weg übers VDI, das ist alles.     ---*)

PROCEDURE GetFont( VAR Fonts : FontArray );

(*--- Liefert eine Tabelle mit den Zeigern auf die drei Systemfonts ---*)

PROCEDURE PutPixel ( X, Y, Color : CARDINAL );

(*--- Setzt einen Pixel absolut mit den gegebenen X-Y-Pixelkoordi-  ---*)
(*--- naten auf dem Bildschirm. Den Farbwert nachlesen im Profibuch.---*)


PROCEDURE GetPixel ( X, Y : CARDINAL ) : CARDINAL;

(*--- Mit GetPixel kann man abfragen welchen Farbwert ein Pixelpunkt---*)
(*--- auf dem Bildschirm hat.                                        ---*)


PROCEDURE Line ( X1, Y1, X2, Y2      : CARDINAL;
                 Bp1, Bp2, Bp3, Bp4 : CARDINAL;
                 LineStyle          : CARDINAL;
                 WrtMode            : WriteModes );

(*--- Zeichnet eine, die beiden Punkte verbindende, Linie unter Be- ---*)
(*--- rücksichtigung der Bitplans Bp1 bis Bp4, einem Liniemuster    ---*)
(*--- LineStyle und dem Verknüpfungsmodi WrtMode.                   ---*)


PROCEDURE Pline( X1, Y1, X2, Y2 : CARDINAL;
                 WrtMode        : WriteModes );

(*--- Entspricht exakt der Line Funktion mit dem Wert 1 für Bp1 bis ---*)
(*--- Bp4 und 65535 für LineStyle. So wird man meistens Linien      ---*)
(*--- ziehen.                                                       ---*)


PROCEDURE HorizLine ( X1, Y1, X2         : CARDINAL;
                      Bp1, Bp2, Bp3, Bp4 : CARDINAL;
                      WrtMode            : WriteModes;
                      PatPointer         : ADDRESS;
                      PatMask            : CARDINAL;
                      MultiFill          : BOOLEAN);

(*--- HorizLine ist angeblich schneller als Line, aber auch kompli- ---*)
(*--- zierter. PatPointer zeigt auf eine Sammlung von Musterlinien, ---*)
(*--- PatMask ist die Anzahl der Linienmuster und MultiFill bestimmt---*)
(*--- ob sich die Linie auf alle Planes durchschlagen soll.         ---*)


PROCEDURE FillRectangle( X1, Y1, X2, Y2     : CARDINAL;
                         Bp1, Bp2, Bp3, Bp4 : CARDINAL;
                         WrtMode            : WriteModes;
                         PatPointer         : ADDRESS;
                         PatMask            : CARDINAL;
                         MultiFill          : BOOLEAN;
                         Clip               : BOOLEAN;
                         XMinClip, YMinClip,
                         XMaxClip, YMaxClip : CARDINAL );

(*--- Füllt ein Rechteck mit den entsprechenden Farben in BpX       ---*)
(*--- und beachtet das Cliprechteck um nicht darüberhinaus zu       ---*)
```

```
(*--- malen                                                          ---*)


PROCEDURE FillPolygon( Coords              : ARRAY OF INTEGER;
                       CoordNo, Y          : CARDINAL;
                       Bp1, Bp2, Bp3, Bp4  : CARDINAL;
                       WrtMode             : WriteModes;
                       PatPointer          : ADDRESS;
                       PatMask             : CARDINAL;
                       MultiFill           : BOOLEAN;
                       Clip                : BOOLEAN;
                       XMinClip, YMinClip,
                       XMaxClip, YMaxClip  : CARDINAL );

(*--- Die in Coords eingetragenen Koordinaten (X0,Y0,X1,Y1...)     ---*)
(*--- bestimmen eine Fläche, die bei jedem Aufruf ein wenig mehr   ---*)
(*--- gefüllt wird. Y gibt dabei an welche Linie gerade an der     ---*)
(*--- Reihe sein soll.  Näheres : Profibuch.                       ---*)


PROCEDURE BitBlt( Ptr : BitBltPointer );

(*--- Siehe BitBltRecord für die Parameter. Aufgabe von BitBlt ist ---*)
(*--- es Bildschirmausschnitte zu kopieren. Die Anwendungen sind   ---*)
(*--- vielzählig.                                                  ---*)

PROCEDURE TextBlt( WrtMode             : WriteModes;
                   Clip                : BOOLEAN;
                   XMinClip, YMinClip,
                   XMaxClip, YMaxClip  : CARDINAL;
                   TextFG, TextBG      : CARDINAL;
                   FontBase            : FontPointer;
                   FontWidth           : CARDINAL;
                   SourceX, SourceY    : CARDINAL;
                   DeltaX, DeltaY      : CARDINAL;
                   Style               : CARDINAL;
                   LightMask           : CARDINAL;
                   SkewMask            : CARDINAL;
                   Weight              : CARDINAL;
                   ROff, LOff          : CARDINAL;
                   Scale               : BOOLEAN;
                   DDAInc, ScaleDir    : CARDINAL;
                   CharUp              : CARDINAL;
                   MonoStatus          : CARDINAL;
                   Scrtchp             : ADDRESS;
                   Scrpt2              : CARDINAL  );

  (*--- Ausgabe von einzelnen Zeichen auf dem Bildschirm. Die      ---*)
  (*--- Parameternamen stimmen mit denen im LineAVarRecord über-   ---*)
  (*--- und ihre Funktion ist dort nachzulesen.                    ---*)

PROCEDURE ShowMouse( Absolute : BOOLEAN );

(*--- Wenn Absolute = TRUE ist, wird die Maus auf jeden Fall       ---*)
(*--- wieder eingeschaltet, sonst nur, wenn zuvor nur einmal       ---*)
(*--- HideMouse aufgerufen worden ist.                             ---*)


PROCEDURE HideMouse;

(*--- Schaltet die Maus aus und vergrößert die Verschachtelungs-   ---*)
(*--- tiefe im Wechselspiel von Maus AN-AUS.                       ---*)

PROCEDURE TransformMouse( Ptr : MFormPointer);
```

```
(*--- Hiermit kann man sich seine eigene Maus kreieren.          ---*)

PROCEDURE UndrawSprite( Save : ADDRESS  );

(*--- Löscht ein mit DrawSprite gemaltet Sprite. Save muß natürlich ---*)
(*--- ein Speicherbereich sein, denn man sich zuvor mit DrawSprite  ---*)
(*--- hat geben lassen. Dadurch wird der Hintergrund wieder herge-  ---*)
(*--- stellt.                                                       ---*)


PROCEDURE DrawSprite( XHot, YHot  : CARDINAL;
                      SDB         : SDBPointer;
                      Save        : ADDRESS );

(*--- Malt ein Sprite, wie in SDB beschrieben. Save muß auf einen  ---*)
(*--- Speicherbereich zeigen, der mindstens 10+64*Anzahl der Farb- ---*)
(*--- ebenen Bytes groß ist.                                       ---*)

END LineA.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIAttributes;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987           *)


(*----- Category : Types and Data --------------------------------------*)

TYPE

        WritingModes            = ( IllegalWriteMode,
                                    Replace,     Transparent,
                                    Xor,         ReverseTransparent);

        ColourRange             =  INTEGER;
        ColourIntensity         = [ 0 .. 1000 ];

        ColourComposition       = RECORD
          Red, Green, Blue      : ColourIntensity;
                                  END;

        TenthDegree             = [ 0 .. 3600 ];

        LineStyles      = ( IllegalLineStyle,
                            Solid   , LongDash , Dots,
                            DashDot , DashDash , DashDotDot,
                            UserDefinedLine );

        LineEndStyles   = ( Normal, Arrow, Rounded );

        MarkerTypes     = ( IllegalMarker,
                            Dot     ,   Plus          ,   Asterisk,
                            Square  ,   DiagonalCross ,   Diamond );

        TextEffect      = ( Bold      , Light    , Italic,
                            Underlined , Outlined , Shadowed );

        TextEffects     = SET OF TextEffect;

        FillStyles      = ( Hollow , Filled  , Pattern,
                            Hatch  , UserDefinedInterior );

        FillRange       = [ 1 .. 24 ];

        FontTypes       = ( IllegalFont,
                            BigFont,     SmallFont);

        HorAlignment    = ( LeftJustified, Centered, RightJustified );

        VertAlignment   = ( Baseline   , HalfLine    , AscentLine,
                            BottomLine , DescentLine , TopLine);
```

```
        CoordinateTypes = ( NormalCoords,
                            ReservedCoords,
                            RasterCoords);

(* attribute functions *)

PROCEDURE SetWritingMode        (    Handle     : INTEGER;
                                     Mode       : WritingModes)
                                : WritingModes;

(* set mode used for subsequent drawing operations *)


PROCEDURE SetColour             (    Handle     : INTEGER;
                                     ColourIndex: ColourRange;
                                     RGBIn      : ColourComposition );

(* set colour representation *)


PROCEDURE SetLineType           (    Handle     : INTEGER;
                                     Style      : LineStyles )
                                : LineStyles;

(* set polyline line type *)


PROCEDURE DefineLineStyle       (   Handle      : INTEGER;
                                    Pattern     : INTEGER );

(* Set user-defined line style using the bits of pattern-parameter *)


PROCEDURE SetLineWidth          (    Handle,
                                     Width      : INTEGER )
                                : INTEGER;

(* Set polyline line width *)


PROCEDURE SetLineColour         (    Handle     : INTEGER;
                                     ColourIndex: ColourRange)
                                : ColourRange;

(* sets colour index for subsequent polyline operations *)


PROCEDURE SetEndLineStyle       (   Handle      : INTEGER;
                                    BeginStyle,
                                    EndStyle    : LineEndStyles );

(* set polyline end styles *)


PROCEDURE SetMarkerType         (    Handle     : INTEGER;
                                     Symbol     : MarkerTypes )
                                : MarkerTypes;

(* set polymarker type *)


PROCEDURE SetMarkerHeight       (   Handle,
                                    Height      : INTEGER )
```

```
                                : INTEGER;

(* Set polymarker height *)

PROCEDURE SetMarkerColour       (    Handle     : INTEGER;
                                     ColourIndex: ColourRange )
                                : ColourRange;

(* set polymarker colour index *)

PROCEDURE SetAbsCharHeight      (    Handle,
                                     AbsoluteHeight: INTEGER;
                                VAR CharWidth,
                                     CharHeight,
                                     CellWidth,
                                     CellHeight : INTEGER );

(* Set character height, absolute mode *)

PROCEDURE SetPointCharHeight    (    Handle,
                                     HeightInPoints: INTEGER;
                                VAR CharWidth,
                                     CharHeight,
                                     CellWidth,
                                     CellHeight : INTEGER )
                                : INTEGER;

(* set character cell height, points mode *)

PROCEDURE SetRotation           (    Handle     : INTEGER;
                                     Angle      : TenthDegree)
                                : TenthDegree;

(* set character baseline vector *)

PROCEDURE SetFont               (    Handle     : INTEGER;
                                     Font       : FontTypes )
                                : FontTypes;

(* Set text face *)

PROCEDURE SetGraphicTextColour  (    Handle     : INTEGER;
                                     ColourIndex: ColourRange )
                                : ColourRange;

(* set graphic text colour index *)

PROCEDURE SetGraphicTextEffects (    Handle     : INTEGER;
                                     Effect     : TextEffects )
                                : TextEffects;

(* set graphic text special effects *)

PROCEDURE SetGraphicTextAlignment(   Handle     : INTEGER;
                                     HorIn      : HorAlignment;
                                     VertIn     : VertAlignment;
                                VAR HorOut      : HorAlignment;
```

```
                                VAR VertOut   : VertAlignment );

(* Set graphic text alignment *)


PROCEDURE SetFillInteriorStyle  (    Handle     : INTEGER;
                                     Style      : FillStyles )
                                : FillStyles;

(* set fill interior style *)


PROCEDURE SetFillStyleIndex     (    Handle     : INTEGER;
                                     StyleIndex : FillRange )
                                : FillRange;

(* Set fill style index *)


PROCEDURE SetFillColour         (    Handle     : INTEGER;
                                     ColourIndex: ColourRange )
                                : ColourRange;

(* set fill colour index *)


PROCEDURE SetFillPerimeterVisibility (Handle    : INTEGER;
                                      PerVisible: BOOLEAN )
                                      : BOOLEAN;

(* set fill perimeter visibility *)


PROCEDURE DefineFillPattern     (    Handle     : INTEGER;
                                 VAR PFillPatt  : ARRAY OF INTEGER;
                                     Planes     : INTEGER );

(* Set user-defined fill pattern *)

END VDIAttributes.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name           : *) VDIBase;
(* .      Function       : *)
(* .      Version/Date   : 1.02 / 19.07.1987                            *)
(* Product Name          : SPC                                         *)
(* Copyright             : (c) 1987,1988, MODsoft, D7500 Karlsruhe     *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM SYSTEM IMPORT ADDRESS;

TYPE
        CtrlArrayType           = ARRAY [0..127] OF INTEGER;

VAR
        contrl          : ARRAY [0..11] OF INTEGER;

        intin,
        ptsin,
        intout,
        ptsout          : CtrlArrayType;

        ADRintin,
        ADRptsin,
        ADRintout,
        ADRptsout,
        ADRParams       : ADDRESS;

        parameterBlock  : ARRAY [0..4] OF ADDRESS;

(*----- Category : interface procedures --------------------------------*)

PROCEDURE CallVDI;

PROCEDURE SetContrl     (   c0, c1,
                            c3, c5,
                            c6          : INTEGER );

END VDIBase.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIControls;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract ---------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987           *)


(*----- Category : Types and Data ----------------------------------------*)

FROM VDIOutputs IMPORT  VDIRectangle;

FROM VDIAttributes      IMPORT  LineStyles,
                                MarkerTypes,
                                ColourRange,
                                FontTypes,
                                FillStyles,
                                FillRange,
                                CoordinateTypes;

TYPE
        DeviceTypes     = INTEGER;

        WorkstationType = ( OutputDevice,
                            InputDevice,
                            InOutDevice,
                            ReservedDevice,
                            MetafileDevice);

        WorkstationInitRec      = RECORD
          DeviceId              : DeviceTypes;
          LineStyle             : LineStyles;
          LineColour            : ColourRange;
          MarkerType            : MarkerTypes;
          MarkerColour          : ColourRange;
          Font                  : FontTypes;
          TextColour            : ColourRange;
          FillStyle             : FillStyles;
          FillIndex             : FillRange;
          FillColour            : ColourRange;
          CoordinateSystem      : CoordinateTypes;
                                END;

        WorkstationDescription  = RECORD
          RasterWidthOfScreen,
          RasterHeightOfScreen  : CARDINAL;
          reserved0             : CARDINAL;
          HorRasterIncrement,
          VertRasterIncrement   : CARDINAL; (* mm/1000 *)
          MaxTextSizes,
          MaxLineStyles,
          MaxLineWidths,
          MaxMarkers,
```

```
            MaxMarkerSizes,
            MaxFonts,
            MaxPatterns,
            MaxHatchings,
            MaxColours,
            MaxBasicGraphFuntions : CARDINAL;
            SupportedGraphFuncs   : ARRAY [ 0 .. 9 ] OF INTEGER;
            SupportedAttributes   : ARRAY [ 0 .. 9 ] OF CARDINAL; (* ???? *)
            ColoursPossible,
            TextRotationPossible,
            FillInteriorPossible,
            FuncCellArrayPossible : BOOLEAN;
            MaxPossibleColours    : CARDINAL;
            LocatorControl        : CARDINAL;
            ValuatorControl       : CARDINAL;
            ChoiceControl         : CARDINAL;
            StringControl         : CARDINAL;
            TypeOfWorkstation     : WorkstationType;
            MinCharWidth,
            MinCharHeight,
            MaxCharWidth,
            MaxCharHeight,
            MinLineWidth,
            Reserved0a,
            MaxLineWidth,
            Reserved0b,
            MinMarkerWidth,
            MinMarkerHeight,
            MaxMarkerWidth,
            MaxMarkerHeight       : CARDINAL;
                                  END;

(* Control functions *)

PROCEDURE OpenWorkstation       (VAR WorkIn     : WorkstationInitRec;
                                 VAR Handle     : INTEGER;
                                 VAR WorkOut    : WorkstationDescription );

(* loads a device driver, and initialises device with parameters passed *)


PROCEDURE CloseWorkstation      (    Handle     : INTEGER );

(* closes graphics device properly, and returns to alpha mode *)


PROCEDURE OpenVirtualWorkstation( VAR WorkIn    : WorkstationInitRec;
                                  VAR Handle    : INTEGER;
                                  VAR WorkOut   : WorkstationDescription );

(* open virtual screen workstation *)


PROCEDURE CloseVirtualWorkstation(    Handle    : INTEGER );

(* close virtual device, preventing further output to it *)


PROCEDURE ClearWorkstation      (    Handle     : INTEGER );

(* clear workstation. Erases the screen *)


PROCEDURE UpdateWorkstation     (    Handle     : INTEGER );
```

```
(* execute immediately all pending graphics commands *)

PROCEDURE LoadFonts             (    Handle,
                                     Select     : INTEGER )
                          : INTEGER;

(* loads fonts and makes them available *)

PROCEDURE UnloadFonts   (    Handle, Select     : INTEGER );

(* dissociates fonts and removes them from memory *)

PROCEDURE SetClipping   (    Handle     : INTEGER;
                             ClippingOn : BOOLEAN;
                         VAR ClipArea   : VDIRectangle );

(* enable/disable clipping of all ouput by GEM VDI *)

END VDIControls.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIEscapes;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM SYSTEM     IMPORT  ADDRESS;

FROM VDIOutputs IMPORT  VDIRectangle,
                        Coordinate;

TYPE
        FilmNameType    = ARRAY [ 0 .. 124 ] OF INTEGER;
        FilmIndexType   = ARRAY [ 0 .. 7 ],[0 .. 1 ] OF INTEGER;


(* escapes *)

PROCEDURE InquireCharCells      (    Handle     : INTEGER;
                                 VAR rows,
                                     colums     : INTEGER );

(* inquire addressable character cells *)


PROCEDURE ExitAlphaMode         (    Handle     : INTEGER );

(* Exit alpha mode *)


PROCEDURE EnterAlphaMode        (    Handle     : INTEGER );

(* enter alpha mode *)


PROCEDURE CursorUp      (    Handle     : INTEGER );

(* alpha cursor up *)


PROCEDURE CursorDown    (    Handle     : INTEGER );

(* alpha cursor down *)


PROCEDURE CursorRight   (    Handle     : INTEGER );

(* alpha cursor right *)
```

```
PROCEDURE CursorLeft    (    Handle     : INTEGER );
(* alpha cursor left *)

PROCEDURE CursorHome    (    Handle     : INTEGER );
(* home alpha cursor *)

PROCEDURE EraseToEOS    (    Handle     : INTEGER );
(* erase to end of alpha screen *)

PROCEDURE EraseToEOL    (    Handle     : INTEGER );
(* erase to end of alpha text line *)

PROCEDURE CursorAddress (    Handle, row, column : INTEGER );
(* direct alpha cursor address *)

PROCEDURE OutputText    (    Handle     : INTEGER;
                         VAR string     : ARRAY OF CHAR );
(* output cursor addressable alpha text *)

PROCEDURE ReverseVideoOn        (    Handle     : INTEGER );
(* reverse video on *)

PROCEDURE ReverseVideoOff       (    Handle     : INTEGER );
(* reverse video off *)

PROCEDURE InquireCursorAddress  (    Handle     : INTEGER;
                                 VAR row,
                                     column     : INTEGER );
(* inquire current alpha cursor address *)

PROCEDURE InquireTabletStatus   (    Handle     : INTEGER )
                                : INTEGER;
(* inquire tablet status *)

PROCEDURE HardCopy      (    Handle     : INTEGER );
(* hard copy *)

PROCEDURE DisplayCursor         (    Handle     : INTEGER;
                                     Location   : Coordinate );
(* place graphic cursor at location *)
```

```
PROCEDURE RemoveCursor  (    Handle     : INTEGER );

(* remove last graphic cursor *)


PROCEDURE FormAdvance   (    Handle     : INTEGER );

(* form advance *)


PROCEDURE OutputWindow  (    Handle     : INTEGER;
                             Area       : VDIRectangle );

(* output window *)


PROCEDURE ClearDisplayList      (    Handle     : INTEGER );

(* clear display list *)


PROCEDURE OutputBitImageFile    (    Handle     : INTEGER;
                                     FileName   : ARRAY OF CHAR;
                                     aspect,
                                     scaling,
                                     numPts     : INTEGER;
                                     Area       : VDIRectangle );

(* output bit image file *)


PROCEDURE SelectPalette         (    Handle,
                                     palette    : INTEGER )
                                : INTEGER;

(* select palette *)


PROCEDURE InquirePaletteFilms   (    Handle     : INTEGER;
                                VAR FilmNames   : FilmNameType );

(* inquire palette film types *)
(* only for Polaroid recorder *)


PROCEDURE InquirePaletteState   (    Handle     : INTEGER;
                                VAR port,
                                     filmName,
                                     lightness,
                                     interlace,
                                     planes     : INTEGER;
                                VAR indexes     : FilmIndexType );

(* inquire palette driver state *)
(* only for Polaroid recorder *)


PROCEDURE SetPaletteState       (    Handle,
                                     port,
                                     filmName,
                                     lightness,
                                     interlace,
                                     planes     : INTEGER;
                                     indexes    : FilmIndexType );
```

```
(* set palette driver state *)
(* only for Polaroid recorder *)


PROCEDURE SavePaletteState      (    Handle     : INTEGER );

(* save palette driver state *)
(* only for Polaroid recorder *)


PROCEDURE SuppressPaletteMessages       (    Handle     : INTEGER );

(* suppress palette messages *)
(* only for Polaroid recorder *)


PROCEDURE PaletteErrorInquire   (    Handle     : INTEGER )
                                : INTEGER;

(* palette error inquire *)
(* only for Polaroid recorder *)


PROCEDURE UpdateMetafileExtents (    Handle,
                                     minX, minY,
                                     maxX, maxY : INTEGER );

(* update metafile extents *)


PROCEDURE WriteMetafile         (    Handle,
                                     numIntin   : INTEGER;
                                 VAR intIn      : ARRAY OF INTEGER;
                                     numPtsin   : INTEGER;
                                 VAR ptsIn      : ARRAY OF INTEGER );

(* write metafile item *)


PROCEDURE ChangeFileName        (    Handle     : INTEGER;
                                     FileName   : ARRAY OF CHAR );

(* change gem vdi filename *)


PROCEDURE SetLineOffset         (    Handle     : INTEGER;
                                     offset     : INTEGER);

(* set raster offset to start of logical screen (normally 0) *)


PROCEDURE InitSystemFont        (    Handle     : INTEGER;
                                     FontHeader : ADDRESS);

(* install a font as system-font *)
(* width of chars always 8 bit for ATARI ! *)

END VDIEscapes.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIInputs;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                            *)
(* Product Name         : SPC                                         *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe     *)


(*----- Category : Module Abstract -----------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data ------------------------------------*)

FROM SYSTEM             IMPORT  ADDRESS;

FROM VDIOutputs         IMPORT  Coordinate;

FROM VDIAttributes      IMPORT  ColourRange;

TYPE
        MouseFormType   = RECORD
          HotSpot       : Coordinate;
          ReservedEq1   : INTEGER;    (* set to 1 *)
          MaskColour,
          CursorColour  : ColourRange;
          MaskForm,
          CursorForm    : ARRAY [ 0 .. 15 ] OF INTEGER;
                        END;

        DeviceTypes     = ( IllegalDT, Locator, Valuator, Choice, String );
        InputModes      = ( IllegalIM, Request, Sample );
        ValuatorStatus  = ( NoAction, ValuatorChanged, KeypressCharacter);
        KeyboardSpecials= ( SHIFTRight, SHIFTLeft, CTRL, ALT );
        MouseCodes      = ( LeftButton, RightButton );
        EchoType        = ( NoEcho, Echo );

        KeyboardState   = SET OF KeyboardSpecials;
        MouseState      = SET OF MouseCodes;

(* input functions *)

PROCEDURE SetInputMode          (    Handle     : INTEGER;
                                     DevType    : DeviceTypes;
                                     Mode       : InputModes );

(* Set input mode *)


PROCEDURE InputLocatorRQ        (    Handle     : INTEGER;
                                     Location   : Coordinate;
                                VAR RetLocation: Coordinate;
                                VAR Term        : INTEGER );

(* input locator, request mode *)
```

```
PROCEDURE InputLocatorSM        (    Handle      : INTEGER;
                                     Location    : Coordinate;
                                VAR RetLocation: Coordinate;
                                VAR Term        : INTEGER )
                                : INTEGER;

(* input locator, sample mode *)
(* TRUE - changed, FALSE - not changed *)

PROCEDURE InputValuatorRQ       (    Handle,
                                     ValuatorIn : INTEGER;
                                VAR ValuatorOut,
                                     Terminator : INTEGER );

(* input valuator, request mode *)


PROCEDURE InputValuatorSM       (    Handle,
                                     ValIn      : INTEGER;
                                VAR ValOut,
                                     Term       : INTEGER;
                                VAR Status      : ValuatorStatus );

(* input valuator, sample mode *)


PROCEDURE InputChoiceRQ         (    Handle,
                                     ChIn       : INTEGER;
                                VAR ChOut       : INTEGER );

(* input choice, request mode *)


PROCEDURE InputChoiceSM         (    Handle      : INTEGER;
                                VAR Choice      : INTEGER )
                                : BOOLEAN;

(* input choice, sample mode *)
(* TRUE - changed, FALSE - not changed *)


PROCEDURE InputStringRQ         (    Handle,
                                     MaxLength  : INTEGER;
                                     EchoMode   : EchoType;
                                VAR EchoXY      : Coordinate;
                                VAR String      : ARRAY OF CHAR );

(* input string, request mode *)
(* read from keyboard until <CR> or MaxLength encountered *)


PROCEDURE InputStringSM         (    Handle,
                                     MaxLength  : INTEGER;
                                     EchoMode   : EchoType;
                                VAR EchoXY      : Coordinate;
                                VAR String      : ARRAY OF CHAR )
                                : INTEGER;

(* input string, sample mode *)
(* returns length of String *)


PROCEDURE SetMouseForm          (    Handle      : INTEGER;
                                     MouseForm   : MouseFormType );
```

```
(* Set mouse form *)

PROCEDURE ExchangeTimerV        (    Handle     : INTEGER;
                                     TimAddr    : ADDRESS;
                                 VAR OTimAddR   : ADDRESS;
                                 VAR TimConv    : INTEGER );

(* Exchange timer interrupt vector *)

PROCEDURE ShowCursor            (    Handle     : INTEGER;
                                     Reset      : INTEGER );

(* show cursor *)

PROCEDURE HideCursor            (    Handle     : INTEGER );

(* Hide cursor *)

PROCEDURE SampleMouseButton     (    Handle     : INTEGER;
                                 VAR PStatus    : MouseState;
                                 VAR Location   : Coordinate );

(* Sample mouse button state *)

PROCEDURE ExchangeButtonV       (    Handle     : INTEGER;
                                     PUsrCode   : ADDRESS;
                                 VAR PSavCode   : ADDRESS );

(* Exchange button change vector *)

PROCEDURE ExchangeMovementV     (    Handle     : INTEGER;
                                     PUsrCode   : ADDRESS;
                                 VAR PSavCode   : ADDRESS );

(* Exchange mouse movement vector *)

PROCEDURE ExchangeCursorV       (    Handle     : INTEGER;
                                     PUsrCode   : ADDRESS;
                                 VAR PSavCode   : ADDRESS );

(* exchange cursor change vector *)

PROCEDURE SampleKeyboard        (    Handle     : INTEGER;
                                 VAR PStatus    : KeyboardState );

(* sample keyboard state information *)

END VDIInputs.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIInquires;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM VDIAttributes      IMPORT  ColourRange,
                                ColourComposition,
                                TenthDegree,
                                LineStyles,
                                LineEndStyles,
                                HorAlignment,
                                VertAlignment,
                                WritingModes,
                                MarkerTypes,
                                FillStyles,
                                FillRange;

FROM VDIInputs          IMPORT  InputModes;

FROM VDIControls        IMPORT  WorkstationInitRec,
                                WorkstationDescription;

FROM VDIOutputs         IMPORT  Coordinate,
                                VDIRectangle;

TYPE
        DeviceTypes     = ( NoScreen,
                            SeparatScreensAndControllers,
                            SeparatScreensOneController,
                            OneControllerSeparatGraphicMemory,
                            OneControllerOneMemory);

        ValidTextRotations      = ( NoRotations,
                                    NintyDegreeRotations,
                                    ContinuousRotations);

        ExtendWSDescription     = RECORD
          Device                : DeviceTypes;
          MaxBackgroundColours  : CARDINAL;
          MaxTextEffects        : CARDINAL;
          ZoomPossible          : BOOLEAN;
          MaxColourPlanes       : CARDINAL;
          LookUpTablePossible   : BOOLEAN;
          RasterOpsPerSecond    : CARDINAL;
          ContourFillPossible   : BOOLEAN;
          SupportedTextRotations: ValidTextRotations;
          MaxWritingModes       : CARDINAL;
          MaxInputMode          : InputModes;
```

```
    TextAlignmentPossible : BOOLEAN;
    ChangePenPossible     : BOOLEAN;
    ChangeRibbonPossible  : BOOLEAN;
    MaxPointsForOutput    : INTEGER;
    MaxLengthOfintin      : INTEGER;
    MaxMouseButtons       : CARDINAL;
    TypesForWidedLinesPoss: BOOLEAN;
    MxWRModesForWidedLines: CARDINAL;
    Reserved              : ARRAY [ 0 .. 37 ] OF INTEGER;
                          END;

  CombinedWSDescr         = RECORD
    CASE : BOOLEAN OF
      TRUE  : NormalDescr : WorkstationDescription;
    | FALSE : ExtendDescr : ExtendWSDescription;
    END;
  END;


  LineAttrType    = RECORD
    LineStyle     : LineStyles;
    LineColour    : ColourRange;
    WriteMode     : WritingModes;
    BeginStyle,
    EndStyle      : LineEndStyles;
    LineWidth     : CARDINAL;
  END;

  MarkerAttrType  = RECORD
    MarkerType    : MarkerTypes;
    MarkerColour  : ColourRange;
    WriteMode     : WritingModes;
    MarkerWidth,
    MarkerHeight  : CARDINAL;
  END;

  FillAttrType    = RECORD
    FillStyle     : FillStyles;
    FillColour    : ColourRange;
    FillType      : FillRange;
    WriteMode     : WritingModes;
    FrameVisible  : BOOLEAN;
  END;

  TextAttrType    = RECORD
    Font          : INTEGER;
    ColourIndex   : ColourRange;
    Rotation      : TenthDegree;
    HorOrient     : HorAlignment;
    VertOrient    : VertAlignment;
    WriteMode     : WritingModes;
    CharWidth,
    CharHeight,
    CellWidth,
    CellHeight    : INTEGER;
                  END;

  ExtendCoords    = ARRAY [ 0 .. 3 ] OF Coordinate;

(* Inquire functions *)

PROCEDURE ExtendedInquire       (    Handle     : INTEGER;
                                     Extended   : BOOLEAN;
                                VAR WorkOut     : CombinedWSDescr);
```

```
(* Extended Inquire function *)

PROCEDURE InquireColour          (    Handle      : INTEGER;
                                  VAR ColourIndex: ColourRange;
                                      SetFlag     : INTEGER;
                                  VAR RGB         : ColourComposition );

(* inquire colour representation *)


PROCEDURE InquireLineAttributes (     Handle     : INTEGER;
                                   VAR Attrib     : LineAttrType );

(* inquire polyline attributes *)


PROCEDURE InquireMarkerAttributes(    Handle     : INTEGER;
                                    VAR Attrib     : MarkerAttrType );

(* inquire polymarker attributes *)


PROCEDURE InquireFillAttributes (    Handle      : INTEGER;
                                  VAR Attrib     : FillAttrType );

(* inquire fill area attributes *)


PROCEDURE InquireTextAttributes (    Handle      : INTEGER;
                                  VAR Attrib     : TextAttrType );

(* Inquire graphic text attributes *)


PROCEDURE InquireTextExtent      (    Handle      : INTEGER;
                                      String      : ARRAY OF CHAR;
                                  VAR Extent     : ExtendCoords );

(* Inquire text extent *)
(* Extent: 4 corner-coordinates of the extent-rectangle *)


PROCEDURE InquireCharWidth       (    Handle      : INTEGER;
                                      Character   : CHAR;
                                  VAR CellWidth,
                                      LeftDelta,
                                      RightDelta : INTEGER )
                                 : INTEGER;

(* inquire character cell width *)
(* RETURN -1 - character not allowed
            n - ORD(character)        *)

PROCEDURE InquireFaceName        (    Handle,
                                      ElementNum : INTEGER;
                                  VAR Name       : ARRAY OF INTEGER )
                                 : INTEGER;

(* inquire face name and index *)
(* Name requires a minimum length of 32 elements *)


PROCEDURE InquireCellArray       (    Handle      : INTEGER;
                                      PxyArray    : VDIRectangle;
```

```
                                    RowLength,
                                    NumRows     : INTEGER;
                                VAR ElUsed,
                                    RowsUsed,
                                    Status      : INTEGER;
                                VAR ColArray    : ARRAY OF INTEGER );

(* inquire cell array *)
(* ColArray requires a minimum length of ?? elements *)

PROCEDURE InquireInputMode      (   Handle,
                                    DevType     : INTEGER;
                                 VAR InputMode  : INTEGER );

(* inquire input mode *)


PROCEDURE InquireFaceInfo       (   Handle      : INTEGER;
                                 VAR MinADE,
                                    MaxADE      : INTEGER;
                                 VAR Distances  : ARRAY OF INTEGER;
                                 VAR MaxWidth   : INTEGER;
                                 VAR Effects    : ARRAY OF INTEGER );

(* inquire current face information *)
(* Distances requires a minimum length of 5 elements *)
(* Effects requires a minimum length of 3 elements *)


END VDIInquires.
```

```
(*----- Category : Module Identification -------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIOutputs;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract -------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987          *)


(*----- Category : Types and Data --------------------------------------*)

FROM VDIAttributes      IMPORT  ColourRange,
                                TenthDegree;

TYPE
        Coordinate              = RECORD
          x, y                  : INTEGER;
                                END;

        VDIRectangle            = RECORD
          LowerLeft,
          UpperRight            : Coordinate;
                                END;


(* Output functions *)

PROCEDURE PolyLine      (   Handle,
                            Count       : INTEGER;
                        VAR PxyArray    : ARRAY OF Coordinate );

(* display a polyline on graphics display *)


PROCEDURE PolyMarker    (   Handle,
                            Count       : INTEGER;
                        VAR PxyArray    : ARRAY OF Coordinate );

(* draw markers at points specified *)


PROCEDURE GraphicText   (   Handle      : INTEGER;
                            Location    : Coordinate;
                        VAR String      : ARRAY OF CHAR );

(* write text to display surface *)


PROCEDURE FillArea      (   Handle,
                            Count       : INTEGER;
                        VAR PxyArray    : ARRAY OF Coordinate );

(* fill a complex polygon *)
```

```
PROCEDURE CellArray     (   Handle      : INTEGER;
                            PxyArray    : VDIRectangle;
                            RowLength,
                            ElUsed,
                            NumRows,
                            WrtMode     : INTEGER;
                            ColArray    : ARRAY OF INTEGER );

(* bit tricky to explain ... see documentation *)

PROCEDURE ContourFill   (   Handle      : INTEGER;
                            StartLocat  : Coordinate;
                            Index       : ColourRange );

(* fill an area *)

PROCEDURE FillRectangle (   Handle      : INTEGER;
                            PxyArray    : VDIRectangle );

(* fill rectangle *)

(* generalised drawing primitives *)

PROCEDURE DrawBar       (   Handle      : INTEGER;
                            PxyArray    : VDIRectangle );

PROCEDURE DrawArc       (   Handle      : INTEGER;
                            Center      : Coordinate;
                            Radius      : INTEGER;
                            BegAng,
                            EndAng      : TenthDegree );

PROCEDURE DrawPieSlice  (   Handle      : INTEGER;
                            Center      : Coordinate;
                            Radius      : INTEGER;
                            BegAng,
                            EndAng      : TenthDegree );

PROCEDURE DrawCircle    (   Handle      : INTEGER;
                            Center      : Coordinate;
                            Radius      : INTEGER );

PROCEDURE DrawEllipticalArc (   Handle          : INTEGER;
                                Center          : Coordinate;
                                xRadius,
                                yRadius         : INTEGER;
                                BegAng,
                                EndAng          : TenthDegree );

PROCEDURE DrawEllipticalPie (   Handle          : INTEGER;
                                Center          : Coordinate;
                                xRadius,
                                yRadius         : INTEGER;
                                BegAng,
                                EndAng          : TenthDegree );

PROCEDURE DrawEllipse   (   Handle      : INTEGER;
                            Center      : Coordinate;
                            xRadius,
                            yRadius     : INTEGER );
```

```
PROCEDURE DrawRoundedBox(    Handle     : INTEGER;
                             PxyArray   : VDIRectangle );

PROCEDURE DrawRoundedFilledBox  (    Handle     : INTEGER;
                                     PxyArray   : VDIRectangle );


(* justified graphics text *)

PROCEDURE JustifiedText (     Handle    : INTEGER;
                              StartLocat: Coordinate;
                         VAR String     : ARRAY OF CHAR;
                              Length,
                              WordSpace,
                              CharSpace : INTEGER );

(* output text both left and right justified *)

END VDIOutputs.
```

```
(*----- Category : Module Identification --------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) VDIRasters;
(* .      Function      : *)
(* .      Version/Date  : 1.02 / 19.07.1987                              *)
(* Product Name         : SPC                                          *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe      *)


(*----- Category : Module Abstract ---------------------------------------**

This module implements the MODULA-2 interface to GEM VDI. All functions
are explained in various documents about GEM on the ATARI ST computer.
However, some datatypes are declared as MODULA-2 records or enumeration
types to simplify access to them. They will be explained as necessary.
Suggested reading: Jankowski, Reschke, Rabich "ATARI ST Profibuch",
                   SYBEX, ISBN 3-88745-501-0, 1.Auflage 1987           *)


(*----- Category : Types and Data ----------------------------------------*)

FROM VDIAttributes      IMPORT  ColourRange,
                                WritingModes;

FROM VDIOutputs         IMPORT  VDIRectangle,
                                Coordinate;

FROM SYSTEM IMPORT ADDRESS;

TYPE
        MemoryFormDefBlock      = RECORD
          UpperLeftRasterAddr   : ADDRESS;
          Width                 : CARDINAL;     (* in points *)
          Height                : CARDINAL;     (* in points *)
          WordWidth             : CARDINAL;     (* in words *)
          FormatFlag            : CARDINAL;     (* 0 = device specific
                                                   1 = standard     *)
          Planes                : CARDINAL;
          Reserved1,
          Reserved2,
          Reserved3             : CARDINAL;     (* for future use *)
                                END;

        MFDBAddress             = POINTER TO MemoryFormDefBlock;

        LogicModes              = ( ClearD     , SAndD   , SAndNotD, S,
                                    NotSAndD   , D       , SXorD   , SOrD,
                                    Nor        , NXor    , NotD    , SOrNotD,
                                    NotS       , NotSOrD, Nand    , SetD);
                                  (* S - soure, D - destination *)

 (* raster operations *)

 PROCEDURE CopyRasterOpaque      (    Handle     : INTEGER;
                                      WrMode     : LogicModes;
                                      FromArea,
                                      ToArea     : VDIRectangle;
                                      SourceMFDB,
                                      DestMFDB   : MFDBAddress );

 (* copy raster, opaque *)


 PROCEDURE CopyRasterTransparent (    Handle     : INTEGER;
```

```
                                    WrMode      : WritingModes;
                                    FromArea,
                                    ToArea      : VDIRectangle;
                                    SourceMFDB,
                                    DestMFDB    : MFDBAddress;
                                    OnColour,
                                    OffColour   : ColourRange);

(* copy raster, transparent *)


PROCEDURE TransformForm         (   Handle      : INTEGER;
                                    SourceMFDB,
                                    DestMFDB    : MFDBAddress );

(* transform form *)


PROCEDURE GetPixel              (   Handle      : INTEGER;
                                    PixelLocat  : Coordinate;
                                VAR PixelValue  : INTEGER;
                                VAR ColourIndex : ColourRange );

(* get pixel *)

END VDIRasters.
```

```
(*----- Category : Module Identification ------------------------------*)

(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) Watch;
(* .      Function      : SPC Desktop Clock                              *)
(* .      Version  Date : 24:33  19. 8.1988                              *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1987,1988, MODsoft, D7500 Karlsruhe        *)


(*----- Category : Initialisation -------------------------------------*)

PROCEDURE Init;

(* Initialise Watch and register at SWiS. *)


PROCEDURE Term;

(* Terminate Watch and deregister from SSWiS. *)

END Watch.
```

```
(*----- Category : Module Identification --------------------------------*)
(* Module Type          : *) DEFINITION MODULE
(* .      Name          : *) XBios;
(* .      Function      : XBIOS Interface TO Modula-2                    *)
(* .      Version/Date  : 1.0 / 13.9.88                                  *)
(* Product Name         : SPC                                            *)
(* Copyright            : (c) 1988, MODsoft, D7500 Karlsruhe             *)


(*----- Category : Module Abstract --------------------------------------**

This module implements the MODULA-2 interface to XBIOS. All functions
are explained in various documents about GEM on the ATARI ST computer.  *)


(*----- Category : Types and Data ----------------------------------------*)

FROM    SYSTEM  IMPORT  ADDRESS;

TYPE    MouseTypes      = ( DisableMouse,
                            RelativeMode,
                            AbsoluteMode,
                            Unused,
                            KeycodeMode );

        MouseParams     = POINTER TO RECORD
           TopMode      :    [0..1];
           Buttons      :    CHAR;
           XParam       :    CHAR;
           YParam       :    CHAR;
           XMax         :    INTEGER;
           YMax         :    INTEGER;
           XInitial     :    INTEGER;
           YInitial     :    INTEGER;
                        END;
        ScreenRes       = ( Low, Medium, High );

        Palette         = ARRAY [0..15] OF INTEGER;

        IoRec           = RECORD
           Buffer       :    ADDRESS;
           BufSize      :    INTEGER;
           Head         :    INTEGER;
           Tail         :    INTEGER;
           Low          :    INTEGER;
           High         :    INTEGER;
                        END;
        IoRecPtr        = POINTER TO IoRec;

        KeyTab          = RECORD
           UnShift      :    ADDRESS;
           Shift        :    ADDRESS;
           CapsLock     :    ADDRESS;
                        END;
        KeyTabPtr       = POINTER TO KeyTab;

        KeyVecs         = RECORD
            MidiIn      :    ADDRESS;
            KbdErr      :    ADDRESS;
            MidiErr     :    ADDRESS;
            StatPack    :    ADDRESS;
            MousePack   :    ADDRESS;
            ClockPack   :    ADDRESS;
            JoyPack     :    ADDRESS;
            MidiSys     :    ADDRESS;
```

```
            KbdSys      :    ADDRESS;
                         END;
        KeyVecsPtr      = POINTER TO KeyVecs;

        PrintParmBlock  = RECORD
            ScreenAdd   :    ADDRESS;
            ScreenOffset:    INTEGER;
            ScreenWidth :    INTEGER;
            ScreenHeigth:    INTEGER;
            Left        :    INTEGER;
            Right       :    INTEGER;
            Resolution  :    ScreenRes;
            PrinterRes  :    INTEGER;
            ColorTab    :    POINTER TO Palette;
            PrinterType :    LONGINT;
            PrinterPort :    INTEGER;
            PrintMask   :    INTEGER;
                         END;
        PrintParmPtr    = POINTER TO PrintParmBlock;

VAR     Result          : LONGINT;


(*----- Category : XBios Functions ------------------------------------*)


PROCEDURE InitMouse     (    MouseType  : MouseTypes;
                             Parms      : MouseParams;
                             Vector     : ADDRESS );

PROCEDURE PhysBase      () : ADDRESS;

PROCEDURE LogBase       () : ADDRESS;

PROCEDURE GetResolution () : ScreenRes;

PROCEDURE SetScreen     (    Log        : ADDRESS;
                             Phys       : ADDRESS;
                             Res        : ScreenRes );

PROCEDURE SetPalette    (VAR P          : Palette );

PROCEDURE SetColor      (    Nummer     : INTEGER;
                             Color      : INTEGER )
                        : INTEGER;

PROCEDURE FlopRead      (    Buffer     : ADDRESS;
                             Device     : INTEGER;
                             Sector     : INTEGER;
                             Track      : INTEGER;
                             Side       : INTEGER;
                             Count      : INTEGER );

PROCEDURE FlopWrite     (    Buffer     : ADDRESS;
                             Device     : INTEGER;
                             Sector     : INTEGER;
                             Track      : INTEGER;
                             Side       : INTEGER;
                             Count      : INTEGER );

PROCEDURE FlopFormat    (    Buffer     : ADDRESS;
                             Device     : INTEGER;
                             SecPerTrack: INTEGER;
                             Track      : INTEGER;
                             Side       : INTEGER;
```

```
                              Interleave : INTEGER;
                              MagicWord  : LONGINT;
                              Virgin     : CARDINAL );

PROCEDURE MidiWrite     (     Count      : INTEGER;
                              Str        : ARRAY OF CHAR );

PROCEDURE MFPInt        (     IntNr      : INTEGER;
                              Int        : ADDRESS );

PROCEDURE GetIoRec      (     Device     : INTEGER )
                        : IoRecPtr;

PROCEDURE RSConfig      (     Speed      : INTEGER;
                              Control    : INTEGER;
                              ucr        : INTEGER;
                              rsr        : INTEGER;
                              tsr        : INTEGER;
                              scr        : INTEGER );

PROCEDURE KeyTable      (     UnShift    : ARRAY OF CHAR;
                              Shift      : ARRAY OF CHAR;
                              CapsLock   : ARRAY OF CHAR )
                        : KeyTabPtr;

PROCEDURE Random        () : LONGINT;

PROCEDURE Protobt       (     Buffer     : ADDRESS;
                              SerialNr   : LONGINT;
                              DiskType   : INTEGER;
                              ExecFlag   : INTEGER );

PROCEDURE FlopVerify    (     Buffer     : ADDRESS;
                              Device     : INTEGER;
                              Sector     : INTEGER;
                              Track      : INTEGER;
                              Side       : INTEGER;
                              Count      : INTEGER );

PROCEDURE ScreenDump;

PROCEDURE CursorConfig  (     Func       : INTEGER;
                              Rate       : INTEGER )
                        : INTEGER;

PROCEDURE SetTime       (     Time : LONGINT );

PROCEDURE GetTime       () : LONGINT;

PROCEDURE BiosKeys;

PROCEDURE KeyboardWrite (     Count      : INTEGER;
                              Str        : ARRAY OF CHAR );

PROCEDURE DisableInt    (     Nr         : INTEGER );

PROCEDURE EnableInt     (     Nr         : INTEGER );

PROCEDURE Giaccess      (     Data       : CHAR;
                              Register   : INTEGER )
                        : CHAR;

PROCEDURE Offgibit      (     Nr         : INTEGER );

PROCEDURE Ongibit       (     Nr         : INTEGER );
```

```
PROCEDURE Xbtimer       (    Timer       : INTEGER;
                             Control     : INTEGER;
                             Data        : INTEGER;
                             Vector      : ADDRESS );

PROCEDURE DoSound       (    Str         : ARRAY OF CHAR );

PROCEDURE SetPrinter    (    Config      : INTEGER )
                        : INTEGER;

PROCEDURE KeyBase       () : KeyVecsPtr;

PROCEDURE KbdRate       (    Init        : CHAR;
                             Repeat      : CHAR )
                        : INTEGER;

PROCEDURE PrintBlock    (    Parms       : PrintParmPtr );

PROCEDURE VSync;

PROCEDURE Supexec       (    Vector      : ADDRESS );

PROCEDURE PuntAES;

PROCEDURE BlitterMode   (    Flag        : INTEGER )
                        : INTEGER;

(* BlitterMode function only in Blitter-TOS *)

END XBios.
```

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Name	Modul	Typ	Lib	Seite
Assign	Strings	PROCEDURE	STDLIB	F-27
AssignFont	TextWindows	TYPE	SPCLIB	G-42
Attribute	GemDos	PROCEDURE	SYSLIB	H-34
Attributes	GemDos	TYPE	SYSLIB	H-31
Attributes	Printer	TYPE	STDLIB	F-22
AttributeSet	Printer	TYPE	STDLIB	F-22
AuxDescr	System	TYPE	SPCLIB	G-34
AuxDescrPtrt	System	TYPE	SPCLIB	G-34
AuxIn	GemDos	PROCEDURE	SYSLIB	H-32
AuxInStat	GemDos	PROCEDURE	SYSLIB	H-33
AuxOut	GemDos	PROCEDURE	SYSLIB	H-32
AuxOutStat	GemDos	PROCEDURE	SYSLIB	H-33
Available	Process	PROCEDURE	SPCLIB	G-18
Available	Storage	PROCEDURE	STDLIB	F-26
Bar	AESMenus	PROCEDURE	SYSLIB	H-13
BasePage	Gemdos	TYPE	SYSLIB	H-31
BasePagePtr	GemDos	TYPE	SYSLIB	H-31
BasePagePtr	System	VAR	SPCLIB	G-33
Bconin	Bios	PROCEDURE	SYSLIB	H-28
Bconout	Bios	PROCEDURE	SYSLIB	H-28
Bconstat	Bios	PROCEDURE	SYSLIB	H-28
Bcostat	Bios	PROCEDURE	SYSLIB	H-28
BeginOfFile	Frags	CONST	SPCLIB	G-10
BEL	ASCII	CONST	STDLIB	F-03
BiosKeys	XBios	PROCEDURE	SYSLIB	H-68
BiosParmBlock	Bios	TYPE	SYSLIB	H-27
BiosParmPtr	Bios	TYPE	SYSLIB	H-27
BitBlk	AESObjects	TYPE	SYSLIB	H-16
BitBlkPtr	AESObjects	TYPE	SYSLIB	H-16
BitBlt	LineA	PROCEDURE	SYSLIB	H-39
BitBltPointer	LineA	TYPE	SYSLIB	H-37
BitBltRecord	LineA	TYPE	SYSLIB	H-37
Black	AESBase	CONST	SYSLIB	H-05
BlitterMode	XBios	PROCEDURE	SYSLIB	H-69
BlockModes	Bios	TYPE	SYSLIB	H-27
Blue	AESBase	CONST	SYSLIB	H-05
BS	ASCII	CONST	STDLIB	F-03
BusyRead	Terminal	PROCEDURE	SPCLIB	Resu
Button	AESEvents	PROCEDURE	SYSLIB	H-07
ButtonActivities	SSWiS	TYPE	SPCLIB	G-25
BYTE	SYSTEM	TYPE	SYSTEM	E-03
Calc	AESWindows	TYPE	SYSLIB	H-24
Call	JCL	PROCEDURE	SPCLIB	G-14
Call	Loader	PROCEDURE	SPCLIB	G-16
CallerOf	System	PROCEDURE	SPCLIB	G-36
CallResult	AESBase	VAR	SYSLIB	H-06
CallVDI	VDIBase	PROCEDURE	SYSLIB	H-45
CAN	ASCII	CONST	STDLIB	F-03
Cardinal	XStr	PROCEDURE	SPCLIB	G-46

Name	Modul	Typ	Lib	Seite
CardToString	NumberConversions	PROCEDURE	STDLIB	F-20
Cd	JCL	PROCEDURE	SPCLIB	G-14
CellArray	VDIOutputs	PROCEDURE	SYSLIB	H-61
Center	AESForms	PROCEDURE	SYSLIB	H-09
Change	AESObjects	PROCEDURE	SYSLIB	H-18
ChangeFileName	VDIEscapes	PROCEDURE	SYSLIB	H-52
Char	XStr	PROCEDURE	SPCLIB	G-46
CharsPerLine	Printer	PROCEDURE	STDLIB	F-23
Clear	Bytes	PROCEDURE	SPCLIB	G-04
Clear	SSWiS	CONST	SPCLIB	G-24
Clear	Strings	PROCEDURE	STDLIB	F-27
Clear	TextWindows	TYPE	SPCLIB	G-43
ClearDisplayList	VDIEscapes	PROCEDURE	SYSLIB	H-51
ClearWorkstation	VDIControls	PROCEDURE	SYSLIB	H-47
Close	AESWindows	TYPE	SYSLIB	H-24
Close	ByteStreams	PROCEDURE	STDLIB	F-04
Close	Files	PROCEDURE	SYSLIB	H-29
Close	FileSystem	PROCEDURE	STDLIB	F-10
Close	GemDos	PROCEDURE	SYSLIB	H-33
Close	TextStreams	PROCEDURE	STDLIB	F-31
CloseInput	InOut	PROCEDURE	STDLIB	F-14
CloseOutput	InOut	PROCEDURE	STDLIB	F-14
CloseVirtualWorkstation	VDIControls	PROCEDURE	SYSLIB	H-47
CloseWorkstation	VDIControls	PROCEDURE	SYSLIB	H-47
ColourComposition	VDIAttributes	TYPE	SYSLIB	H-41
ColourIntensity	VDIAttributes	TYPE	SYSLIB	H-41
ColourRange	VDIAttributes	TYPE	SYSLIB	H-41
Combine	Rectangle	PROCEDURE	SPCLIB	G-19
CombinedWSDescr	VDIInquires	TYPE	SYSLIB	H-57
Compare	Strings	PROCEDURE	STDLIB	F-28
Concat	Strings	PROCEDURE	STDLIB	F-27
ConfigureMouse	SSWiS	PROCEDURE	SPCLIB	G-30
ConIn	GemDos	PROCEDURE	SYSLIB	H-32
ConInStat	GemDos	PROCEDURE	SYSLIB	H-33
ConNegIn	GemDos	PROCEDURE	SYSLIB	H-32
ConOut	GemDos	PROCEDURE	SYSLIB	H-32
ConOutStat	GemDos	PROCEDURE	SYSLIB	H-33
ConRawIn	GemDos	PROCEDURE	SYSLIB	H-32
ConRawIO	GemDos	PROCEDURE	SYSLIB	H-32
ConReadString	GemDos	PROCEDURE	SYSLIB	H-33
Consume	Process	PROCEDURE	SPCLIB	G-18
ContentOf	Fraqs	PROCEDURE	SPCLIB	G-12
ContentOf	TextFiles	PROCEDURE	SPCLIB	G-39
ContourFill	VDIOutputs	PROCEDURE	SYSLIB	H-61
contrl	VDIBase	VAR	SYSLIB	H-45
ControlKeys	SSWiS	TYPE	SPCLIB	G-24
ConWriteString	GemDos	PROCEDURE	SYSLIB	H-33
Coordinate	VDIOutputs	TYPE	SYSLIB	H-60
Coordinates	SSWiS	TYPE	SPCLIB	G-23
Coordinates	TextWindows	TYPE	SPCLIB	G-41
CoordinateTypes	VDIAttributes	TYPE	SYSLIB	H-42

Name	Modul	Typ	Lib	Seite
Copy	Strings	PROCEDURE	STDLIB	F-28
CopyFor	Bytes	PROCEDURE	SPCLIB	G-04
CopyForWhile	XStr	PROCEDURE	SPCLIB	G-45
CopyRasterOpaque	VDIRasters	PROCEDURE	SYSLIB	H-63
CopyRasterTransparent	VDIRasters	PROCEDURE	SYSLIB	H-63
CopyWhileIn	Bytes	PROCEDURE	SPCLIB	G-04
CopyWhileNot	Bytes	PROCEDURE	SPCLIB	G-04
CopyWhileNotIn	Bytes	PROCEDURE	SPCLIB	G-04
cos	LMathLib	PROCEDURE	STDLIB	F-17
cos	MathLib	PROCEDURE	STDLIB	F-19
cosh	LMathLib	PROCEDURE	STDLIB	F-17
cosh	MathLib	PROCEDURE	STDLIB	F-19
cot	LMathLib	PROCEDURE	STDLIB	F-17
cot	MathLib	PROCEDURE	STDLIB	F-19
Cp	JCL	PROCEDURE	SPCLIB	G-14
CR	ASCII	CONST	STDLIB	F-03
Create	AESWindows	PROCEDURE	SYSLIB	H-24
Create	Files	PROCEDURE	SYSLIB	H-29
Create	Frags	PROCEDURE	SPCLIB	G-11
Create	GemDos	PROCEDURE	SYSLIB	H-33
Create	Process	PROCEDURE	SPCLIB	G-18
Create	SplittedPieces	PROCEDURE	SPCLIB	G-21
Create	TextFiles	PROCEDURE	SPCLIB	G-38
Create	TextWindows	TYPE	SPCLIB	G-42
CreateWindow	SSWiS	PROCEDURE	SPCLIB	G-27
CtrlArrayType	VDIBase	TYPE	SYSLIB	H-45
Current	Frags	PROCEDURE	SPCLIB	G-11
CurrentFolder	HFS	PROCEDURE	STDLIB	F-12
CurrentLine	Printer	PROCEDURE	STDLIB	F-23
CurrentPage	Printer	PROCEDURE	STDLIB	F-23
CurrentPosition	Printer	PROCEDURE	STDLIB	F-23
CurrExcFrame	System	VAR	SPCLIB	G-37
CurrExcRoot	System	VAR	SPCLIB	G-37
CurrExcType	System	VAR	SPCLIB	G-37
CursorAddress	VDIEscapes	PROCEDURE	SYSLIB	H-50
CursorConfig	XBios	PROCEDURE	SYSLIB	H-68
CursorDown	VDIEscapes	PROCEDURE	SYSLIB	H-49
CursorHome	VDIEscapes	PROCEDURE	SYSLIB	H-50
CursorLeft	VDIEscapes	PROCEDURE	SYSLIB	H-50
CursorRight	VDIEscapes	PROCEDURE	SYSLIB	H-49
Cursors	AESGraphics	TYPE	SYSLIB	H-11
CursorUp	VDIEscapes	PROCEDURE	SYSLIB	H-49
Cyan	AESBase	CONST	SYSLIB	H-05
DataPtr	AltResource	TYPE	SYSLIB	H-26
DC1	ASCII	CONST	STDLIB	F-03
DC2	ASCII	CONST	STDLIB	F-03
DC3	ASCII	CONST	STDLIB	F-03
DC4	ASCII	CONST	STDLIB	F-03
Deallocate	Storage	PROCEDURE	STDLIB	F-26
DEALLOCATE	Storage	PROCEDURE	STDLIB	F-26

Name	Modul	Typ	Lib	Seite
ExchangeButtonV	VDIInputs	PROCEDURE	SYSLIB	H-55
ExchangeCursorV	VDIInputs	PROCEDURE	SYSLIB	H-55
ExchangeMovementV	VDIInputs	PROCEDURE	SYSLIB	H-55
ExchangeTimerV	VDIInputs	PROCEDURE	SYSLIB	H-55
Excl	Bytes	PROCEDURE	SPCLIB	G-05
ExcTypes	System	TYPE	SPCLIB	G-36
Exec	GemDos	PROCEDURE	SYSLIB	H-34
Exists	JCL	PROCEDURE	SPCLIB	G-14
Exit	AESApplications	PROCEDURE	SYSLIB	H-03
ExitAlphaMode	VDIEscapes	PROCEDURE	SYSLIB	H-49
exp	LMathLib	PROCEDURE	STDLIB	F-17
exp	MathLib	PROCEDURE	STDLIB	F-19
ExplicitRestore	SSWiS	PROCEDURE	SPCLIB	G-28
ExplicitRestore	TextWindows	TYPE	SPCLIB	G-42
Expose	Terminal	PROCEDURE	STDLIB	F-29
ExtendCoords	VDIInquires	TYPE	SYSLIB	H-57
ExtendedInquire	VDIInquires	PROCEDURE	SYSLIB	H-57
ExtendWSDescription	VDIInquires	TYPE	SYSLIB	H-56
F1	SSWiS	CONST	SPCLIB	G-24
F10	SSWiS	CONST	SPCLIB	G-24
F2	SSWiS	CONST	SPCLIB	G-24
F3	SSWiS	CONST	SPCLIB	G-24
F4	SSWiS	CONST	SPCLIB	G-24
F5	SSWiS	CONST	SPCLIB	G-24
F6	SSWiS	CONST	SPCLIB	G-24
F7	SSWiS	CONST	SPCLIB	G-24
F8	SSWiS	CONST	SPCLIB	G-24
F9	SSWiS	CONST	SPCLIB	G-24
FABSd	System	PROCEDURE	SPCLIB	G-34
FABSs	System	PROCEDURE	SPCLIB	G-33
FADDd	System	PROCEDURE	SPCLIB	G-33
FADDs	System	PROCEDURE	SPCLIB	G-33
FCMPd	System	PROCEDURE	SPCLIB	G-34
FCMPs	System	PROCEDURE	SPCLIB	G-33
FDIVd	System	PROCEDURE	SPCLIB	G-33
FDIVs	System	PROCEDURE	SPCLIB	G-33
FF	ASCII	CONST	STDLIB	F-03
File	Files	TYPE	SYSLIB	H-29
File	FileSystem	TYPE	STDLIB	F-09
File	Frags	TYPE	SPCLIB	G-10
File	TextFiles	TYPE	SPCLIB	G-38
FileArg	CmdLine	PROCEDURE	SPCLIB	G-06
FileProc	HFS	TYPE	STDLIB	F-12
FileSelectorInput	AESForms	PROCEDURE	SYSLIB	H-10
FileTimes	GemDos	TYPE	SYSLIB	H-32
FileTypes	Environment	TYPE	SPCLIB	G-08
FillArea	VDIOutputs	PROCEDURE	SYSLIB	H-60
FillAttrType	VDIInquires	TYPE	SYSLIB	H-57
FillPolygon	LineA	PROCEDURE	SYSLIB	H-39
FillRange	VDIAttributes	TYPE	SYSLIB	H-41

Name	Modul	Typ	Lib	Seite
GetAddr	AltResource	PROCEDURE	SYSLIB	H-26
Getbpb	Bios	PROCEDURE	SYSLIB	H-28
GetDate	GemDos	PROCEDURE	SYSLIB	H-33
GetDrive	GemDos	PROCEDURE	SYSLIB	H-33
GetDTA	GemDos	PROCEDURE	SYSLIB	H-33
GetFilename	Environment	PROCEDURE	SPCLIB	G-08
GetFileType	Environment	PROCEDURE	SPCLIB	G-09
GetFont	LineA	PROCEDURE	SYSLIB	H-38
GetIndexed	Environment	PROCEDURE	SPCLIB	G-08
GetIoRec	XBios	PROCEDURE	SYSLIB	H-68
GetModes	GemDos	TYPE	SYSLIB	H-32
Getmpb	Bios	PROCEDURE	SYSLIB	H-28
GetName	Printer	PROCEDURE	STDLIB	F-24
GetPath	GemDos	PROCEDURE	SYSLIB	H-34
GetPixel	LineA	PROCEDURE	SYSLIB	H-38
GetPixel	VDIRasters	PROCEDURE	SYSLIB	H-64
GetPos	Files	PROCEDURE	SYSLIB	H-30
GetPos	FileSystem	PROCEDURE	STDLIB	F-10
GetResolution	XBios	PROCEDURE	SYSLIB	H-67
GetTime	GemDos	PROCEDURE	SYSLIB	H-33
GetTime	XBios	PROCEDURE	SYSLIB	H-68
Giaccess	XBios	PROCEDURE	SYSLIB	H-68
GraphicText	VDIOutputs	PROCEDURE	SYSLIB	H-60
Greater	Strings	TYPE	STDLIB	F-27
Green	AESBase	CONST	SYSLIB	H-05
GrowBox	AESGraphics	PROCEDURE	SYSLIB	H-12
GS	ASCII	CONST	STDLIB	F-03
HALTX	System	PROCEDURE	SPCLIB	G-33
Handle	AESGraphics	PROCEDURE	SYSLIB	H-12
Handles	GemDos	TYPE	SYSLIB	H-32
HardCopy	VDIEscapes	PROCEDURE	SYSLIB	H-50
Head	Printer	PROCEDURE	STDLIB	F-23
HeadProc	Printer	TYPE	STDLIB	F-22
Help	SSWiS	CONST	SPCLIB	G-24
Hide	Terminal	PROCEDURE	STDLIB	F-29
HideCursor	VDIInputs	PROCEDURE	SYSLIB	H-55
HideMouse	LineA	PROCEDURE	SYSLIB	H-39
HorAlignment	VDIAttributes	TYPE	SYSLIB	H-41
HorizLine	LineA	PROCEDURE	SYSLIB	H-38
HourGlass	AESGraphics	CONST	SYSLIB	H-11
HT	ASCII	CONST	STDLIB	F-03
IconBlk	AESObjects	TYPE	SYSLIB	H-16
IconBlkPtr	AESObjects	TYPE	SYSLIB	H-16
IconiseWindow	SSWiS	PROCEDURE	SPCLIB	G-27
Identify	SSWiS	PROCEDURE	SPCLIB	G-31
Identify	TextWindows	TYPE	SPCLIB	G-43
Incl	Bytes	PROCEDURE	SPCLIB	G-05
Includes	Rectangle	PROCEDURE	SPCLIB	G-20
Init	LMathLib	PROCEDURE	STDLIB	F-17

Name	Modul	Typ	Lib	Seite
InvLongcard	XStr	PROCEDURE	SPCLIB	G-47
InvLongint	XStr	PROCEDURE	SPCLIB	G-47
InvLongreal	XStr	PROCEDURE	SPCLIB	G-47
InvReal	XStr	PROCEDURE	SPCLIB	G-47
IoRec	XBios	TYPE	SYSLIB	H-66
IoRecPtr	XBios	TYPE	SYSLIB	H-66
IOTRANSFER	Coroutines	PROCEDURE	STDLIB	F-08
ItemCheck	AESMenus	PROCEDURE	SYSLIB	H-13
ItemEnable	AESMenus	PROCEDURE	SYSLIB	H-13
ItemHandles	SSWiS	TYPE	SPCLIB	G-23
Items	AESWindows	TYPE	SYSLIB	H-23
ItemStruc	AESWindows	TYPE	SYSLIB	H-23
ItemTypes	AESResources	TYPE	SYSLIB	H-19
Justifications	AESObjects	TYPE	SYSLIB	H-15
JustifiedText	VDIOutputs	PROCEDURE	SYSLIB	H-62
KbdRate	XBios	PROCEDURE	SYSLIB	H-69
KbdStates	AESEvents	TYPE	SYSLIB	H-07
KeyBase	XBios	PROCEDURE	SYSLIB	H-69
Keyboard	AESEvents	PROCEDURE	SYSLIB	H-07
KeyboardSpecials	VDIInputs	TYPE	SYSLIB	H-53
KeyboardState	VDIInputs	TYPE	SYSLIB	H-53
KeyboardWrite	XBios	PROCEDURE	SYSLIB	H-68
KeyEvent	SSWiS	TYPE	SPCLIB	G-25
KeyShifts	Bios	PROCEDURE	SYSLIB	H-28
KeyTab	XBios	TYPE	SYSLIB	H-66
KeyTable	XBios	PROCEDURE	SYSLIB	H-68
KeyTabPtr	XBios	TYPE	SYSLIB	H-66
KeyVecs	XBios	TYPE	SYSLIB	H-66
KeyVecsPtr	XBios	TYPE	SYSLIB	H-67
KillThread	System	PROCEDURE	SPCLIB	G-36
Label	TextFiles	PROCEDURE	SPCLIB	G-39
Labels	TextFiles	TYPE	SPCLIB	G-38
LeftArrow	SSWiS	CONST	SPCLIB	G-24
LeftBorderSize	Printer	PROCEDURE	STDLIB	F-23
Length	Files	PROCEDURE	SYSLIB	H-30
Length	FileSystem	PROCEDURE	STDLIB	F-10
Length	Strings	PROCEDURE	STDLIB	F-27
LengthOf	Frags	PROCEDURE	SPCLIB	G-12
LengthOf	TextFiles	PROCEDURE	SPCLIB	G-39
Less	Strings	TYPE	STDLIB	F-27
LF	ASCII	CONST	STDLIB	F-03
lg	LMathLib	PROCEDURE	STDLIB	F-17
lg	MathLib	PROCEDURE	STDLIB	F-19
LightBlack	AESBase	CONST	SYSLIB	H-05
LightBlue	AESBase	CONST	SYSLIB	H-05
LightCyan	AESBase	CONST	SYSLIB	H-05
LightGreen	AESBase	CONST	SYSLIB	H-05
LightMagenta	AESBase	CONST	SYSLIB	H-05

Name	Modul	Typ	Lib	Seite
MFDBAddress	VDIRasters	TYPE	SYSLIB	H-63
MFormPointer	LineA	TYPE	SYSLIB	H-37
MFormRecord	LineA	TYPE	SYSLIB	H-37
MFPInt	XBios	PROCEDURE	SYSLIB	H-68
MidiWrite	XBios	PROCEDURE	SYSLIB	H-68
MkDir	JCL	PROCEDURE	SPCLIB	G-14
ModuleDescr	System	TYPE	SPCLIB	G-34
ModuleDescrPtr	System	TYPE	SPCLIB	G-34
ModuleHandles	SSWiS	TYPE	SPCLIB	G-23
ModuleKeys	System	TYPE	SPCLIB	G-34
ModuleNames	System	TYPE	SPCLIB	G-34
Mouse	AESEvents	PROCEDURE	SYSLIB	H-08
Mouse	AESGraphics	PROCEDURE	SYSLIB	H-12
MouseCodes	VDIInputs	TYPE	SYSLIB	H-53
MouseEvent	SSWiS	TYPE	SPCLIB	G-25
MouseFormType	VDIInputs	TYPE	SYSLIB	H-53
MouseKeyboardState	AESGraphics	PROCEDURE	SYSLIB	H-12
MouseOff	AESGraphics	CONST	SYSLIB	H-11
MouseOn	AESGraphics	CONST	SYSLIB	H-11
MouseParams	XBios	TYPE	SYSLIB	H-66
MouseSprites	SSWiS	TYPE	SPCLIB	G-25
MouseState	VDIInputs	TYPE	SYSLIB	H-53
MouseStyles	SSWiS	TYPE	SPCLIB	G-25
MouseTypes	XBios	TYPE	SYSLIB	H-66
MoveAbs	Rectangle	PROCEDURE	SPCLIB	G-19
MoveBox	AESGraphics	PROCEDURE	SYSLIB	H-11
MoveRel	Rectangle	PROCEDURE	SPCLIB	G-19
MULS32	System	PROCEDURE	SPCLIB	G-33
Multiple	AESEvents	PROCEDURE	SYSLIB	H-08
MULU32	System	PROCEDURE	SPCLIB	G-33
Mv	JCL	PROCEDURE	SPCLIB	G-14
NAK	ASCII	CONST	STDLIB	F-03
NameLength	GemDos	CONST	SYSLIB	H-31
NameLength	HFS	VAR	STDLIB	F-12
NationalKeys	SSWiS	TYPE	SPCLIB	G-25
NeverClip	SSWiS	VAR	SPCLIB	G-26
NEWPROCESS	Coroutines	PROCEDURE	STDLIB	F-08
Next	Frags	PROCEDURE	SPCLIB	G-11
NextDescriptor	System	PROCEDURE	SPCLIB	G-35
NilKey	SSWiS	CONST	SPCLIB	G-24
NotifyForm	SSWiS	PROCEDURE	SPCLIB	G-31
NUL	ASCII	CONST	STDLIB	F-03
NullLine	SSWiS	VAR	SPCLIB	G-26
NullPoint	SSWiS	VAR	SPCLIB	G-26
NullScreenLine	SSWiS	VAR	SPCLIB	G-26
NullScreenPoint	SSWiS	VAR	SPCLIB	G-26
Num0	SSWiS	CONST	SPCLIB	G-24
Num1	SSWiS	CONST	SPCLIB	G-24
Num2	SSWiS	CONST	SPCLIB	G-24
Num3	SSWiS	CONST	SPCLIB	G-24

Name	Modul	Typ	Lib	Seite
ParamBlk	AESObjects	TYPE	SYSLIB	H-17
ParamBlkPtr	AESObjects	TYPE	SYSLIB	H-16
parameterBlock	VDIBase	VAR	SYSLIB	H-45
Paths	GemDos	TYPE	SYSLIB	H-32
Phases	AESForms	TYPE	SYSLIB	H-09
PhysBase	XBios	PROCEDURE	SYSLIB	H-67
pi	LMathLib	CONST	STDLIB	F-17
pi	MathLib	CONST	STDLIB	F-19
PlaceWindowOnTop	SSWiS	PROCEDURE	SPCLIB	G-27
PlainText	Frags	CONST	SPCLIB	G-10
Pline	LineA	PROCEDURE	SYSLIB	H-38
PointerOf	Frags	PROCEDURE	SPCLIB	G-12
PointerOf	TextFiles	PROCEDURE	SPCLIB	G-40
PointHand	AESGraphics	CONST	SYSLIB	H-11
Points	SSWiS	TYPE	SPCLIB	G-23
Points	TextWindows	TYPE	SPCLIB	G-41
PollEvents	SSWiS	PROCEDURE	SPCLIB	G-27
PolyLine	VDIOutputs	PROCEDURE	SYSLIB	H-60
PolyMarker	VDIOutputs	PROCEDURE	SYSLIB	H-60
Pos	Strings	PROCEDURE	STDLIB	F-28
Position	TextFiles	PROCEDURE	SPCLIB	G-39
Position	TextWindows	TYPE	SPCLIB	G-43
PositionOf	TextFiles	PROCEDURE	SPCLIB	G-39
PositionOf	TextWindows	TYPE	SPCLIB	G-43
PositionOfWindow	SSWiS	PROCEDURE	SPCLIB	G-28
PositionOfWorld	SSWiS	PROCEDURE	SPCLIB	G-29
PositionWindow	SSWiS	PROCEDURE	SPCLIB	G-28
PositionWorld	SSWiS	PROCEDURE	SPCLIB	G-29
PositionWorld	TextWindows	TYPE	SPCLIB	G-42
Preset	Rectangle	PROCEDURE	SPCLIB	G-19
Prev	Frags	PROCEDURE	SPCLIB	G-11
PrintBlock	XBios	PROCEDURE	SYSLIB	H-69
PrintParmBlock	XBios	TYPE	SYSLIB	H-67
PrintParmPtr	XBios	TYPE	SYSLIB	H-67
PrnOut	GemDos	PROCEDURE	SYSLIB	H-32
PrnOutStat	GemDos	PROCEDURE	SYSLIB	H-33
ProcedureFrames	System	TYPE	SPCLIB	G-36
Produce	Process	PROCEDURE	SPCLIB	G-18
Protobt	XBios	PROCEDURE	SYSLIB	H-68
ptsin	VDIBase	VAR	SYSLIB	H-45
ptsout	VDIBase	VAR	SYSLIB	H-45
PuntAES	XBios	PROCEDURE	SYSLIB	H-69
Put	SplittedPieces	PROCEDURE	SPCLIB	G-21
PutPixel	LineA	PROCEDURE	SYSLIB	H-38
Quantity	Storage	TYPE	STDLIB	F-26
Query	JCL	PROCEDURE	SPCLIB	G-13
Random	XBios	PROCEDURE	SYSLIB	H-68
RasterWorld	SSWiS	PROCEDURE	SPCLIB	G-29
Read	AESApplications	PROCEDURE	SYSLIB	H-03

Name	Modul	Typ	Lib	Seite
SetOfDrives	Bios	TYPE	SYSLIB	H-27
SetOfDrives	GemDos	TYPE	SYSLIB	H-31
SetOfElements	AESWindows	TYPE	SYSLIB	H-23
SetOfEvents	AESEvents	TYPE	SYSLIB	H-07
SetOfFlags	AESObjects	TYPE	SYSLIB	H-15
SetOfKbdStates	AESEvents	TYPE	SYSLIB	H-07
SetOfMetaKeys	SSWiS	TYPE	SPCLIB	G-25
SetOfStates	AESObjects	TYPE	SYSLIB	H-15
SetOfThreadIds	System	TYPE	SPCLIB	G-34
SetOfWindowElements	SSWiS	TYPE	SPCLIB	G-26
SetPalette	XBios	PROCEDURE	SYSLIB	H-67
SetPaletteState	VDIEscapes	PROCEDURE	SYSLIB	H-51
SetPath	GemDos	PROCEDURE	SYSLIB	H-33
SetPointCharHeight	VDIAttributes	PROCEDURE	SYSLIB	H-43
SetPos	Files	PROCEDURE	SYSLIB	H-30
SetPos	FileSystem	PROCEDURE	STDLIB	F-10
SetPrinter	XBios	PROCEDURE	SYSLIB	H-69
SETREG	SYSTEM	PROCEDURE	SYSTEM	E-03
SetRotation	VDIAttributes	PROCEDURE	SYSLIB	H-43
SetScreen	XBios	PROCEDURE	SYSLIB	H-67
SetTime	GemDos	PROCEDURE	SYSLIB	H-33
SetTime	XBios	PROCEDURE	SYSLIB	H-68
SetWindowElements	SSWiS	PROCEDURE	SPCLIB	G-28
SetWindowMessage	SSWiS	PROCEDURE	SPCLIB	G-28
SetWindowTitle	SSWiS	PROCEDURE	SPCLIB	G-28
SetWritingMode	VDIAttributes	PROCEDURE	SYSLIB	H-42
SHIFT	SYSTEM	PROCEDURE	SYSTEM	E-04
SHORT	SYSTEM	PROCEDURE	SYSTEM	E-04
ShowCursor	VDIInputs	PROCEDURE	SYSLIB	H-55
ShowMouse	LineA	PROCEDURE	SYSLIB	H-39
Shrink	GemDos	PROCEDURE	SYSLIB	H-34
ShrinkBox	AESGraphics	PROCEDURE	SYSLIB	H-12
SI	ASCII	CONST	STDLIB	F-03
sin	LMathLib	PROCEDURE	STDLIB	F-17
sin	MathLib	PROCEDURE	STDLIB	F-19
sinh	LMathLib	PROCEDURE	STDLIB	F-17
sinh	MathLib	PROCEDURE	STDLIB	F-19
SIZE	SYSTEM	PROCEDURE	SYSTEM	E-03
SizeOfWindowContent	SSWiS	PROCEDURE	SPCLIB	G-29
SizeOfWorld	SSWiS	PROCEDURE	SPCLIB	G-29
SizeWindowContent	SSWiS	PROCEDURE	SPCLIB	G-28
SizeWorld	SSWiS	PROCEDURE	SPCLIB	G-29
SizeWorld	TextWindows	TYPE	SPCLIB	G-42
SlideBox	AESGraphics	PROCEDURE	SYSLIB	H-12
SO	ASCII	CONST	STDLIB	F-03
SOH	ASCII	CONST	STDLIB	F-03
Space	JCL	PROCEDURE	SPCLIB	G-15
sqrt	LMathLib	PROCEDURE	STDLIB	F-17
sqrt	MathLib	PROCEDURE	STDLIB	F-19
States	AESObjects	TYPE	SYSLIB	H-15
StdIn	GemDos	CONST	SYSLIB	H-31

Name	Modul	Typ	Lib	Seite
StdOut	GemDos	CONST	SYSLIB	H-31
Streams	ByteStreams	TYPE	STDLIB	F-04
Streams	TextStreams	TYPE	STDLIB	F-31
StringToCard	NumberConversions	PROCEDURE	STDLIB	F-20
StringToInt	NumberConversions	PROCEDURE	STDLIB	F-20
StringToLongCard	NumberConversions	PROCEDURE	STDLIB	F-20
StringToLongInt	NumberConversions	PROCEDURE	STDLIB	F-21
StringToLongReal	RealConversions	PROCEDURE	STDLIB	F-25
StringToReal	RealConversions	PROCEDURE	STDLIB	F-25
STX	ASCII	CONST	STDLIB	F-03
SUB	ASCII	CONST	STDLIB	F-03
Sub	Clock	PROCEDURE	STDLIB	F-06
SubResult	Rectangle	PROCEDURE	SPCLIB	G-20
Subtract	Rectangle	PROCEDURE	SPCLIB	G-19
Super	GemDos	PROCEDURE	SYSLIB	H-33
Supexec	XBios	PROCEDURE	SYSLIB	H-69
SupportedAttributes	Printer	PROCEDURE	STDLIB	F-24
SupportedFonts	Printer	PROCEDURE	STDLIB	F-24
SuppressPaletteMessages	VDIEscapes	PROCEDURE	SYSLIB	H-52
SYN	ASCII	CONST	STDLIB	F-03
tan	LMathLib	PROCEDURE	STDLIB	F-17
tan	MathLib	PROCEDURE	STDLIB	F-19
tanh	LMathLib	PROCEDURE	STDLIB	F-17
tanh	MathLib	PROCEDURE	STDLIB	F-19
TedInfo	AESObjects	TYPE	SYSLIB	H-16
TedInfoPtr	AESObjects	TYPE	SYSLIB	H-16
TenthDegree	VDIAttributes	TYPE	SYSLIB	H-41
Term	GemDos	PROCEDURE	SYSLIB	H-34
Term	Printer	PROCEDURE	STDLIB	F-24
Term	Process	PROCEDURE	SPCLIB	G-17
Term	Watch	PROCEDURE	SYSLIB	H-65
TermCh	InOut	VAR	STDLIB	F-14
TermO	GemDos	PROCEDURE	SYSLIB	H-32
TermProc	XStr	TYPE	SPCLIB	G-45
TermResident	GemDos	PROCEDURE	SYSLIB	H-33
Test	Frags	PROCEDURE	SPCLIB	G-12
Text	AESMenus	PROCEDURE	SYSLIB	H-13
Text	Frags	TYPE	SPCLIB	G-10
Text	TextFiles	TYPE	SPCLIB	G-38
TextAttrType	VDIInquires	TYPE	SYSLIB	H-57
TextBlt	LineA	PROCEDURE	SYSLIB	H-39
TextCursor	AESGraphics	CONST	SYSLIB	H-11
TextEffect	VDIAttributes	TYPE	SYSLIB	H-41
TextEffects	VDIAttributes	TYPE	SYSLIB	H-41
TextOfExc	System	PROCEDURE	SPCLIB	G-37
TextPtr	Frags	TYPE	SPCLIB	G-10
TextPtr	TextFiles	TYPE	SPCLIB	G-38
ThickCross	AESGraphics	CONST	SYSLIB	H-11
ThinCross	AESGraphics	CONST	SYSLIB	H-11
ThreadIds	System	TYPE	SPCLIB	G-34

Name	Modul	Typ	Lib	Seite
TickCal	Bios	PROCEDURE	SYSLIB	H-28
Time	Clock	TYPE	STDLIB	F-06
Timer	AESEvents	PROCEDURE	SYSLIB	H-08
TimeStamp	Files	PROCEDURE	SYSLIB	H-30
Timestamp	GemDos	PROCEDURE	SYSLIB	H-34
Timestamp	JCL	PROCEDURE	SPCLIB	G-14
TitleHandles	SSWiS	TYPE	SPCLIB	G-23
TitleNormal	AESMenus	PROCEDURE	SYSLIB	H-13
TotalLinesOf	TextFiles	PROCEDURE	SPCLIB	G-40
TPlayback	AESApplications	PROCEDURE	SYSLIB	H-03
TRANSFER	Coroutines	PROCEDURE	STDLIB	F-08
TransformForm	VDIRasters	PROCEDURE	SYSLIB	H-64
TransformMouse	LineA	PROCEDURE	SYSLIB	H-39
TRecord	AESApplications	PROCEDURE	SYSLIB	H-03
TreePtr	AESObjects	TYPE	SYSLIB	H-16
TRUNCd	System	PROCEDURE	SPCLIB	G-34
TRUNCs	System	PROCEDURE	SPCLIB	G-33
ts	TextStreams	TYPE	STDLIB	F-31
Tst	Bytes	PROCEDURE	SPCLIB	G-05
TypeLength	GemDos	CONST	SYSLIB	H-31
TypeLength	HFS	VAR	STDLIB	F-12
Types	AESObjects	TYPE	SYSLIB	H-15
Types	ByteStreams	TYPE	STDLIB	F-04
Types	Files	TYPE	SYSLIB	H-29
Types	Frags	TYPE	SPCLIB	G-10
Types	TextStreams	TYPE	STDLIB	F-31
TypeSep	GemDos	CONST	SYSLIB	H-31
TypeSep	HFS	VAR	STDLIB	F-12
Umlauts	SSWiS	VAR	SPCLIB	G-26
Undo	SSWiS	CONST	SPCLIB	G-24
UndrawSprite	LineA	PROCEDURE	SYSLIB	H-39
UnlinkThread	System	PROCEDURE	SPCLIB	G-36
UnloadFonts	VDIControls	PROCEDURE	SYSLIB	H-48
UpArrow	SSWiS	CONST	SPCLIB	G-24
Update	AESWindows	TYPE	SYSLIB	H-24
UpdateFlags	AESWindows	TYPE	SYSLIB	H-23
UpdateMetafileExtents	VDIEscapes	PROCEDURE	SYSLIB	H-52
UpdateWorkstation	VDIControls	PROCEDURE	SYSLIB	H-47
US	ASCII	CONST	STDLIB	F-03
UserDef	AESGraphics	CONST	SYSLIB	H-11
UtilityName	CmdLine	PROCEDURE	SPCLIB	G-06
VAL	SYSTEM	PROCEDURE	SYSTEM	E-04
ValidTextRotations	VDIInquires	TYPE	SYSLIB	H-56
ValuatorStatus	VDIInputs	TYPE	SYSLIB	H-53
VDIDescription	LineA	PROCEDURE	SYSLIB	H-38
VDIRectangle	VDIOutputs	TYPE	SYSLIB	H-60
Version	GemDos	PROCEDURE	SYSLIB	H-33
VertAlignment	VDIAttributes	TYPE	SYSLIB	H-41
VolumeSep	GemDos	CONST	SYSLIB	H-31